

به نام خدا

برنامه‌سازی ۱

رشته‌ی کامپیوتر

گروه تحصیلی کامپیوتر

زمینه‌ی خدمات

شاخه‌ی آموزش فنی و حرفه‌ای

چهارم، علیرضا، ۱۳۴۷ -

برنامه‌سازی ۱، فنی و حرفه‌ای (گروه تحصیلی کامپیوتر)، سال دوم رشته‌ی کامپیوتر / مؤلفان علیرضا چهارم، کامبیز جمع‌نار؛ برنامه‌ریزی محتوا و نظارت بر تألیف دفتر برنامه‌ریزی و تألیف آموزش‌های فنی و حرفه‌ای و کاردانش سازمان پژوهش و برنامه‌ریزی وزارت آموزش و پرورش. - تهران: فاطمی، ۱۳۸۹.
دوازده، ۲۴۳ ص: مصور، جدول.

ISBN 978-964-318-488-9

فهرست‌نویسی بر اساس اطلاعات فنی.

ویاست قبلی این کتاب تحت عنوان «برنامه‌سازی ۱ [یک]، رشته‌ی کامپیوتر، گروه تحصیلی کامپیوتر، زمینه‌ی خدمات، شاخه‌ی آموزش فنی و حرفه‌ای (اجرای آزمایشی)» توسط انتشارات فاطمی در سال ۱۳۸۴ نیز منتشر شده است.
کتابنامه: ص [۲۴۳].

۱. ویزال بیسیک (زبان برنامه‌نویسی کامپیوتر) - راه‌های آموزشی (متوسطه). ۲. ویزال بیسیک مایکروسافت تحت ویندوز الف. جمع‌نار، کامبیز، ۱۳۵۰ - ب. مؤسسه انتشارات فاطمی. ج. سازمان پژوهش و برنامه‌ریزی آموزشی. دفتر برنامه‌ریزی درسی آموزش‌های فنی و حرفه‌ای و کاردانش. د. عنوان.

--۵/۲۶۸

Q۸۷۶/۷۳/۹ج۱۶۵

۱۳۸۹

۱۳۲۸۸۸۶

کتابخانه‌ی ملی ایران



انتشارات فاطمی

۱۳۹۰

همکاران محترم و دانش‌آموزان عزیز:

پیشنهادات و نظرات خود را در باره‌ی محتوای این کتاب به نشانی تهران - صندوق پستی شماره‌ی ۴۸۷۴/۱۵ دفتر برنامه‌ریزی و تألیف آموزش‌های فنی و حرفه‌ای و کاردانش، ارسال فرمایند.

info@tvoccd.sch.ir

پست الکترونیکی

www.tvoccd.sch.ir

آدرس الکترونیکی

وزارت آموزش و پرورش

سازمان پژوهش و برنامه‌ریزی آموزشی

برنامه‌ریزی محتوا و نظارت بر تألیف: دفتر برنامه‌ریزی و تألیف آموزش‌های فنی و حرفه‌ای و کاردانش
اعضای کمیسیون تخصصی برنامه‌ریزی و تألیف: محمد مشاهری‌فرد، محمدرضا یمقانی، عسگر قندچی،
سید حمیدرضا ضیایی، هادی عابدی، ملیحه طزری، حمید احدی

نام کتاب: برنامه‌سازی ۱ - ۳۵۸/۷۰

مؤلفان: علیرضا جباریه، کامبیز جمعدار

ناشر: انتشارات فاطمی

نوبت چاپ و سال انتشار: چهارم، ۱۳۹۰

شابک ۹۷۸-۹۶۴-۳۱۸-۴۸۸-۹

ISBN 978-964-318-488-9

تیراژ: نسخه

آماده‌سازی پیش از چاپ: واحد تولید انتشارات فاطمی

- مدیر تولید: فرید مصلحی

- طراح جلد: زهرا قورچیان

- حرفچینی و صفحه‌بندی (TEX-پاک): هنگامه صادقی

- نمونه‌خوان: شقایق میرصیافی

- بازسازی تصاویر و صفحه‌آرایی: فاطمه ثقفی

نظارت بر چاپ: اداره‌ی کل چاپ و توزیع کتاب‌های درسی

چاپخانه: شرکت چاپ و نشر کتاب‌های درسی ایران («سهامی خاص»)

کلیه‌ی حقوق برای انتشارات فاطمی محفوظ است.

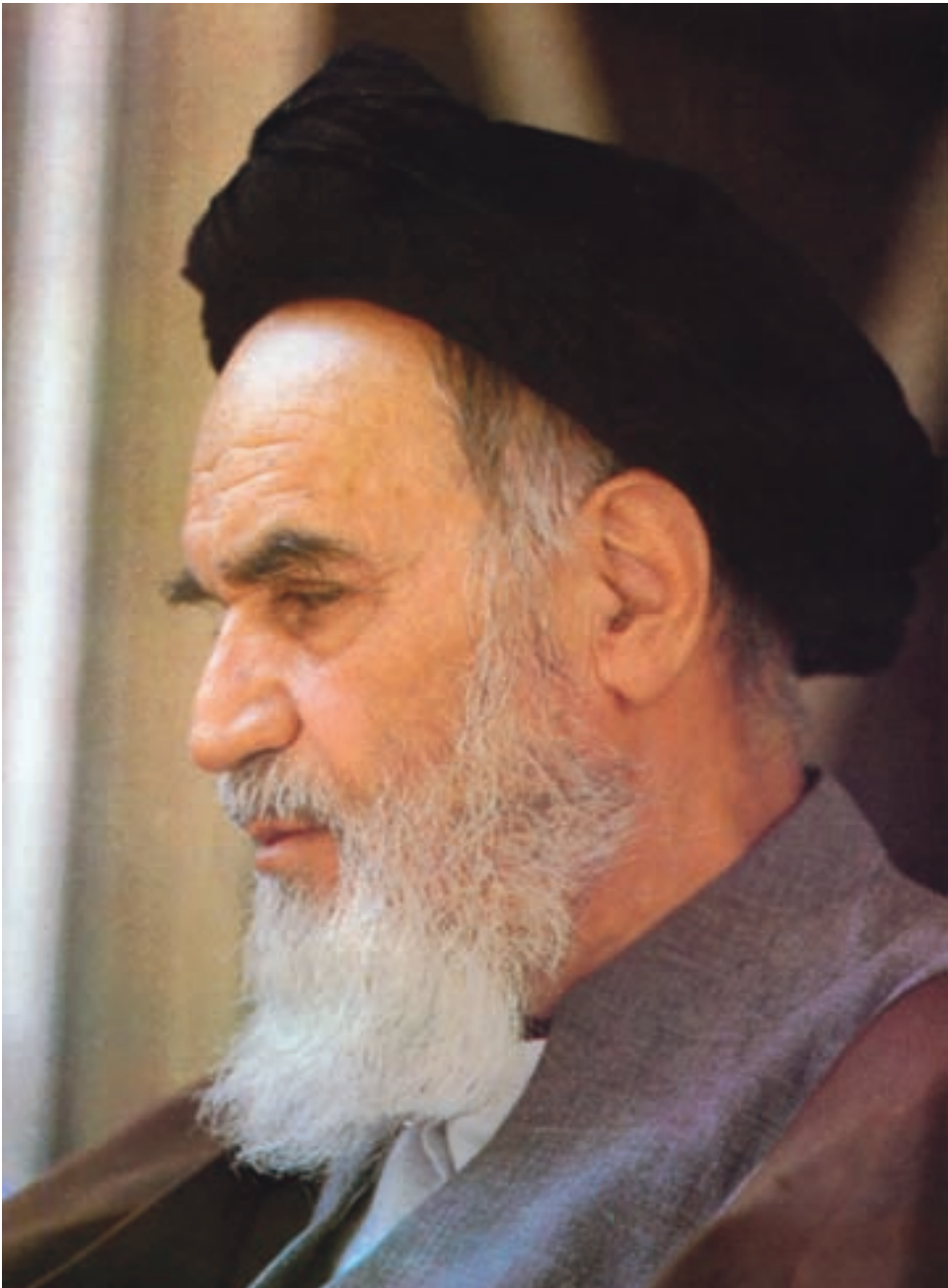
انتشارات فاطمی تهران، میدان دکتر فاطمی، خیابان جویبار، خیابان میرهادی،

شماره‌ی ۱۴، کدپستی ۱۴۱۵۸۸۴۷۴۱. تلفن: ۸۸۹۴۵۵۴۵ (۲۰ خط)

www.fatemi.ir • info@fatemi.ir



این کتاب به سفارش دفتر برنامه‌ریزی و تألیف آموزش‌های فنی و حرفه‌ای و کاردانش سازمان پژوهش و برنامه‌ریزی آموزشی وزارت آموزش و پرورش به‌وسیله‌ی انتشارات فاطمی تألیف و پس از تصویب و تأیید دفتر مذکور به چاپ رسیده است.



بدانید مادام که در احتیاجات صنایع پیشرفته، دست خود را پیش دیگران دراز کنید و به در یوزگی عمر را بگذرانید، قدرت ابتکار و پیشرفت در اختراعات در شما شکوفا نخواهد شد.

امام خمینی «قدس سرّه الشریف»

فهرست

| | |
|--|-------|
| پیشگفتار ناشر | نه |
| پیشگفتار مؤلفان | یازده |
| فصل اول آشنایی با ویژوال بیسیک و حل مسأله به کمک آن | ۱ |
| ۱-۱ انواع زبان‌های برنامه‌نویسی بر اساس نزدیکی به زبان ماشین | ۲ |
| ۱-۲ انواع زبان‌های برنامه‌نویسی بر اساس نوع ترجمه | ۴ |
| ۱-۳ انواع زبان‌های برنامه‌نویسی بر اساس رابط برنامه‌نویسی | ۵ |
| ۱-۴ تاریخچه‌ی زبان بیسیک (Basic) | ۵ |
| ۱-۵ طراحی برنامه‌ها برای حل مسایل | ۶ |
| ۱-۵-۱ طرح برنامه برای حل مسأله | ۷ |
| ۱-۶ کار با ویژوال بیسیک | ۱۹ |
| ۱-۶-۱ آشنایی با محیط ویژوال بیسیک | ۲۰ |
| ۱-۷ مشخصه‌ها، متدها و رویدادها | ۲۸ |
| ۱-۸ برنامه‌نویسی رویدادگرا | ۳۱ |
| ۱-۹ تغییر مشخصه | ۳۲ |
| ۱-۱۰ اجرای برنامه | ۳۴ |
| ۱-۱۱ انواع پرونده‌ها در ویژوال بیسیک | ۳۵ |
| خلاصه‌ی فصل | ۳۶ |
| خودآزمایی | ۳۸ |
| فصل دوم انواع داده‌ها | ۳۹ |
| ۲-۱ داده‌های عددی | ۳۹ |

| | |
|----|--|
| ۴۱ | ۲-۲ سایر انواع داده‌ها |
| ۴۳ | ۲-۳ متغیرها |
| ۴۳ | ۲-۳-۱ اعلان متغیرها |
| ۴۶ | ۲-۳-۲ متغیرهای رشته‌ای |
| ۴۶ | ۲-۳-۳ مقدار دادن به متغیرها |
| ۴۸ | ۲-۴ ثابت‌ها (Constants) |
| ۴۹ | ۲-۵ عملگرهای ویژه‌ال بیسیک |
| ۵۰ | ۲-۵-۱ تقدم عملگرها |
| ۵۱ | ۲-۶ دکمه‌ی فرمان (Command Button) |
| ۵۲ | ۲-۷ نمایش متن روی فرم و کادر تصویر |
| ۵۳ | ۲-۸ متد cls |
| ۵۴ | ۲-۹ توابع داخلی |
| ۵۵ | ۲-۹-۱ تابع SPC() |
| ۵۵ | ۲-۹-۲ تابع MsgBox() |
| ۶۱ | ۲-۹-۳ تابع InputBox() |
| ۶۸ | خلاصه‌ی فصل |
| ۶۹ | خودآزمایی |
| ۷۰ | فصل سوم کنترل برنامه |
| ۷۰ | ۳-۱ عملگرهای رابطه‌ای یا مقایسه‌ای |
| ۷۳ | ۳-۲ عملگرهای منطقی |
| ۷۴ | ۳-۳ ترکیب عملگرهای رابطه‌ای و منطقی |
| ۷۶ | ۳-۴ کنترل Label |
| ۷۶ | ۳-۴-۱ مقداردهی توضیح برجسب |
| ۷۶ | ۳-۴-۲ ترازبندی متن |
| ۷۶ | ۳-۴-۳ مشخصه‌های AutoSize و WordWrap |
| ۷۷ | ۳-۴-۴ کاربرد برجسب‌ها برای ایجاد کلیده‌ای دسترسی |
| ۷۷ | ۳-۵ کنترل Text Box (کادر متن) |
| ۷۷ | ۳-۵-۱ مشخصه‌ی Text |

| | |
|-----|-----------------------------------|
| ۷۸ | ۳-۵-۲ قالب‌بندی متن |
| ۷۸ | ۳-۵-۳ ایجاد کادر متن گذرواژه |
| ۷۸ | ۳-۶ دستور If |
| ۸۰ | ۳-۶-۱ Else با If کامل کردن |
| ۸۷ | ۳-۶-۲ خروج زودرس |
| ۸۸ | ۳-۶-۳ دستورهای If ... Else متداخل |
| ۹۴ | ۳-۷ انتخاب با Select Case |
| ۱۰۳ | ۳-۸ تابع IIf() |
| ۱۰۴ | ۳-۹ تابع Choose() |
| ۱۰۷ | ۳-۱۰ تابع Switch() |
| ۱۰۷ | ۳-۱۱ کنترل Check Box (کادر علامت) |
| ۱۰۸ | ۳-۱۱-۱ Value مشخصه‌ی |
| ۱۰۸ | ۳-۱۱-۲ رویداد Click |
| ۱۱۱ | ۳-۱۲ کنترل Option Button |
| ۱۱۵ | ۳-۱۳ کنترل Frame |
| ۱۲۲ | خلاصه‌ی فصل |
| ۱۲۳ | خودآزمایی |
| ۱۲۴ | فصل چهارم ساختارهای تکرار |
| ۱۲۵ | ۴-۱ حلقه‌ی تکرار For ... Next |
| ۱۳۴ | ۴-۱-۱ نکاتی درباره‌ی حلقه‌ی FOR |
| ۱۳۴ | ۴-۲ حلقه‌ی Do |
| ۱۴۱ | ۴-۳ حلقه‌های متداخل |
| ۱۴۴ | ۴-۴ توابع |
| ۱۴۵ | ۴-۴-۱ توابع ریاضی |
| ۱۵۱ | خلاصه‌ی فصل |
| ۱۵۲ | خودآزمایی |
| ۱۵۴ | فصل پنجم رشته‌ها و توابع رشته‌ای |
| ۱۵۴ | ۵-۱ نویسه‌ها و رشته‌ها |

| | |
|-----|---------------------------------------|
| ۱۵۵ | ۵-۱-۱ الحاق رشته با & و + |
| ۱۵۶ | ۵-۱-۲ مقایسه‌ی رشته‌ها |
| ۱۶۰ | ۵-۱-۳ عملگر Like |
| ۱۶۳ | ۵-۱-۴ تابع Mid() و تابع Len() |
| ۱۶۵ | ۵-۱-۵ توابع Left() و Right() |
| ۱۶۵ | ۵-۱-۶ توابع InStr و InStrRev |
| ۱۶۹ | ۵-۱-۷ توابع Trim()، RTrim() و LTrim() |
| ۱۷۰ | ۵-۱-۸ توابع Space() و String() |
| ۱۷۱ | ۵-۱-۹ جایگزینی زیررشته در یک رشته |
| ۱۷۲ | ۵-۱-۱۰ معکوس کردن رشته |
| ۱۷۳ | ۵-۱-۱۱ سایر توابع رشته‌ای |
| ۱۷۴ | ۵-۲ توابع تبدیلی |
| ۱۷۶ | ۵-۲-۱ تبدیل نوع نویسه‌های رشته |
| ۱۷۶ | ۵-۳ مشخصه‌های تکمیلی کنترل Text Box |
| ۱۷۷ | ۵-۳-۱ ایجاد کادر متن فقط خواندنی |
| ۱۷۷ | ۵-۳-۲ نمایش علامت کوتیشن در یک رشته |
| ۱۸۱ | خلاصه‌ی فصل |
| ۱۸۳ | خودآزمایی |
| ۱۸۴ | فصل ششم پروژه‌های برنامه‌نویسی |
| ۱۸۴ | ۶-۱ ماشین حساب |
| ۱۸۴ | ۶-۱-۱ مرحله‌ی اول طراحی |
| ۱۸۹ | ۶-۱-۲ مرحله‌ی دوم طراحی |
| ۱۹۶ | ۶-۱-۳ مرحله‌ی سوم طراحی |
| ۲۰۰ | ۶-۱-۴ مرحله‌ی چهارم طراحی |
| ۲۲۰ | ۶-۲ نمایش تصویر |
| ۲۲۸ | تمرینات تکمیلی |
| ۲۲۹ | پیوست |
| ۲۴۳ | منابع |

پیشگفتار ناشر

آغاز هزاره‌ی جدید با ظهور فناوری نوین اطلاعات و ارتباطات، عصر اطلاعات نامیده شده و کلیه‌ی شیئون فرهنگی و اقتصادی را تحت تأثیر قرار داده است. این فناوری به‌عنوان ابزاری توانمندکننده و برابرساز فرصت‌های جدیدی را فراهم کرده است تا بتوانیم با شتاب بیشتری در مسیر توسعه گام برداریم.

جوانان ترقی‌خواه با دستیابی به این فناوری نوین، ضمن آموزش و کسب مهارت‌های فنی و حرفه‌ای لازم، می‌توانند نقش ویژه‌ای را در مسیر توسعه ایفا نمایند.

از این رو برنامه‌های آموزشی وزارت آموزش و پرورش در شاخه‌ی فنی و حرفه‌ای توانمندسازی دانش‌آموزان در رشته‌ی رایانه را با توجه به رویکردهای نوین عصر اطلاعات، هدف قرار داده است.

بدیهی است که یکی از رسانه‌های آموزشی مهم، کتاب است. وزارت آموزش و پرورش کشورمان با ابتکاری جدید، تألیف کتاب‌های آموزش رایانه توسط مؤلفان متعدد با مساعی ناشران بخش خصوصی را در دستور کار و در عرصه‌ی رقابت قرار داده است که این امر از جهات گوناگون نویدبخش و ارزشمند است و امید است که افق‌ها و دریچه‌های جدیدی را به‌روى دانش‌آموزان کشور بگشاید تا نسل جوان پرشور و پرتوان به‌سوی آینده‌ای نویدبخش گام بردارند.

مؤسسه انتشارات فاطمی، در راستای رسیدن به این هدف، طی موافقت‌نامه‌ی همکاری با سازمان پژوهش و برنامه‌ریزی آموزشی تألیف تعدادی از این کتاب‌ها را برعهده گرفت و کار تألیف و تولید مطابق با راهنمای برنامه‌ی درسی و اصول تدوین محتوای کتاب‌های درسی که از طرف دفتر برنامه‌ریزی و تألیف آموزش‌های فنی و حرفه‌ای و کاردانش در اختیار

این مؤسسه قرار گرفت، به انجام رسید. این کتاب‌ها در مرحله‌ی تألیف به وسیله‌ی کارشناسان محترم دفتر برنامه‌ریزی و تألیف آموزش‌های فنی و حرفه‌ای و کاردانش از نظر محتوا و ساختار مورد بررسی و تجدیدنظر قرار گرفته است. امید است این کتاب‌ها که برای اولین بار مورد استفاده‌ی هنرجویان عزیز قرار می‌گیرد مورد توجه آنان واقع شود. از دریافت اظهار نظرهای سازنده‌ی هنرآموزان و کارشناسان محترم و هنرجویان عزیز پیشاپیش سپاسگزاری می‌شود و در چاپ‌های بعدی نسبت به رفع کمبودها و نقایص احتمالی اقدام خواهد شد.

مؤسسه انتشارات فاطمی

پیشگفتار مؤلفان

برکسی پوشیده نیست که عصر حاضر را عصر اطلاعات نامیده‌اند. از همین رو اهمیت دسترسی به اطلاعات، پردازش و تبادل آنها از اهمیت ویژه‌ای برخوردار است. پیدایش انفورماتیک قدمتی حداکثر ۶۵ ساله دارد. با این حال بخش قابل توجهی از نیروی انسانی جوامع صنعتی و در حال رشد، در این زمینه مشغول فعالیت هستند و سرمایه‌گذاری خوبی برای پرورش نیروی انسانی متخصص و صدور فناوری آن به سایر نقاط انجام داده‌اند.

در کشور ما نیز کم و بیش فعالیت‌هایی برای ارتقای دانش عمومی انفورماتیک صورت گرفته و رشته‌ی تقریباً نوپایی در مقطع متوسطه و عالی به‌وجود آمده است. مجموعه‌ی درس‌های این دوره‌ها، تلاشی برای حرکت نسل جوان کشور به‌منظور بهره‌وری بالا در بازار کار و برداشتن باری از دوش جامعه است.

کتابی که هم‌اکنون پیش روی شماست، قصد دارد شما را با گذراندن یک دوره‌ی محدود، با مفاهیم اولیه و اساسی برنامه‌نویسی آشنا کند. در این کتاب، اصول اولیه‌ی یکی از زبان‌های برنامه‌نویسی متداول امروزی را آموزش خواهیم داد. بدیهی است مانند یادگیری سایر مهارت‌ها، تسلط عمومی بر این زبان برنامه‌نویسی، مستلزم انجام کار عملی و صرف وقت است، به‌همین دلیل به هنرجویان عزیز توصیه می‌کنیم که مطالب این کتاب و مثال‌ها را به‌صورت دقیق مطالعه و اجرا نمایند.

توجه هنرآموزان محترم را به این نکته جلب می‌کنیم که رویکرد ما در این کتاب و درس‌های برنامه‌سازی ۲ و ۳ به‌سمت برنامه‌نویسی شی‌اگرا خواهد بود. هر چند که معتقدیم برنامه‌نویسی ساخت‌یافته مقدمه‌ای برای آموزش مفاهیم شی‌اگراست. شروع درس برنامه‌سازی با حل مسأله است که در درس مبانی کامپیوتر مطرح شده است. مناسب است هنرآموزان محترم پیش از پیاده‌سازی یک مسأله در مورد چگونگی حل آن، توضیحات لازم را ارائه نمایند. همچنین توصیه می‌شود هنرجویان کد برنامه‌ها را از روی پرونده‌های دیگر کپی نکنند تا در نوشتن صحیح و سریع کدها مهارت لازم را کسب کنند. در این کتاب سعی شده است

اجزای کدها مانند عملگرها و عملوندها، کنترل و متدها، متغیرها و شناسه‌ها، کلمات کلیدی و موارد مشابه مشخص شوند تا تشخیص آنها ساده‌تر باشد.

این کتاب دارای ۶ فصل است که در پایان هر فصل، تمرین و خودآزمایی‌هایی مطرح شده است که هنرجو می‌تواند بر اساس مطالب فصل و مثال‌های مطرح‌شده، به این تمرین‌ها پاسخ دهد.

با آرزوی توفیق روزافزون برای
پژوهندگان علم در کشورمان

فصل اول

آشنایی با ویژوال بیسیک و حل مسأله به کمک آن

پس از پایان این فصل، انتظار می رود که فراگیر بتواند:

- زبان برنامه نویسی را تعریف کند و انواع آن را نام ببرد.
- روش طراحی برنامه را توضیح دهد.
- برنامه ها را تحلیل کند و شبه کد مربوطه را بنویسد.
- مراحل حل مسأله را با ذکر مثال شرح دهد.
- شیءها را روی فرم قرار دهد.
- مشخصه ها، متدها و رویدادهای متداول در فرم و شیءها را شرح دهد.

در درس مبانی کامپیوتر با روش حل مسأله و انواع زبان های برنامه نویسی آشنا شدید. زبان برنامه نویسی، به مجموعه ای از علائم، قواعد و دستورالعمل ها گفته می شود که امکان ارتباط با کامپیوتر را برای بیان کاری یا حل مسأله ای فراهم می کند. به تعبیر ساده تر، می توان گفت که برنامه ی کامپیوتری، مجموعه ای از دستورالعمل هاست که برای حل مسأله ای نوشته می شود. برنامه ها باید به زبان های خاصی نوشته شوند تا برای کامپیوتر قابل فهم باشند.

زبان های برنامه نویسی را می توان از چهار دیدگاه متفاوت مورد بررسی قرار داده و تقسیم بندی کرد:

الف) نزدیکی به زبان ماشین

۱. سطح پایین

۲. سطح میانی
۳. سطح بالا
- ب) نوع ترجمه
 ۱. مفسری
 ۲. کامپایلری
- ج) رابط برنامه‌نویسی
 ۱. مبتنی بر متن
 ۲. مبتنی بر گرافیک (ویژوال)
- د) روش‌های برنامه‌نویسی
 ۱. زیرروالی
 ۲. ساخت یافته
 ۳. مدولار
 ۴. شیء‌گرا

سه دیدگاه اول در این کتاب و دیدگاه چهارم در کتاب برنامه‌سازی ۳ مورد بررسی قرار می‌گیرد.

۱-۱ انواع زبان‌های برنامه‌نویسی بر اساس نزدیکی به زبان ماشین

برنامه‌نویس می‌تواند دستورهای خود را در انواع متفاوتی از زبان‌های برنامه‌نویسی بنویسد. تعدادی از این زبان‌ها به صورت مستقیم به وسیله‌ی رایانه درک می‌شوند و تعداد دیگری نیاز به ترجمه دارند تا قابل فهم برای رایانه شوند. امروزه صدها زبان رایانه‌ی استفاده می‌شوند که می‌توان آنها را به سه دسته تقسیم کرد:

۱. **زبان سطح پایین (low-level languages):** زبان‌هایی که به زبان ماشین نزدیک هستند.
۲. **زبان سطح میانی (medium-level languages):** زبان‌هایی که هم به زبان ماشین و هم به زبان محاوره‌ی انگلیسی نزدیک هستند.
۳. **زبان‌های سطح بالا (high-level languages):** زبان‌هایی هستند که به زبان محاوره‌ی انگلیسی نزدیک هستند.

هر رایانه به‌طور مستقیم فقط زبان ماشین خود را درک می‌کند. زبان ماشین، زبان ذاتی و انحصاری رایانه است که هنگام طراحی سخت‌افزار رایانه تعریف می‌شود. **زبان ماشین**، شامل رشته‌ای از اعداد است و سبب می‌شود که رایانه عملیات اصلی مربوط به خود را در هر بار راه‌اندازی، اجرا کند. زبان ماشین، وابسته به ماشین است (به‌عنوان مثال، زبان ماشین یک دستگاه، فقط روی همان نوع ماشین اجرا می‌شود). درک زبان ماشین برای انسان بسیار مشکل است. به‌عنوان مثال، به دستوره‌های زبان ماشین که در قسمت پایین آورده شده است، توجه کنید. این برنامه، اضافه‌کار را بر مبنای حقوق محاسبه و نتیجه را در متغیر grosspay ذخیره می‌کند.

+ 1300042774

+ 1400593419

+ 1200274027

نکته

متغیر (variable) مکانی در حافظه است که برای نگهداری یک مقدار مورد استفاده قرار می‌گیرد. مقداری که در متغیر قرار داده می‌شود، قابل تغییر است (نام آن هم بر همین ویژگی دلالت دارد). وقتی مقداری را در یک متغیر قرار می‌دهید، مقدار قبلی آن از بین خواهد رفت.

هنگامی که رایانه‌ها مورد استفاده‌ی عموم قرار گرفتند، مشخص شد که برنامه‌نویسی به زبان ماشین برای بسیاری از برنامه‌نویسان خسته‌کننده و ملال‌آور است. برنامه‌نویسان به‌جای به‌کار بردن رشته‌ای از اعداد که رایانه بتواند به‌طور مستقیم آنها را درک کند، از عبارتهای مخفف‌شده‌ای شبیه زبان انگلیسی برای فهماندن عملیات ابتدایی به رایانه استفاده کردند. این عبارتهای مخفف‌شده و شبه‌انگلیسی، مبنای **زبان اسمبلی** هستند. برنامه‌های مترجم به‌نام اسمبلر مشهور هستند که زبان **اسمبلی** را به زبان ماشین ترجمه می‌کنند. قطعه برنامه‌ای که در قسمت پایین آورده شده است، همان عملیات بالا را انجام می‌دهد، با این تفاوت که با استفاده از زبان اسمبلی نوشته شده‌اند که نسبت به زبان ماشین از وضوح (قابلیت فهم) بیشتری برخوردار است.

LOAD BASEPAY

ADD OVERPAY

STORE GROSSPAY

این کد برای انسان وضوح بیشتری دارد، اما برای رایانه تا زمانی که به زبان ماشین ترجمه نشود، معنی ندارد. زبان اسمبلی سبب افزایش سرعت برنامه نویسی شد اما هنوز هم برای انجام یک عمل ساده مستلزم دستورهای فراوانی بود. برای افزایش سرعت برنامه نویسی، **زبان‌های سطح بالا** توسعه پیدا کردند که با استفاده از یک عبارت می‌توانند وظایف و اعمال وسیعتری را انجام دهند. برنامه‌های مترجم که وظیفه‌ی تبدیل زبان‌های سطح بالا به زبان ماشین را برعهده دارند، کامپایلر نامیده می‌شوند. زبان‌های سطح بالا به برنامه نویس امکان می‌دهند که دستورهای مورد نیاز خود را تقریباً مانند زبان انگلیسی و عملیات ریاضی را به صورت عادی بنویسد.

$$\text{Grosspay} = \text{basepay} + \text{Overpay}$$

واضح است که زبان‌های سطح بالا نسبت به زبان‌های ماشین یا اسمبلی از محبوبیت بیشتری نزد برنامه‌نویسان برخوردارند.

۲-۱ انواع زبان‌های برنامه‌نویسی بر اساس نوع ترجمه

عمل **کامپایل کردن** برنامه‌ی زبان سطح بالا به زبان ماشین، می‌تواند وقت زیادی از رایانه بگیرد. برنامه‌های **مفسر (interpreter)** توسعه‌یافته می‌توانند به صورت مستقیم برنامه‌های زبان‌های سطح بالا را بدون نیاز به کامپایل، به زبان ماشین تبدیل کنند. اگر چه برنامه‌های مفسر نسبت به برنامه‌های کامپایلر آهسته‌تر عمل می‌کنند، ولی برنامه‌های مفسر فوراً شروع به فعالیت می‌کنند، بدون اینکه تأخیرهای ذاتی از عمل کامپایل را داشته باشند. ویژوال بیسیک نمونه‌ای از زبان مفسر و کامپایلری است.

تاکنون صدها زبان سطح بالا ایجاد شده است، اما فقط تعدادی از آنها موفقیت قابل قبولی به دست آورده‌اند. فرتن را شرکت IBM بین سال‌های ۱۹۵۴ و ۱۹۵۷ ایجاد کرد و در کاربردهای علمی و مهندسی که نیاز به محاسبات پیچیده‌ی ریاضی دارند به‌کارگرفت. فرتن هنوز هم به صورت گسترده‌ای مخصوصاً در کاربردهای مهندسی استفاده می‌شود. کوبول را گروهی از سازنده‌های رایانه و کارخانه‌هایی که از رایانه استفاده می‌کردند، در سال ۱۹۵۹ طراحی و ایجاد کردند. کوبول به صورت زبان تجاری مورد استفاده قرار گرفت که به انجام عملیات دقیق بر روی مقادیر زیادی از داده‌ها نیاز داشت.

زبان C را «دنيس ریچی»^۱ در آزمایشگاه‌های بل در سال ۱۹۷۲ نوشت. C یکی از محبوب‌ترین زبان‌های مورد استفاده در صنایع است و از آن در تولید سیستم عامل یونیکس استفاده شده است.

1. Dennis Ritchie

پاسکال^۱ همزمان با C طراحی شد. این زبان را پرفسور نیکلاس ورت^۲ برای استفاده‌های علمی نوشت.

C++ توسعه‌یافته‌ی C است که در اوایل سال ۱۹۸۰ در آزمایشگاه‌های بل نوشته شد. C++ نسبت به C توانایی‌های بیشتری دارد. یکی از توانایی‌های آن، برنامه‌نویسی شی‌اگرا است.

۳-۱ انواع زبان‌های برنامه‌نویسی بر اساس رابط برنامه‌نویسی

زبان‌هایی که در بالا توضیح دادیم، زبان‌هایی **بر مبنای متن (text-based)** هستند. زبان‌های مبتنی بر متن، به‌کاربر امکان دسترسی مستقیم به گرافیک را نمی‌دهند. یک بسته‌ی نرم‌افزاری ویژه اغلب به عناصر گرافیکی نیاز دارد که به آن افزوده شود. این بسته‌های نرم‌افزاری اغلب به نوشتن خطوط زیادی از کدها نیاز دارند که به کد برنامه‌ی اصلی اضافه می‌شوند.

زبان‌های مبتنی بر گرافیک (ویژوال)، امکان دسترسی مستقیم کاربر به گرافیک را فراهم می‌آورند. از زبان‌های مبتنی بر گرافیک، می‌توان برای ایجاد سریع برنامه‌های تحت ویندوز استفاده کرد، بدون اینکه برنامه‌نویس نیازی به یادگیری و استفاده از بسته‌های نرم‌افزاری اضافی داشته باشد. جاوا (Java) را سان‌میکروسیستم ایجاد و در سال ۱۹۹۵ به بازار عرضه کرد. جاوا از ترکیب C و C++ و ادغام نکته‌های برجسته‌ی این دو زبان ایجاد شده است. جاوا شامل توابع وسیع کتابخانه‌ای برای استفاده در چند رسانه‌ای، شبکه، گرافیک، واسط گرافیکی توسعه‌یافته‌ی کاربر، دستیابی به پایگاه داده‌ها، محاسبات توزیع شده و... است. نوع مایکروسافت جاوا به نام Visual J++ معروف است.

پاور بیلدر (Power Builder) ساخت شرکت Powersoft Corporation و دلفی (Delphi) ساخت شرکت Borland International است و در مقایسه با دلفی، کاربا ویژوال بیسیک ساده‌تر است.

۴-۱ تاریخچه‌ی زبان بیسیک (Basic)

BASIC سرنام کلمات "Beginner's All-purpose Symbolic Instruction Code" به معنای زبان همه‌منظوره برای افراد مبتدی است. این زبان برنامه‌سازی، به دلیل

1. Pascal 2. Nicklaus Wirth

سادگی ساختاری، از محبوبیت زیادی برخوردار است. یک هنرجوی مبتدی که آشنایی زیادی با رایانه و برنامه‌نویسی ندارد، پس از آموزشی کوتاه خواهد بود که این زبان را یاد بگیرد و امکان نوشتن برنامه در محیط این زبان را به دست آورد.

زبان برنامه‌سازی بیسیک در سال ۱۹۶۴ میلادی، به وسیله‌ی جان کمپی^۱ و توماس کورتس^۲ در کالج دارتموث^۳ پدید آمد. این زبان، نخستین زبان برنامه‌سازی نبود ولی هدف آن، فراهم کردن یک زبان ساده برای دانشجویان رشته‌های مختلف بود. تا به امروز نسخه‌های متعددی از زبان بیسیک ارایه شده است که می‌توان به **ANSI BASIC**، **BASICA**، **GW BASIC**، **QBASIC** و **QUICK BASIC** اشاره کرد.

زبان برنامه‌سازی بیسیک با ارایه‌ی ویژوال بیسیک جان تازه‌ای گرفت و دوباره رونق یافت. به همین دلیل، به‌عنوان یک زبان برنامه‌سازی پایه در دوره‌ی آموزش رایانه شناخته شده است. هنرجو، با آموختن این زبان، با اصول برنامه‌سازی و همچنین برنامه‌نویسی در محیط ویژوال بیسیک آشنا خواهد شد.

در هر زبان برنامه‌سازی اگر مقدمات آن زبان را بیاموزید، نوشتن برنامه‌ها با آن زبان ساده خواهد بود.

مقدمات یک زبان عبارت‌اند از: انواع داده‌های موجود، چگونگی اعلان متغیرها، انواع عملگرها، دستورهای شرطی، انواع حلقه‌های تکرار و دستورهای ورودی/خروجی.

۵-۱ طراحی برنامه‌ها برای حل مسایل

همواره در طول تحصیل، از شما خواسته شده است که مسایلی را حل کنید. برخی از این مسایل شامل محاسباتی نیز هستند، نظیر آنهایی که در درس ریاضیات یا فیزیک مطرح می‌شوند (میانگین پنج مقدار زیر چقدر است؟ ریشه‌ی دوم عدد ۵۹۳ را محاسبه کنید. حداقل سرعتی که یک سفینه‌ی فضایی باید با آن پرواز کند تا بتواند از کشش میدان جاذبه فرار کند چقدر است؟ و...). بقیه‌ی مسایل شامل «دستکاری» متن هستند (حرف بزرگ کلمه‌ی "Wyoming" کدام است؟ لیست کلمات زیر را از نظر الفبایی مرتب کنید. هم معنی کلمه‌ی "great" کدام کلمه است؟ و...).

در طول تحصیل در هنرستان، شما روش‌هایی را برای حل دامنه‌ی گسترده‌ای از مسایل

آموخته‌اید. در این کتاب درسی، خواهید آموخت که چگونه این گونه مهارت‌های حل مسایل را به نحوی تبدیل کنید که بتوانید از یک رایانه برای حل مسأله، بهره بگیرید. این موضوع، اساس برنامه‌نویسی رایانه‌ای است؛ یعنی باید بتوان یک تکنیک حل مسأله را به صورت مجموعه‌ای از گام‌هایی که یک رایانه می‌تواند آنها را به‌کار گیرد، درآورد.

در تشریح تکنیک حل مسأله برای یک رایانه، باید آن را به صورتی بسیار ساخت یافته بیان کرد. توصیف شما باید شامل تمامی گام‌های لازم برای حل مسأله باشد و نیز این گام‌ها باید دارای ترتیب و تقدم مناسب باشند. هر گونه اطلاعات «میانی» (intermediate) که وجود آن در یک گام ضروری است باید در یکی از گام‌های قبلی ایجاد شده باشد. پروسه‌ی برنامه‌نویسی بسیار شبیه به نوشتن یک دستورالعمل برای یک کتاب آشپزی است که در آن اگر افراد به‌کارگیرنده‌ی این کتاب بی‌دقت باشند و یا گام‌های لیست شده را با اولویت اشتباهی اجرا کنند، آنگاه غذایی که پخته شده است با آنچه که انتظار داشتید، بسیار متفاوت خواهد بود!

۱-۵-۱ طرح برنامه برای حل مسأله

روش ما در طراحی یک برنامه شامل سه گام اساسی است:

گام ۱: تحلیل مسأله

گام ۲: طراحی برنامه

گام ۳: پیاده‌سازی برنامه

- کدنویسی برنامه و ترجمه

- بررسی و ردگیری اشتباهات (debugging) برنامه

- نوشتن مستندات (documenting) برنامه

گام ۱: تحلیل مسأله

در این گام باید:

۱. اطلاعاتی را که باید تولید شود (یعنی خروجی‌ها) به‌عنوان بخشی از راه‌حل مسأله تعیین کنید. به‌عنوان بخشی از این گام فرعی (sub step)، باید متغیرهایی را که برای نمایش خروجی مورد نیاز هستند، اعلان کنید.

۲. داده‌های (یعنی ورودی) مورد نیاز را برای تولید خروجی‌ها تعیین کنید. به‌عنوان بخشی

از این گام فرعی، باید متغیرهای مورد نیاز را برای دریافت ورودی اعلان کنید.

۳. الگوریتمی را برای به دست آوردن خروجی‌ها از ورودی‌ها تعیین کنید که دارای گام‌های محدودی باشد. الگوریتم، به عنوان مجموعه‌ای ترتیبی و سازماندهی شده از عملیات لازم برای حل یک مسأله، طی مراحل محدود، محسوب می‌شود. در صورت وجود مسایل عددی، این الگوریتم از مجموعه‌ای ترتیبی از محاسبات مورد نیاز برای به دست آوردن خروجی‌های مطلوب تشکیل می‌شود. ولی در مورد مسایل غیر عددی، این الگوریتم‌ها می‌توانند شامل متن‌ها و عملیات ترسیمی متعددی، علاوه بر عملیات عددی باشند.

هدف اصلی از اجرای گام ۱، اطمینان از این است که شما به درستی مسأله را درک کرده‌اید. به خاطر داشته باشید که هرگونه عدم درک صحیح در تحلیل یک مسأله به برنامه‌ی شما نیز منتقل خواهد شد و نمی‌توانید متوقع باشید که رایانه‌ای برای اجرای تحلیل‌های نامناسب، ایجاد شود. یک ضرب‌المثل فرنگی قدیمی می‌گوید:

آشغال ورودی = آشغال خروجی (GARBAGE IN = GARBAGE OUT)

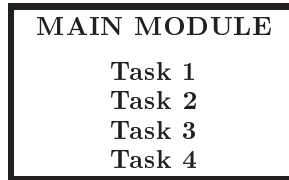
این ضرب‌المثل معمولاً به صورت مخفف GIGO به کار می‌رود. در زبان شیرین فارسی نیز معادل این ضرب‌المثل را داریم: آنچه کشته‌ای، درو خواهی کرد.

گام ۲: طراحی برنامه

در این گام، با استفاده از جملات شبه‌انگلیسی که به آنها شبه‌کد (pseudo-code) گفته می‌شود، برنامه را طرح‌ریزی (outline) کنید. هر جمله به یک عملیات برنامه‌ای ساده مربوط است. در یک برنامه‌ی ساده می‌توان به طور مستقیم «شبه‌کد» را با لیست کردن وظایف مختلفی که یک برنامه باید انجام دهد (به ترتیب تقدم اجرای آنها) ایجاد کرد. اما، در برنامه‌های پیچیده‌تر باید پروسه‌ی طرح‌ریزی را سازماندهی کرد. به این منظور ما از یک روش سازماندهی، موسوم به «طرح از بالا به پایین» (top-down design) استفاده خواهیم کرد.

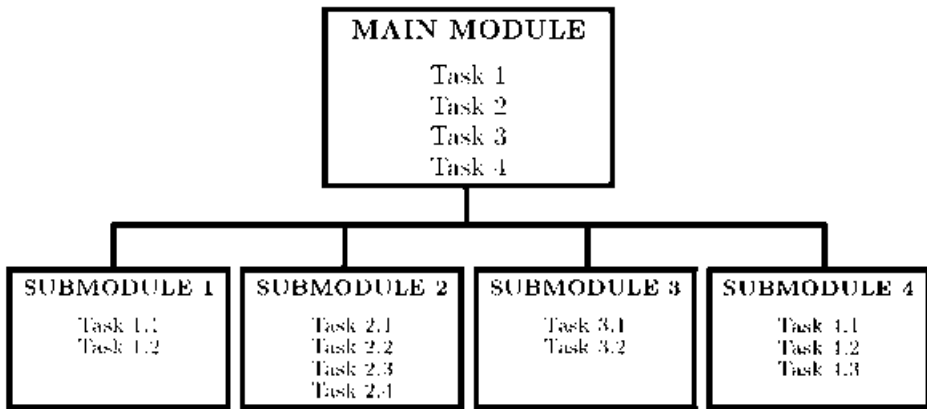
هنگام استفاده از روش «از بالا به پایین» برای طراحی، برنامه را به تعدادی از وظایف (tasks) تقسیم‌بندی می‌کنیم. لیست این وظایف، یک طرح (outline) یا دورنما از برنامه را ارائه می‌دهد که به آن **مدول اصلی (main module)** گفته می‌شود. هنگامی که یک وظیفه در مدول اصلی لیست می‌شود، تنها به وسیله‌ی نام خود معرفی خواهد شد و هیچ‌گونه نشانه‌ای از نحوه‌ی اجرای این وظیفه، در دسترس نخواهد بود. در طراحی برنامه، از این گام برای عبور

به مرحله‌ی بعد استفاده می‌شود. تقسیم‌بندی برنامه به چند وظیفه‌ی خاص، سبب می‌شود که یک طرح اولیه از برنامه به دست آید. این طرح برنامه‌ای، در جدولی مشابه آنچه که در شکل ۱-۱ نشان داده شده است، جمع‌بندی می‌شود.



شکل ۱-۱ طرح اولیه‌ی برنامه

طرح برنامه با تشریح هر وظیفه‌ی مدول اصلی در یک مدول جداگانه (وظیفه‌ی فرعی "subtask" گفته می‌شود)، جزئیات بیشتری را ارایه خواهد کرد (شکل ۱-۲). این مدول‌ها (modules) به «مدول‌های فرعی» (submodules) موسوم‌اند. رابطه‌ی بین مدول‌های فرعی مختلف یک برنامه را می‌توان به صورت نشان داده شده در شکل ۱-۲ جمع‌بندی کرد. به چنین جدولی «جدول ساختاری» (structure chart) گفته می‌شود.



شکل ۱-۲ طرح برنامه (مرحله‌ی دوم)

در طراحی مدول‌ها هنوز هم می‌توانید جزئیاتی را «معوق» بگذارید. در این حالت، باید مدول‌ها را با استفاده از مدول‌های فرعی تجزیه کرده و به یک طرح «مرحله‌ی سوم» برسید. این پروسه باید تا هنگام ورود کلیدی جزئیات برنامه، ادامه یابد. به پروسه‌ی مرحله به مرحله‌ای که هم اکنون توضیح داده شد، تجزیه‌ی گام به گام (stepwise refinement) گفته می‌شود. این یک تجربه‌ی برنامه‌نویسی مناسب برای شما خواهد بود که پیش از برنامه‌نویسی، مسأله را

به صورت گام به گام تجزیه و تحلیل کنید.

تا اینجا روش «از بالا به پایین» را در طرح برنامه‌ها، تشریح کرده‌ایم. در واقع ما از «بالای» (top) برنامه (که دارای بیشترین عمومیت است) شروع به کار کرده و به «پایین» (down) آن (که دارای کمترین عمومیت است) رسیده‌ایم. البته روش‌های دیگری نیز برای طراحی برنامه وجود دارد که در درس برنامه‌سازی ۳ بیان خواهد شد.

گام ۳: پیاده‌سازی برنامه

گام نهایی در طراحی یک برنامه، نوشتن «کد اصلی» (source code) برنامه است. در این گام، شبه‌کد (pseudo-code) طراحی شده در مورد مدول‌ها، به عبارتهای مورد قبول در یک زبان برنامه‌نویسی (مثل ویژوال بیسیک) تبدیل می‌شوند.

به‌عنوان بخشی از کد اصلی، باید تهیه‌ی مستندات (documentation) را نیز انجام دهید. منظور از نوشتن مستندات، وارد کردن توضیحاتی است که عملکرد قسمت‌های مختلف یک برنامه را تشریح می‌کند. علاوه بر این، کد اصلی می‌تواند شامل کد ردگیری اشتباهات (debugging code) نیز باشد. شما به کمک این کد می‌توانید عملیات برنامه را بررسی کرده و اشتباهات (bugs) برنامه‌نویسی را ردیابی کنید. باید پس از آنکه برنامه به‌طور صحیحی شروع به کار کرد، کد ردگیری اشتباهات را از آن خارج کنید. با این وجود، عمل تهیه‌ی مستندات باید به‌عنوان بخش دائمی از کد اصلی در برنامه باقی بماند تا شما (یا هر فرد دیگری) بتوانید از برنامه نگهداری کرده یا در صورت لزوم آن را اصلاح کنید.

مثال‌های زیر، پروسه‌ی حل یک مسأله را که از طراحی ساده آغاز می‌شود، نمایش می‌دهند:

مثال ۱-۱

برنامه‌ای طراحی کنید که مساحت یک مستطیل را که ابعاد آن به وسیله‌ی کاربر تعیین می‌شود، محاسبه کند.

تحلیل مسأله: یک مستطیل دارای دو بعد است: طول و عرض. اینها مقادیری هستند که باید به وسیله‌ی کاربر تعیین شوند. حال متغیرهای مربوط به این دو بعد را اعلان می‌کنیم:

Length = طول مستطیل

Width = عرض مستطیل

این مسأله به برنامه‌ای نیاز دارد که مساحت مستطیل را محاسبه کند. از این رو متغیری را اعلان می‌کنیم که این خروجی را تعریف کند:

$$\text{Area} = \text{مساحت مستطیل}$$

باید توجه داشته باشید که ما از نام‌های کامل در مورد متغیرهای خود استفاده کرده‌ایم (Area به جای A، Length به جای L، و ...). ولی در ریاضیات، تمایل به مخفف‌سازی وجود دارد. با این وجود، بهترین حالت برنامه‌نویسی، استفاده از نام‌های متغیری توصیفی خوانا است که در مواقعی سبب طولانی‌تر به نظر رسیدن فرمول‌ها می‌شوند. با استفاده از نام‌های متغیری توصیفی، کد شما «خود مستند» (self-documenting) خواهد شد.

طراحی برنامه:

همانطور که می‌دانید، برای محاسبه‌ی مساحت یک مستطیل، فرمول ساده‌ای به صورت زیر وجود دارد:

$$\text{عرض} \times \text{طول} = \text{مساحت مستطیل}$$

با این فرمول می‌توانیم مقدار Area (مساحت) را از ورودی‌های برنامه محاسبه کنیم. در زیر، خلاصه‌ای از تحلیل ما ارائه شده است:

Program Inputs: Length, Width (ورودی‌های برنامه: طول و عرض)

Program Output: Area (خروجی برنامه: مساحت)

Algorithm: الگوریتم:

Obtain Length and Width from the user.

(طول و عرض را از کاربر دریافت کن.)

Compute Area using the formula:

(مساحت را با استفاده از فرمول زیر محاسبه کن):

$$\text{Area} = \text{Length times Width}$$

(عرض \times طول = مساحت)

Display the value of Area.

(مساحت را نمایش بده.)

این برنامه باید سه مطلب را دربرگیرد:

Obtain the dimensions of rectangle from the user.

(دریافت ابعاد مستطیل از کاربر)

Calculate the area. (محاسبه‌ی مساحت)

Display the result. (نمایش نتیجه)

با توجه به سه مطلب فوق، طرح برنامه نیز باید شامل سه وظیفه (task) باشد:

Program Rectangle. (برنامه‌ی مستطیل)

Main Module: Rectangle. (مدول اصلی: مستطیل)

Module 1: obtain the dimensions. (مدول ۱: دریافت ابعاد)

Module 2: calculate the area. (مدول ۲: محاسبه‌ی مساحت)

Module 3: display the result. (مدول ۳: نمایش نتیجه)

طرح فوق، «شبه‌کد» مدول اصلی برنامه را ارائه می‌دهد ولی شامل هیچ گونه جزئیاتی در مورد وظایف گوناگون نیست. این جزئیات در شبه‌کد زیر به صورت سه مدول فرعی ارائه شده‌اند:

Module 1: obtain the dimensions. (مدول ۱: دریافت ابعاد)

Display prompt 'Type the dimensions:'

(نمایش پیغام «ابعاد را وارد کن:»)

Display prompt 'Length = ' . (نمایش پیغام «طول = »)

Input the value for Length. (دریافت مقدار طول)

Display prompt 'Width = ' . (نمایش پیغام «عرض = »)

Input the value for Width. (دریافت مقدار عرض)

Module 2: calculate the area. (مدول ۲: محاسبه‌ی مساحت)

Area = Length times Width. (عرض \times طول = مساحت)

Module 3: display the result. (مدول ۳: نمایش نتیجه)

Display the message 'The area equals'.

(نمایش پیغام «مساحت برابر است با»)

Display the value of Area. (نمایش مقدار مساحت)

اکنون جزئیات طراحی برنامه کامل شده است. توجه داشته باشید که این طرح نیاز به دو «مرحله» یا «سطح» از جزئیات دارد. این بدان معنی است که برنامه به مرحله‌ی مدول

اصلی و مرحله‌ی مدول فرعی نیاز دارد. رابطه‌ی بین این مدول‌ها در جدول ساختاری موجود در شکل ۱-۳ نشان داده شده است.



شکل ۱-۳ جدول ساختاری مدول‌های برنامه‌ی مربوط به مستطیل.

طرح برنامه را می‌توان در «شبه‌کد» (pseudo-code) زیر جمع‌بندی کرد:

/* Calculate the Area of a Rectangle */

(/* محاسبه‌ی مساحت مستطیل */)

Main Module: Rectangle. (مدول اصلی: مستطیل)

Obtain the dimensions. (دریافت ابعاد)

Calculate the area. (محاسبه‌ی مساحت)

Display the result. (نمایش نتیجه)

Module 1: Obtain the dimensions. (مدول ۱: دریافت ابعاد)

Display prompt 'Type the dimensions'
(نمایش پیغام «ابعاد را وارد کن»)

Display prompt 'Length = '
(نمایش پیغام «طول =»)

Input the value for Length.
(دریافت مقدار طول)

Display prompt 'Width = '
(نمایش پیغام «عرض =»)

Input the value for Width.
(دریافت مقدار عرض)

Module 2: Calculate the area. (مدول ۲: محاسبه‌ی مساحت)

Area = Length times Width. (عرض \times طول = مساحت)

Module 3: Display the result. (مدول ۳: نمایش نتیجه)

Display the message 'The area equals'.
(نمایش پیغام «مساحت برابر است با»)

Display the value of Area. (نمایش مقدار مساحت)

پیاده‌سازی برنامه:

لیستی از متغیرهایی را که باید در کد اصلی مورد استفاده قرار گیرند، آماده می‌کنیم. سپس بر اساس ساختارهای برنامه‌نویسی در یک زبان خاص، کد برنامه را پیاده‌سازی می‌کنیم.

مثال ۲-۱

امیررضا اتاق‌ها را رنگ‌آمیزی کرده و پوشش‌های سقف را نصب کرده است. وی امیدوار است که بتواند از رایانه شخصی خود برای فراهم آوردن ایده‌ای برای مناقصه‌ی کارهای بعدی خود استفاده کند. وی بابت رنگ‌آمیزی هر متر مربع ۱۰۰۰۰ ریال، بابت رنگ‌آمیزی کارهای چوبی به‌ازای هر متر ۱۰۰۰۰ ریال، برای رنگ‌آمیزی سقف به‌ازای هر متر مربع ۲۰۰۰۰ ریال و برای نصب پوشش سقف به‌ازای هر متر مربع ۳۰۰۰۰ ریال هزینه متقبل شده است. برنامه‌ای طراحی کنید که هزینه‌ی یک کار بخصوص از این نوع را برآورد کند. همانطور که مشاهده خواهیم کرد، این مسأله پیچیده‌تر از مثال قبلی است. با این وجود می‌توان آن را با استفاده از همان روش سه مرحله‌ای حل کرد.

تحلیل مسأله:

در این مسأله‌ی خاص، باید هزینه‌ی کلی یک کار برآورد شود. این هزینه از دو مؤلفه تشکیل شده است: هزینه‌ی رنگ‌آمیزی و هزینه‌ی پوشش سقف. اجازه دهید که ابتدا سه متغیر تعریف کنیم. این متغیرها خروجی موردنظر را نمایش می‌دهند:

$$\text{Pntcost} = \text{هزینه‌ی رنگ‌آمیزی}$$

$$\text{Flrcost} = \text{هزینه‌ی قرار دادن پوشش سقف}$$

$$\text{Totcost} = \text{هزینهی کل کار}$$

همانطور که در مثال قبیل نیز مشاهده کردید، در اینجا نیز به جای استفاده از نام‌های کوتاه‌تر، از نام متغیرهای «خود مستند» استفاده کرده‌ایم، اما این متغیرها چندان واضح نیستند. شما ممکن است پرسید که چگونه دریافته‌ایم که بهتر است به جای یک متغیر (Totcost) از سه متغیر استفاده شود. واقعیت آن است که در اینجا یک مسأله‌ی طراحی را تجربه کرده‌ایم: اگر این برنامه بتواند به جای یک نوع اطلاعات، سه نوع اطلاعات ارائه دهد، مفیدتر خواهد بود.

همچنین ممکن است بخواهید بدانید که چرا از کلمات خلاصه‌شده‌ای نظیر Pntcost، به جای کلمه‌ی قابل فهم‌تر Paintcost استفاده کرده‌ایم. در اینجا ما پروسه‌ی تبدیل طرح برنامه به ساختار ویژوال بیسیک را پیش‌بینی کرده‌ایم. یک حالت تعادلی بین طول نام متغیر و سهولت استفاده از نام‌های کوتاه‌تر در معادلات وجود دارد.

اکنون باید ورودی مورد نیاز برای تولید خروجی را تعیین کنیم. به ما تعدادی هزینه داده شده است. برای اعمال این هزینه‌ها به مشخصات یک کار بخصوص، باید ابعاد اتاق را داشته باشیم (یعنی طول، عرض و ارتفاع) و نیز باید از طول کار چوبی نیز آگاه باشیم. اینها ورودی‌های ما هستند. اجازه دهید که متغیرهای مربوط به این ورودی‌ها را اعلان کنیم.

$$\text{Length} = \text{طول اتاق}$$

$$\text{Width} = \text{عرض اتاق}$$

$$\text{Height} = \text{ارتفاع اتاق}$$

$$\text{Woodwork} = \text{طول کار چوبی موجود در اتاق}$$

بیان مسأله شامل مقادیر عددی ویژه‌ای است. ما به این مقادیر ثابت که هزینه‌ها هستند، به صورت زیر، نامی توصیفی نسبت خواهیم داد:

$$\text{WallRate} = 10000$$

$$\text{CeilRate} = 20000$$

$$\text{WoodRate} = 10000$$

$$\text{FlrRate} = 30000$$

این برنامه باید مقدار خروجی را از مقادیر ورودی به دست آورد. ما باید فرمول‌هایی که

این محاسبات را انجام می‌دهند، به‌دست آوریم. این فرمول‌ها از فرمول آشنای مساحت مستطیل و نیز برخی حاصل ضرب‌ها به‌دست می‌آیند:

$$\begin{aligned} \text{کل هزینه رنگ آمیزی} &= \text{هزینه رنگ آمیزی هر مترمربع دیوار} * (\text{مساحت دیوارها}) \\ &+ \text{هزینه رنگ آمیزی هر مترمربع سقف} * (\text{مساحت سقف}) \\ &+ \text{هزینه رنگ آمیزی هر مترکارهای چوبی} * \\ &(\text{طول کارهای چوبی}) \end{aligned}$$

$$= \text{WallRate} * (\text{meter of room}) * \text{Height} + \text{CeilRate} * (\text{area of ceiling}) + \text{WoodRate} * (\text{length of woodwork})$$

$$= \text{WallRate} * 2 * (\text{Length} + \text{Width}) * \text{Height} + \text{CeilRate} * \text{Length} * \text{Width} + \text{WoodRate} * \text{WoodWork}$$

کل هزینه نصب پوشش سقف = هزینه نصب هر مترمربع پوشش سقف * (مساحت سقف)

$$\text{FlrCost} = \text{FlrRate} * (\text{area of floor}) = \text{FlrRate} * \text{Length} * \text{Width}$$

$$\text{TotCost} = \text{PntCost} + \text{FlrCost}$$

همانطور که در مثال قبل نیز مشاهده کردید، این الگوریتم شامل: ۱. عبارتهای لازم برای به‌دست آوردن ورودی‌ها از کاربر، ۲. عبارتهای مورد نیاز برای محاسبه‌ی خروجی‌ها با استفاده از فرمول‌های فوق و ۳. عباراتی برای نمایش خروجی‌ها به‌کاربر است.

طراحی برنامه:

از «شبه‌کد» برای ایجاد طرح کلی گام‌های اساسی این برنامه استفاده می‌کنیم. در این مرحله در مورد جزئیات، نگرانی به خود راه ندهید. زیرا ما متعاقباً آن را وارد خواهیم کرد. این برنامه به پنج گام اساسی تقسیم می‌شود:

program Estimate. (به معنی برآورد هزینه)

Main Module: Estimate. (مدول اصلی: برآورد هزینه)

Module 1: Obtain input data. (مدول ۱: دریافت داده‌های ورودی)

Modul 2: Calculate painting cost.

(مدول ۲: محاسبه‌ی هزینه‌های رنگ‌آمیزی)

Module 3: Calculate floor covering cost.

(مدول ۳: محاسبه‌ی هزینه‌ی پوشش سقف)

Module 4: Calculate total cost.

(مدول ۴: محاسبه‌ی کل هزینه)

Module 5: Display output.

(مدول ۵: نمایش خروجی)

این شبه‌کد، دورنمایی از برنامه را ارائه می‌دهد. این دورنما برحسب پنج وظیفه بیان شده است. توجه کنید که در این شبه‌کد، کلیه‌ی ملاحظات مربوط به جزییات، نظیر چگونگی اعمال این وظایف منظور نشده است.

اکنون اجازه دهید که جزییات مربوط به هر وظیفه (task) را وارد کنیم. جزییات مربوط به وظیفه اول، عبارت‌اند از:

Module 1: Obtain input data.

(مدول ۱: دریافت داده‌های ورودی)

Display prompt 'What is the length of the room in meter?'.
(نمایش پیغام «طول اتاق برحسب متر چقدر است؟»)

Input Length.

(دریافت طول)

Display prompt 'What is the width of the room in meter?'.
(نمایش پیغام «عرض اتاق برحسب متر چقدر است؟»)

Input Width.

(دریافت عرض)

Display prompt 'What is the height of the room in meter?'.
(نمایش پیغام «ارتفاع اتاق برحسب متر چقدر است؟»)

Input Height.

(دریافت ارتفاع)

Display prompt 'What is the number of running meter Of
woodwork?'.
(نمایش پیغام «طول کارهای چوبی برحسب متر چقدر است؟»)

Input WoodWork.

(دریافت طول کار چوبی)

آخرین وظیفه (task) دارای عنوان "Display Output" (نمایش خروجی) است. این وظیفه نیز به سادگی قابل اجراست. در زیر، جزییات این وظیفه ارائه شده است:

Module 5: Display Output.

(مدول ۵: نمایش خروجی)

Display title 'Painting cost equals'.
(نمایش پیغام «هزینه‌ی رنگ‌آمیزی برابر است با»)

(نمایش پیغام «هزینه‌ی رنگ‌آمیزی برابر است با»)

Display PntCost. (نمایش هزینه‌ی رنگ‌آمیزی)

Display title 'Floor covering cost equals'.

(نمایش پیغام «هزینه‌ی پوشش سقف برابر است با»)

Display FlrCost. (نمایش هزینه‌ی سقف)

Display blank line. (نمایش یک خط خالی)

Display title 'Total cost of job equals'.

(نمایش پیغام «کل هزینه برابر است با»)

Display TotCost. (نمایش کل هزینه)

توجه کنید که در هر قطعه از داده‌های خروجی، یک عبارت توصیفی قرار داده‌ایم که کاربر را قادر می‌سازد از موضوع آن خروجی اطلاع حاصل کند. هیچ چیز بهبوده‌تر از لیستی از خروجی‌های بدون توضیح نیست!

محاسبه‌ی هزینه‌ی رنگ‌آمیزی به‌وسیله‌ی یکی از فرمول‌های فوق انجام می‌گیرد. در زیر، مدول مربوط به این عمل ارائه شده است:

Module 2: Calculate painting cost.

(مدول ۲: محاسبه‌ی هزینه‌ی رنگ‌آمیزی)

$PntCost = WallRate * 2 * (Length + Width) * Height$

$+ CeilRate * Length * Width + WoodRate * WoodWork$

محاسبات مربوط به هزینه‌ی پوشش سقف نیز به‌وسیله‌ی یکی از فرمول‌های فوق برآورد می‌شود. در زیر، مدول فرعی (submodule) مربوط به این برآورد، ارائه شده است:

Module 3: Calculate floor covering cost.

(مدول ۳: محاسبه‌ی هزینه‌ی پوشش سقف)

$FlrCost = FlrRate * Length * Width$

همچنین جزییات مربوط به محاسبه‌ی هزینه‌ی کل، به‌صورت زیر به‌دست می‌آیند. مدول زیر، این برآورد را انجام می‌دهد:

Module 4: Calculate total cost. (مدول ۴: محاسبه‌ی کل هزینه)

$$\text{TotCost} = \text{PntCost} + \text{FlrCost}$$

مدول‌های متفاوت برآورد هزینه و رابطه‌ی آنها با یکدیگر در جدول ساختاری شکل ۴-۱ ارائه شده‌اند.



شکل ۴-۱: مدول‌های برآورد هزینه‌ی رنگ‌آمیزی

پیاده‌سازی برنامه:

اکنون می‌توانید شبکه‌کد مربوط به برنامه‌ی برآورد هزینه را به ساختارهای برنامه‌نویسی متناسب تبدیل کنید. این تبدیل بسیار شبیه پروسه‌ای خواهد بود که در مورد برنامه‌ی Rectangle (مستطیل) استفاده شده است.

۶-۱ کار با ویژوال بیسیک

محیط ویژوال بیسیک ساده است و به برنامه‌نویسان امکان می‌دهد که برنامه‌های تحت ویندوز خود را بدون نیاز به استفاده از برنامه‌های کاربردی دیگر ایجاد، اجرا و خطایابی کنند. در این فصل مروری بر محیط ویژوال بیسیک^۱ خواهیم داشت. هم‌چنین با تعدادی از مفاهیم اولیه‌ی برنامه‌های ویژوال آشنا خواهیم شد.

۱. در این کتاب، نسخه‌ی Visual Basic 6.0 مورد بحث قرار می‌گیرد.

۱-۶-۱ آشنایی با محیط ویژوال بیسیک

بعد از اجرای برنامه‌ی ویژوال بیسیک Microsoft Visual (Start → All Programs → Microsoft Visual Studio 6 → Microsoft Visual Basic 6.0)، کادر محاوره‌ای New Project به نمایش در می‌آید (شکل ۱-۵)، که این کادر به برنامه‌نویس امکان انتخاب یکی از انواع برنامه‌هایی را می‌دهد که می‌توان در ویژوال بیسیک ایجاد کرد.



شکل ۱-۵ کادر محاوره‌ای New Project

نوع Standard EXE که به‌طور پیش‌فرض در این کادر انتخاب شده است، به برنامه‌نویس امکان می‌دهد که برنامه‌ی اجرایی استاندارد را ایجاد کند (برنامه‌های اجرایی استاندارد از اکثر خصیصه‌های ویژوال بیسیک استفاده می‌کنند). در این کتاب، تمام مثال‌های خود را با استفاده از این نوع برنامه ایجاد خواهیم کرد.

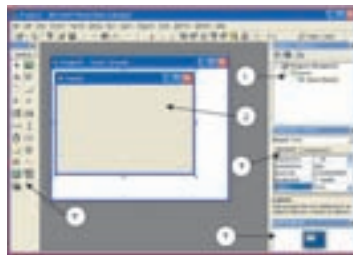
کادر محاوره‌ای New Project دارای سه زبانه (Tab) است:

- زبانه‌ی New: برای ایجاد یک پروژه‌ی جدید
- زبانه‌ی Existing: برای بازکردن پروژه‌ای که از قبل وجود دارد.
- زبانه‌ی Recent: لیستی از آخرین پروژه‌های باز شده یا ایجاد شده را نشان می‌دهد.

پروژه (Project): پروژه عبارت است از مجموعه پرونده‌هایی (فرم، برنامه و ...) که در کل، یک هدف واحد را دنبال می‌کنند. کدهای برنامه، مشخصات ظاهری برنامه و احتمالاً پرونده‌های بانک اطلاعاتی در این مجموعه از پرونده‌ها قرار دارند.

در قسمت پایین کادر محاوره‌ای، کادر علامتی با عنوان Don't show dialog in the future وجود دارد. اگر این کادر علامت‌دار نباشد در هر بار اجرای ویژوال بیسیک، کادر محاوره‌ای موردنظر نمایش داده می‌شود.

برای باز کردن یک پروژه بر روی نشانه موردنظر، دوبار کلیک کرده یا روی نشانه، کلیک کنید، سپس کلید Enter یا دکمه‌ی Open را فشار دهید. با باز شدن پروژه، کادر محاوره‌ای بسته شده و وارد محیط ویژوال بیسیک می‌شویم (شکل ۶-۱). این محیط دارای چندین پنجره، یک نوار منو و یک نوار ابزار است که مشابه نوار منو و ابزار در اکثر برنامه‌های تحت ویندوز است.

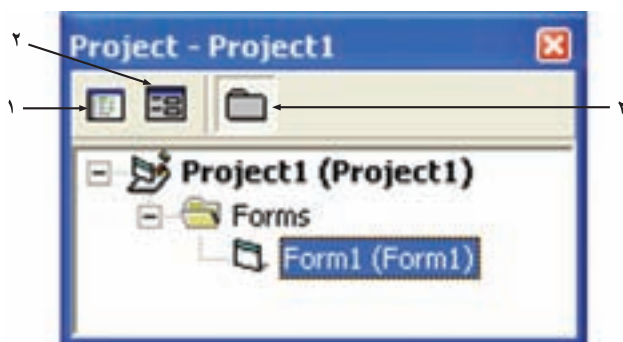


شکل ۶-۱ محیط ویژوال بیسیک پس از انتخاب Standard EXE

پروژه‌ی Standard EXE دارای پنجره‌های زیر است:

۱. پنجره‌ی پروژه (Project)
۲. پنجره‌ی Form Layout
۳. جعبه ابزار (ToolBox)
۴. پنجره‌ی مشخصه‌ها (Properties)
۵. پنجره‌ی فرم (Form)

پنجره‌ی پروژه: پنجره‌ای است که معمولاً به نام Project Explorer نیز معروف بوده و شامل تمام پرونده‌های مربوط به پروژه است (شکل ۷-۱). در صورت عدم مشاهده‌ی این پنجره، کلیدهای Ctrl+R را فشار دهید یا از منوی View گزینه‌ی Project Explorer را انتخاب کنید.



شکل ۷-۱ پنجره‌ی Project

نوار ابزار این پنجره شامل سه دکمه به نام‌های: ۱. View code، ۲. View object و ۳. Toggle Folders است.

- دکمه‌ی View code پنجره‌ای را باز می‌کند که کد (دستورهای برنامه) مربوط به پروژه‌ی فعال را نمایش می‌دهد.
- دکمه‌ی View object شکل ظاهری فرم فعال در پنجره‌ی پروژه را نمایش می‌دهد.

نکته

اگر در پنجره‌ی پروژه هیچ پرونده‌ای فعال نباشد، هر دو دکمه‌ی View code و View object به صورت غیرفعال درمی‌آیند.

- دکمه‌ی Toggle Folders سبب می‌شود که با هر بار فشار آن، پوشه‌ی Forms درون این پنجره به صورت متناوب به نمایش درآمده و پنهان شود.

پنجره‌ی Project یکی از مهم‌ترین ابزارهای مدیریت پروژه است. پنجره‌ی **Form Layout**: این پنجره محل فرم را به هنگام اجرای برنامه (Run time) بر روی صفحه‌ی نمایش مشخص می‌کند (شکل ۱-۸). در صورت عدم مشاهده‌ی این پنجره، از منوی View گزینه‌ی Form Layout Window را انتخاب کنید.

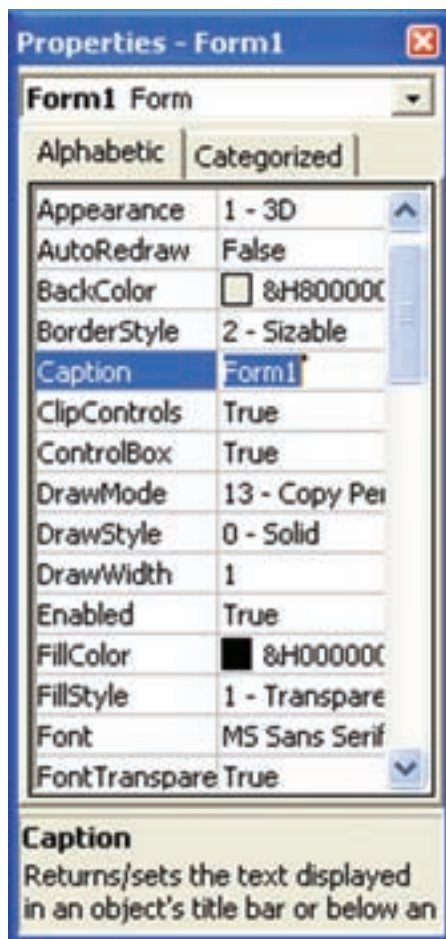


شکل ۱-۸ پنجره‌ی Form Layout

همانطور که در شکل مشاهده می‌کنید، این پنجره یک صفحه‌ی نمایش را نشان می‌دهد که در داخل آن محل قرار گرفتن فرم مشخص شده است. با کشیدن می‌توان فرم را در محل جدید خود قرار داد. به این ترتیب در زمان اجرا، فرم موردنظر در محل مشخص شده ظاهر می‌شود.

پنجره‌ی مشخصه‌ها (Properties): این پنجره ویژگی‌ها و مشخصه‌های فرم یا شیء^۱ را نشان می‌دهد که به ترتیب الفبایی^۲ یا گروهی مرتب شده‌اند (شکل ۱-۹). در صورت عدم مشاهده‌ی این پنجره، کلید F4 را فشار دهید یا از منوی View گزینه‌ی Properties Window را انتخاب کنید.

۱. در برنامه‌سازی شیء‌گرا به متغیری گفته می‌شود که از روال‌ها و داده‌هایی تشکیل شده باشد و به صورت یک موجودیت واحد با آن برخورد شود.
 ۲. مشخصه‌ی Name از این قاعده مستثنی است.



شکل ۱-۹ پنجره‌ی مشخصه‌ها

همانطور که در شکل ۱-۹ مشاهده می‌کنید در قسمت بالای پنجره، جعبه‌ی لیست‌مانندی وجود دارد که در آن نام شیء یا فرمی که مشخصه‌های آن در این پنجره آورده شده است، نمایش داده می‌شود. داخل این لیست نام تمامی شیء‌ها و همچنین نام فرمی که فعال است، آورده شده است. با انتخاب هر شیء یا فرم دیگری از این لیست، مشخصه‌های مربوط به آن در پنجره نشان داده می‌شود. توجه کنید که بعضی از این مشخصه‌ها مشترک هستند، مانند مشخصه‌ی Name که در هر مورد نشان‌دهنده‌ی نام شیء یا فرم است. برخی دیگر، برای کنترل‌ها یا فرم مشترک نیستند بلکه منحصر به فرد هستند.

جعبه ابزار (Tool box): این جعبه (شکل ۱-۱۰) شامل شیء‌هایی است که می‌توان هر یک از آنها را به تعداد دلخواه، بر روی فرم‌های مربوط به پروژه اضافه کرد، به این شرط که نام هر شیء ایجاد شده منحصر به فرد باشد. جدول ۱-۱ اسامی این شیء‌ها را بیان می‌کند.






















شکل ۱-۱۰ جعبه ابزار

نکته

سعی کنید نامی که برای هر شیء در برنامه‌هایتان انتخاب می‌کنید، متناسب با عملکردی باشد که در برنامه خواهد داشت. این نام در مراجعه به شیء موردنظر از طریق کدنویسی، مورد استفاده قرار خواهد گرفت.

جدول ۱-۱ کنترل‌های ویژوال بیسیک

| Icon | Control |
|---|--|
|  | Not a control; enables the mouse pointer |
|  | Label |
|  | TextBox |
|  | CommandButton |
|  | Timer |
|  | PictureBox |
|  | Frame |
|  | CheckBox |
|  | OptionButton |
|  | ComboBox |
|  | ListBox |
|  | Vertical scrollbar |
|  | Horizontal scrollbar |
|  | DriveListBox |
|  | DirListBox |
|  | FileListBox |
|  | Shape |
|  | Line |
|  | Data |

شیء‌های مربوط به جعبه‌ابزار نسبت به نوع پروژه‌ای که در ابتدای ایجاد پروژه‌ی جدید مشخص می‌شود، متغیر است و معمولاً در نوع Standard EXE تعداد این کنترل‌ها بیشتر است. با این حال روش‌هایی برای اضافه کردن کنترل‌هایی که به صورت استاندارد در جعبه ابزار قرار داده نشده‌اند ولی در ویژوال بیسیک موجود است، وجود دارد که در درس‌های برنامه‌سازی ۲ و ۳ در مورد روش اضافه کردن هر یک از آنها به جعبه‌ابزار توضیحاتی داده خواهد شد. برای اضافه کردن یک شیء به فرم، می‌توان به دو روش عمل کرد:

الف) با دوبار کلیک کردن روی هر شیء، نمونه‌ای از شیء مربوطه در وسط فرم فعال، ظاهر می‌شود که همیشه از نظر اندازه و موقعیت ثابت است. با کشیدن شیء، می‌توان محل آن را تغییر داد. همچنین می‌توان با قرار دادن مکان‌نما در گوشه‌های شیء و کشیدن، اندازه‌ی آن شیء را نیز به دلخواه تعیین کرد.

ب) شیء را در جعبه‌ابزار انتخاب کرده و سپس بر روی فرم موردنظر در محل دلخواه کلیک کرد و با کشیدن شیء را به اندازه‌ی دلخواه بر روی فرم اضافه کرد.

پنجره‌ی فرم‌ها (Forms): این پنجره، فرم فعال در پنجره‌ی پروژه را با تمام شیء‌های مربوط به آن، در یک رابط گرافیکی کاربر^۱ (GUI) نشان می‌دهد.

در ابتدای ایجاد یک فرم جدید، هیچ شیئی در آن وجود ندارد. البته در صورتی که دکمه‌ی View code در پنجره‌ی پروژه انتخاب شود یا روی هر شیء یا فرم دوبار کلیک شود، پنجره‌ی مربوط به کد، در این قسمت نمایش داده می‌شود.

نوار منو (Menu bar): مکانی است که در اکثر برنامه‌های تحت ویندوز وجود دارد و شامل دستوراتی برای ساخت، نگهداری و راه‌اندازی برنامه‌هاست (شکل ۱-۱۱). با توجه به نوع ویژوال بیسیکی که در اختیار دارید، ممکن است برخی از گزینه‌های این منوها با آنچه که در شکل ۱-۱۱ مشاهده می‌کنید، متفاوت باشد. جدول ۱-۲ وظایف هر بخش از منو را به طور خلاصه ارایه کرده است.



شکل ۱-۱۱

جدول ۱-۲ وظایف هر بخش از منو

| وظایف هر بخشی از منو | زیرمنو |
|---|---------|
| باز کردن، ذخیره و چاپ پروژه | File |
| کپی، حذف و غیره | Edit |
| نحوه‌ی نمایش پنجره‌های محیط ویژوال بیسیک | View |
| افزودن خصیصه‌هایی مانند فرم‌ها به یک پروژه | Project |
| تنظیم شیء‌های موجود بر روی فرم | Format |
| خطایابی | Debug |
| اجرا، متوقف کردن برنامه و ... | Run |
| بازیابی داده‌ها از پایگاه داده‌ها | Query |
| ویرایش و اصلاح در طراحی پایگاه‌های داده | Diagram |
| ابزارهای ویژوال بیسیک و بهینه‌سازی محیط کار | Tools |
| نصب و حذف برنامه‌های افزودنی | Add_ins |
| مرتب کردن و نمایش پنجره‌ها | Window |
| راهنمای کاملی برای کاربر (در صورت نصب MSDN) | Help |

در پایین نوار منو، نوار ابزاری وجود دارد (شکل ۱-۱۱) که دارای دکمه‌های معادل گزینه‌های منوهاست و به وسیله‌ی آنها می‌توان به سرعت برخی از دستورهای موجود در منو را اجرا کرد.

۱-۷ مشخصه‌ها، متدها و رویدادها

هر شیئی مثل فرم یا کنترل، دارای مجموعه‌ای از مشخصه‌هاست که آن را توصیف می‌کند. اگرچه این مجموعه برای همه‌ی شیء‌ها یکسان نیست، ولی بعضی از مشخصه‌ها (مانند آنهایی که در جدول ۱-۳ لیست شده‌اند) برای اغلب کنترل‌ها یکسان هستند. مشخصه‌های مربوط به هر کنترل را می‌توان در پنجره‌ی Properties مشاهده کرد.

مشخصه‌ی مهم دیگر `BorderStyle, form` است که عناصر پنجره‌ای (نوار عنوان، دکمه‌های حداکثر و حداقل‌رسانی و غیره) را که فرم خواهد داشت، تعیین می‌کند. جدول ۱-۴، شش مقدار این مشخصه را ارائه می‌کند.

متدها، بلوک‌های کد هستند که برای بیان اینکه کنترل در مقابل رویدادها چگونه عمل کند، در داخل کنترل طراحی شده‌اند (مثل جابه‌جایی به محل دیگری در روی فرم به وسیله‌ی متد

(Move). همه‌ی کنترل‌ها مانند مشخصه‌ها، دارای متدهای یکسانی نیستند، اگرچه بعضی از متدها برای اغلب کنترل‌ها متداول هستند (جدول ۱-۵).

جدول ۱-۳ مشخصه‌های متداول کنترل‌های ویژوال بیسیک

| مشخصه | توضیح |
|---------|---|
| Left | فاصله‌ی سمت چپ کنترل از دربرگیرنده‌ی خودش |
| Top | فاصله از بالای کنترل نسبت به دربرگیرنده |
| Height | بلندی کنترل |
| Width | پهنای کنترل |
| Name | رشته‌ی مورد استفاده برای اشاره به کنترل (نام کنترل) |
| Enabled | یک مقدار منطقی که تعیین می‌کند آیا کار بران می‌توانند روی کنترل کار کنند یا نه؟ |
| Visible | یک مقدار منطقی که تعیین می‌کند آیا کار بران می‌توانند کنترل را مشاهده کنند یا نه؟ |

جدول ۱-۴ مقادیر مشخصه‌ی Border Style در form

| مقدار مشخصه | توضیح |
|----------------------|---|
| 0-None | بدون حاشیه و نوار عنوان و غیر قابل جابه‌جایی. |
| 1-Fixed Single | با کشیدن حاشیه‌ها نمی‌توان تغییر اندازه داد. |
| 2-Sizable (default) | دارای قابلیت تغییر اندازه به وسیله‌ی کشیدن و دکمه‌های Maximize و Minimize. |
| 3-Fixed Dialog | عدم قابلیت تغییر اندازه و عدم وجود دکمه‌های Maximize و Minimize. |
| 4-Fixed ToolWindow | شبه حالت قبل با این تفاوت که نوار عنوان کوتاه‌تر است و قلم نوار عنوان و دکمه‌ی Close کوچک‌تر است. |
| 5-Sizable ToolWindow | شبه حالت فوق با این تفاوت که با کشیدن حاشیه تغییر اندازه ممکن است. |

جدول ۱-۵

| شیوه | کاربرد |
|----------|---|
| Move | محل شیء را تغییر می‌دهد. |
| Drag | اجرای عمل کشیدن و رها کردن به وسیله‌ی کاربر را مدیریت می‌کند. |
| SetFocus | فوکوس را به شیء تعیین شده انتقال می‌دهد. ^۱ |

۱. انتقال فوکوس یعنی فعال کردن شیء تعیین شده.

رویدادها، رخدادهایی در داخل و بیرون از برنامه هستند. به عنوان مثال، هنگامی که کاربر روی دکمه‌ی فرمانی کلیک می‌کند، چندین رویداد رخ می‌دهد: دکمه‌ی ماوس فشار داده می‌شود، روی `CommandButton` در برنامه، کلیک می‌شود و سپس دکمه‌ی ماوس رها می‌شود. این سه عمل به ترتیب عبارت‌اند از: رویدادهای `MouseDown`، `Click` و `MouseUp`. در این فرایند، رویداد `GotFocus` برای `CommandButton` و رویداد `LostFocus` برای شیئی که قبلاً فوکوس داشته است، رخ می‌دهد.

همه‌ی کنترل‌ها نیز دارای رویدادهای یکسانی نیستند، ولی بعضی از رویدادها بین کنترل‌ها مشترک هستند. این رویدادها به صورت نتیجه‌ی چندین عمل خاص کاربر، مثل جابه‌جایی ماوس، فشار دادن کلیدی از صفحه‌کلید، یا کلیک کردن روی یک کادر متن، رخ می‌دهند (جدول ۱-۶). البته در این مورد استثناهایی نیز وجود دارد، مثل رویداد `timer` مربوط به شیء `timer`.

جدول ۱-۶

| رویداد | توضیح |
|-----------|--|
| Change | کاربر متن کنترل کادر متن یا کادر ترکیبی را تغییر می‌دهد. |
| Click | کاربر روی شیئی کلیک می‌کند. |
| DbClick | کاربر روی شیئی دوبار کلیک می‌کند. |
| DragDrop | کاربر شیئی را به محل دیگری می‌کشد. |
| DragOver | کاربر شیئی را روی کنترل دیگری می‌کشد. |
| GotFocus | یک شیء، فوکوس را دریافت می‌کند. |
| KeyDown | کاربر کلیدی را هنگامی که شیئی فوکوس دارد، فشار می‌دهد. |
| KeyPress | کاربر کلیدی را هنگامی که شیئی فوکوس دارد، فشار می‌دهد و رها می‌کند. |
| KeyUp | کاربر کلیدی را هنگامی که شیئی فوکوس دارد، رها می‌کند. |
| LostFocus | یک شیء، فوکوس را از دست می‌دهد. |
| MouseDown | کاربر یکی از دکمه‌های ماوس را همزمان با اینکه اشاره‌گر ماوس روی یک شیء است، فشار می‌دهد. |
| MouseMove | کاربر اشاره‌گر ماوس را روی یک شیء انتقال می‌دهد. |
| MouseUp | کاربر یکی از دکمه‌های ماوس را همزمان با اینکه اشاره‌گر ماوس روی یک شیء است، رها می‌کند. |

نکته

رویدادهای GotFocus و LostFocus به اغلب رویدادهای دیگر مرتبط هستند زیرا این رویدادها هنگامی که کنترل جدیدی برای دریافت ورودی کاربر فعال می‌شود، رخ می‌دهند.

نکته

در ویژوال بیسیک، مشخصه‌های Right و Bottom وجود ندارد. برای تعیین موقعیت یک شیء، از مشخصه‌های Top, Height, Left, Width استفاده کنید.

۸-۱ برنامه‌نویسی رویدادگرا^۱

هنگامی که برنامه‌ای را در ویژوال بیسیک ایجاد می‌کنید، از برنامه‌نویسی رویدادگرا استفاده خواهید کرد. در برنامه‌نویسی رویدادگرا، کدی که می‌نویسید هنگامی اجرا خواهد شد که کاربر کاری انجام دهد یا هنگامی که در ویندوز رویدادی رخ دهد. البته برنامه‌نویسی با این روش مستلزم این است که رویدادهایی را که در برنامه رخ می‌دهند، بدانید و کدی بنویسید که پاسخ مناسبی برای این رویداد باشد.

خوشبختانه ویندوز و ویژوال بیسیک، اغلب کارها را انجام می‌دهند. هر زمانی که رویدادی واقع شود، ویندوز پیامی را به برنامه ارسال می‌کند. برنامه این پیام را خوانده و سپس کد مربوط به رویداد را اجرا می‌کند. اگر کدی را برای رویداد تعیین نکنید، برنامه از رویداد صرف‌نظر خواهد کرد.

به‌طور کلی، این کد را روال (Procedure) می‌نامند که به‌صورت بلاکی از کد که می‌تواند از داخل برنامه‌ی کاربردی فراخوانی شود، تعریف می‌شود. این کد ممکن است برای جابه‌جایی شیء‌ها روی فرم، محاسبه‌ی مقدار یک فرمول، یا نوشتن داده‌ها در یک بانک اطلاعاتی مورد استفاده قرار گیرد.

یک روال همیشه از قالب زیر استفاده می‌کند که در این کتاب و درس‌های برنامه‌سازی ۲ و ۳ با این مفهوم به‌طور کامل آشنا خواهید شد:

```
[Public|Private] [Static] Sub|Function|Property_
```

```
function_name (arguments) [As Type]
```

```
{ ...Your procedure code... }
```

```
End Sub|Function|Property
```

یک روال رویداد، در محلی از پروژه است که هنگام رخ دادن رویداد، کد مربوطه اجرا خواهد شد. برای نوشتن یک روال رویداد، باید به پنجره‌ی Code دسترسی داشته باشید. هر شیء یا کنترل دارای یک رویداد پیش فرض است که با انجام یکی از این عملیات، کد مربوط به آن باز می‌شود:

- دوبار کلیک روی شیء
- انتخاب شیء با ماوس و فشار دادن کلید F7
- انتخاب شیء و انتخاب گزینه‌ی Code از منوی View
- انتخاب شیء فرم در Project Explorer، کلیک کردن روی دکمه‌ی View Code و انتخاب شیء از پنجره‌ی Code.

ویژوال بیسیک به‌طور خودکار، هنگام انتخاب رویدادی در پنجره‌ی Code، روال آن رویداد را تولید می‌کند.

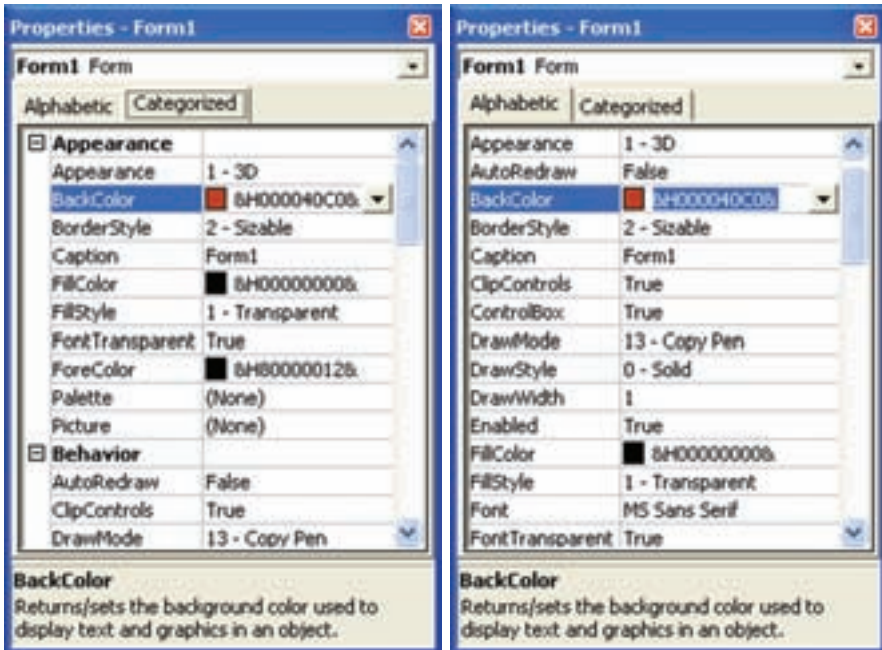
۹-۱ تغییر مشخصه

برای تغییر مشخصه‌های یک کنترل، می‌توان از پنجره‌ی مشخصه‌ها استفاده کرد. به‌عنوان مثال، برای تغییر رنگ زمینه‌ی فرم، ابتدا از پنجره‌ی پروژه، کنترل فرم را انتخاب کرده و سپس از پنجره‌ی properties مشخصه‌ی BackColor را انتخاب می‌کنیم. با بازکردن کادر لیست و انتخاب زبانه‌ی Palette می‌توان رنگ دلخواه را تنظیم کرد (شکل ۱۲-۱).

پنجره‌ی مشخصه‌ها دارای دو زبانه‌ی Alphabetic و Categorized است. زبانه‌ی Alphabetic لیست مشخصه‌ها را به ترتیب حروف الفبا نشان می‌دهد و زبانه‌ی Categorized لیست را بر اساس نوع عملکرد مشخصه‌ها نشان می‌دهد (شکل ۱۳-۱). در این طبقه‌بندی، مشخصه‌ها مانند جدول ۷-۱ دسته‌بندی می‌شوند.



شکل ۱-۱۲



شکل ۱-۱۳

جدول ۱-۷

| | |
|---|-----------------------------|
| مشخصه‌های ظاهری و نمایشی | Appearance |
| مشخصه‌های رفتاری | Behavior |
| مشخصه‌ی مربوط به انتقال اطلاعات بین برنامه‌های کاربردی دیگر | DDE (Dynamic Data Exchange) |
| مشخصه‌ی قلم | Font |
| مشخصه‌های متفرقه | Misc (Miscellaneous) |
| مشخصه‌های محل قرارگیری و ابعاد | Position |
| مشخصه‌های مربوط به اندازه و سیستم اندازه‌گذاری | Scale |

شکل ۱-۱۴ قسمتی از نوار ابزار را نشان می‌دهد که به ترتیب از چپ به راست برای همین منظور مورد استفاده قرار می‌گیرد.



شکل ۱-۱۴ به وسیله‌ی نوار ابزار نیز می‌توان پنجره‌های Project، Properties و Form Layout را نمایان کرد.

۱-۱۰ اجرای برنامه

برای اجرای یک برنامه، روش‌های متفاوتی وجود دارد که متداولترین آنها عبارت‌اند از:

الف) فشار دادن کلید F5

ب) باز کردن منوی Run و انتخاب گزینه‌ی Start

پ) انتخاب ابزار Start (▶)

تمرین

۱. مشخصه‌ی BorderStyle یک فرم را تغییر دهید و با اجرای فرم، مقدار مشخصه را مشاهده کنید.
۲. با تغییر مشخصه‌ی Caption و Name، کاربرد و تفاوت هر دو را بیان کنید.
۳. آیا می‌توان به صورت دستی مشخصه‌ی رنگ یک کنترل را تنظیم کرد؟ چگونه؟

۱-۱۱ انواع پرونده‌ها در ویژوال بیسیک

۱. **پرونده‌ی پروژه:** این پرونده با پسوند (Visual Basic Project) VBP ذخیره می‌شود و محتوای آن مشخصات پروژه، نوع پروژه، نام پرونده‌های فرم و فرم اصلی و ... است.
۲. **پرونده‌ی محیط کاری:** این پرونده با پسوند (Visual Basic WorkSpace) VBW ذخیره می‌شود و محتوای آن، اطلاعات محیط کاری و فرم‌های پروژه است.
۳. **پرونده‌ی فرم:** این پرونده با پسوند FRM ذخیره می‌شود و محتوای آن اطلاعات یک فرم و تمام مشخصات فرم به‌همراه نام و مشخصات کنترل‌های روی فرم است، در ضمن تمام رویدادها و کدهای مربوط به آن نیز در این پرونده ذخیره می‌شوند.
۴. **پرونده‌ی تصاویر:** این پرونده با پسوند FRX ذخیره می‌شود و محتوای تصاویری است که روی فرم یا کنترل‌های دیگر از آنها استفاده شده است.
۵. **پرونده‌های OCX, DLL:** و ... که در آینده با آنها آشنا خواهید شد.

نکته

در حالت استاندارد، پرونده‌های فرم با نام Form1، Form2 و ... ذخیره می‌شوند و بهتر است برای درک بهتر، نام مناسبی برای هر فرم در نظر گرفته شود. در ضمن برای جلوگیری از تداخل پرونده‌های یک پروژه با پروژه‌ی دیگر، بهتر است هر پروژه را در یک پوشه‌ی مناسب و جداگانه ذخیره کنید.

خلاصه‌ی فصل

روش‌های برنامه‌نویسی از ابتدا تاکنون، پیشرفت قابل‌ملاحظه‌ای داشته‌اند و سیر تکاملی آنها را می‌توان به صورت زیر فهرست کرد:

۱. روش نامشخص

۲. زیرروالی

۳. ساخت یافته

۴. مدولار

۵. شیء‌گرا

۶. رویدادگرا

زبان‌های برنامه‌نویسی را می‌توان بر اساس نزدیکی به زبان ماشین به سه دسته‌ی زیر تقسیم کرد:

۱. سطح پایین

۲. سطح میانی

۳. سطح بالا

زبان‌های برنامه‌نویسی را از نقطه‌نظر گرافیکی، به زبان‌های متنی و گرافیکی (ویژوال) تقسیم می‌کنند که ویژوال بیسیک نیز جزو گروه دوم محسوب می‌شود.

برای حل مسایل، باید سه گام اساسی زیر را طی کنید:

گام ۱: تحلیل مسأله

گام ۲: طراحی برنامه

گام ۳: پیاده‌سازی برنامه

در گام تحلیل مسأله، داده‌های ورودی، اطلاعات خروجی و الگوریتم تبدیل ورودی‌ها به خروجی‌ها تعیین می‌شوند.

در گام طراحی برنامه، از شبه‌کدها برای حل مسأله استفاده خواهیم کرد.

درگام به‌کارگیری برنامه، کد اصلی برنامه با زبان برنامه‌نویسی موردنظر پیاده‌سازی می‌شود. ویژوال بیسیک برای طراحی ظاهر برنامه‌ی کاربردی و نوشتن کد اصلی دارای محیط IDE است. با انتخاب پروژه از نوع Standard EXE، محیط ویژوال بیسیک دارای اجزای زیر خواهد بود:

۱. پنجره‌ی پروژه (Project)

۲. پنجره‌ی Form Layout

۳. جعبه‌ابزار (ToolBox)

۴. پنجره‌ی مشخصه‌ها (Properties)

۵. پنجره‌ی فرم (Form)

مشخصه‌ها شیء را توصیف می‌کنند. متدها سبب می‌شوند که شیء کاری انجام دهد. رویداد، عملی است که هنگام انجام کاری روی شیء رخ می‌دهد. روال رویداد، عکس‌العمل تعیین‌شده به‌صورت کد است که به رویداد پاسخ می‌دهد.

خودآزمایی

۱. حجم یک جعبه‌ی مستطیلی به وسیله‌ی فرمول زیر به دست می‌آید:

$$\text{Volume} = \text{Length} * \text{Width} * \text{Height}$$

شبه‌کدی بنویسید که حجم جعبه‌ای را که ابعاد آن به وسیله‌ی کاربر تعیین می‌شود، محاسبه کند.

۲. فرض کنید یک شرکت بسته‌بندی برای بسته‌بندی کارتن‌ها، هزینه‌ای معادل ۱۰۰۰ ریال به ازای هر متر مربع از سطوح زیرین و جانبی و ۱۶۰۰ ریال به ازای هر مترمربع از سطح فوقانی کارتن دریافت می‌کند. شبه‌کدی بنویسید که هزینه‌ی چنین بسته‌بندی را بر اساس ابعادی که کاربر وارد می‌کند، محاسبه کند.

۳. حجم یک کره به وسیله‌ی فرمول زیر به دست می‌آید:

$$\text{Volume} = (4/3) * \text{Pi} * R^3$$

که در آن R شعاع کره است. $\text{Pi} \simeq 3.14159$ و برنامه‌ای طراحی کنید که حجم یک کره به شعاع تعیین‌شده توسط کاربر را محاسبه کند.

۴. یک سوپرمارکت بسته‌های محتوی کنسروهای گوناگون را به فروش می‌رساند. در این بسته‌ها یکی از این چهار نوع کنسرو وجود دارد: کنسرو ۱ هر کیلو ۲۴۰۰ ریال، کنسرو ۲ هر کیلو ۳۵۰۰ ریال، کنسرو ۳ هر کیلو ۸۰۰ ریال، و نهایتاً کنسرو ۴ هر کیلو ۴۵۰۰ ریال. شبه‌کدی بنویسید که قیمت کنسروهای یک بسته را برحسب وزن‌های هر نوع (بر اساس آنچه که کاربر تعیین می‌کند)، محاسبه کند.

۵. برای رنگ‌آمیزی مخازن استوانه‌ای شامل مواد شیمیایی به یک شرکت پیمانکار شده است که برای رنگ‌آمیزی وجوه مخزن‌ها هر متر مربع ۴۰۰۰ ریال و برای رنگ‌آمیزی قاعده‌ی مخزن‌ها هر متر مربع ۴۰۰ ریال دریافت کند. علاوه بر این هزینه‌ی نقاشی نام شرکت بر روی وجوه مخزن ۲۰۰۰ ریال هزینه برآورد شده است. شبه‌کدی بنویسید که صورت حساب مربوط به رنگ‌آمیزی یک مخزن را محاسبه کند. کاربر باید شعاع مخزن و ارتفاع آن را وارد کند. مساحت وجوه مخزن به وسیله‌ی فرمول زیر محاسبه می‌شود:

$$\text{SideArea} = 2 * \text{pi} * \text{Radius} * \text{Height}$$

مساحت قاعده‌ی فوقانی نیز به وسیله‌ی فرمول زیر محاسبه می‌شود:

$$\text{TopArea} = \text{pi} * \text{Radius}^2$$

که در آنها $\text{pi} \simeq 3.14159$ است.

فصل دوم

انواع داده‌ها

پس از پایان این فصل، انتظار می‌رود که فراگیر بتواند:

- داده، متغیر و ثابت را شرح داده و انواع آنها را نام ببرد.
- نحوه‌ی اعلان متغیر و ثابت را بیان کند.
- میزان حافظه و محدوده‌ی عددی انواع داده‌ها را بیان کند.
- نحوه‌ی مقداردهی به متغیرها را بیان کند و انجام دهد.
- انواع عملگرها و ترتیب اولویت آنها را شرح دهد.
- `CommandButton` را در برنامه‌های خود به‌کار ببرد و مشخصه‌ها و رویدادهای آن را شرح دهد.
- از توابع ورودی و خروجی در برنامه‌های خود استفاده کند و آرگومان‌های آنها را شرح دهد.
- متدهای `Print` و `cls` را شرح دهد و در برنامه‌های خود به‌کار ببرد.

هر زبان برنامه‌نویسی برای پردازش داده‌ها، به انواع مختلفی از داده‌ها نیاز دارد و ویژوال بیسیک هم از این قاعده مستثنی نیست. ویژوال بیسیک از انواع داده‌های مختلف پشتیبانی می‌کند که می‌توانند نیازهای متعدد برنامه‌نویس را برآورده سازند. به‌طور کلی، می‌توان داده‌ها را به دو نوع عددی و غیرعددی تقسیم کرد.

۲-۱ داده‌های عددی

تمام انواع داده‌های عددی در یکی از دو گروه زیر قرار دارند:

- **اعداد صحیح (Integer):** اعداد صحیح بدون نقطه‌ی اعشاری؛ مانند ۶۱۴، ۰، ۹۳۴-.
 - **اعداد اعشاری (Decimal):** اعداد با نقطه‌ی اعشاری (ممیز)؛ مانند ۷۰۹/۸، ۰/۰۵، ۰/۰۷، ۳۵۵/۴۰۲-.
- به اعداد اعشاری، اعداد ممیز شناور هم گفته می‌شود. در تمام اعداد اعشاری باید ممیز اعشار وجود داشته باشد، حتی اگر رقم‌های بعد از آن صفر باشند.
- ویژوال بیسیک اعداد اعشاری و صحیح را به روش‌های مختلف ذخیره و با آنها کار می‌کند. با آنکه برای کار بر ۸ و ۸۷۰۰ هیچ فرقی ندارد، ولی از نظر ویژوال بیسیک آنها متفاوت هستند. مقدار حافظه‌ای که انواع داده‌های مختلف به خود اختصاص می‌دهند، یکسان نیست. با نگاه کردن به یک عدد نمی‌توان گفت که چقدر حافظه اشغال خواهد کرد. مثلاً، اعداد ۲۹۹۹۹ و ۷۰۱ هر دو به یک مقدار حافظه می‌گیرند. با وجود اینکه امروزه، دیگر حافظه یک مشکل کلیدی نیست و شما هم به‌عنوان برنامه‌نویس نباید زیاد نگران آن باشید، ولی همیشه سعی کنید برای داده‌هایتان نوعی انتخاب کنید که حافظه‌ی کمتری را اشغال کنند.
- در جدول ۱-۲ شش نوع داده‌ی عددی و ویژوال بیسیک، مقدار حافظه‌ی مورد نیاز هر یک و محدوده‌ای را که می‌توانند در خود جای دهند، مشاهده می‌کنید. هنگام تعریف داده‌ها این جدول را مدنظر داشته باشید. به‌عنوان مثال، اگر می‌خواهید با اعداد منفی کار کنید نباید از نوع Byte استفاده کنید، اما اگر با سن افراد سر و کار دارید، این نوع بهترین انتخاب ممکن است.

جدول ۱-۲ شش نوع داده‌ی عددی و ویژوال بیسیک

| نوع داده | میزان حافظه‌ی موردنیاز | محدوده‌ی مقادیر |
|----------|------------------------|--|
| Byte | ۱ بایت | 0 تا 255 |
| Integer | ۲ بایت | -32,768 تا 32,767 |
| Long | ۴ بایت | تقریباً $-2.1E9$ / + |
| Single | ۴ بایت | اعداد منفی: $-3.402823E38$ تا $-1.401298E-45$ اعداد مثبت: $1.401298E-45$ تا $3.402823E38$ |
| Double | ۸ بایت | اعداد منفی: $-1.79769313486232E308$ تا $-4.94065645841247E-324$ اعداد مثبت: $4.94065645841247E-324$ تا $1.79769313486232E308$ |
| Currency | ۸ بایت | 922,337,203,685,477.5807 تا -922,337,203,685,477.5807 (چهار رقم اعشار برای دقت محاسبات است.) |

وقتی در یک برنامه، عددی را صریحاً می‌نویسید، ویژوال بیسیک مناسبترین نوع را برای آن برمی‌گزیند ولی گاهی لازم است داده‌ی عددی مورد استفاده از نوعی باشد که شما دارید، نه آنچه که ویژوال بیسیک تعیین می‌کند. در چنین مواردی می‌توانید نوع داده را صریحاً به ویژوال بیسیک معرفی کنید. این کار با استفاده از پسوند نوع داده (Data-type suffix) امکان‌پذیر است. جدول ۲-۲ انواع پسوندهای عددی در ویژوال بیسیک را نشان می‌دهد.

جدول ۲-۲ پسوندهای عددی ویژوال بیسیک

| نوع داده | پسوند |
|----------|-------|
| Integer | % |
| Long | & |
| Single | ! |
| Double | # |
| Currency | @ |

۲-۲ سایر انواع داده‌ها

درک داده‌های غیر عددی (به طور معمول) آسان‌تر است. یکی از دلایلی که BASIC، علیرغم حضور زبان‌های پیشرفته‌تر همچنان مطرح مانده است، توانایی‌های آن در کار با رشته‌های متنی است. رشته (String) ترکیبی است از چند نویسه^۱، که حتی می‌توانند رقم عددی باشند ولی نمی‌توان روی آنها محاسبه انجام داد. نام، آدرس، شماره تلفن یا حساب بانکی را می‌توان به صورت رشته نمایش داد. همواره سعی کنید فقط برای اعدادی که نیاز به محاسبه دارند، از انواع عددی استفاده کنید. در جدول ۲-۳ انواع داده‌ی غیر عددی ویژوال بیسیک را مشاهده می‌کنید.

متغیر Boolean (که به افتخار ریاضیدان انگلیسی، جرج بول نام‌گذاری شده است) فقط می‌تواند دو مقدار بگیرد: True (درست) یا False (نادرست).

داده‌های رشته‌ای (String literal) همیشه بین دو علامت نقل قول (" ") قرار می‌گیرند و می‌توانند شامل هر نویسه‌ای باشند. مثال‌های زیر همگی رشته هستند:

"Oh me, oh my"

"543-00-0324"

"1020 S.Yale Avenue"

1. character

جدول ۲-۳ انواع داده‌ی غیر عددی و ویژال بیسیک

| نوع داده | مقدار حافظه | محدوده |
|--------------------|--------------------|-------------------------------------|
| String (طول ثابت) | طول رشته | از ۱ تا تقریباً ۶۵۴۰۰ نویسه |
| String (طول متغیر) | طول رشته + ۱۰ بایت | ۰ تا ۲ میلیارد نویسه |
| Date | ۸ بایت | از اول ژانویه ۱۰۰ تا ۳۱ دسامبر ۹۹۹۹ |
| Boolean | ۲ بایت | True یا False |
| Object | ۴ بایت | معادل شیء تعریف شده |
| Variant (عددی) | ۱۶ بایت | هر عددی تا Double |
| Variant (متن) | طول رشته + ۲۲ بایت | مانند (طول متغیر) String |

هر چیزی که بین دو علامت "" قرار گیرد، یک رشته است حتی اگر هیچ نویسه‌ای در آن نباشد. رشته‌ای که طول آن صفر باشد، رشته‌ی Null نامیده می‌شود. هنگام استفاده از مقادیری که شامل تاریخ و زمان هستند، از علامت # در ابتدا و انتهای این مقادیر، استفاده کنید. و ویژال بیسیک از تمام قالب‌های تاریخ و زمان پشتیبانی می‌کند. به مثال‌های زیر توجه کنید:

#July 4, 1776#

#7:11 pm#

#19:11:22#

#1-2-2003#

#5-Dec-99#

نوع داده‌ی Boolean برای مواردی مناسب است که فقط دو مقدار مخالف هم (True یا False, Yes یا No و از این قبیل) دارند. مشخصه‌ی Enabled کنترل‌ها، از این نمونه است. نوع داده‌ی Variant می‌تواند هر مقداری (بجز رشته‌های با طول ثابت) را در خود جای دهد. هنگامی از این نوع داده استفاده کنید که از قبل، دقیقاً نمی‌دانید با چه نوع داده‌ای سر و کار خواهید داشت.

به احتمال زیاد تا به حال چیزهایی درباره‌ی مشکل سال ۲۰۰۰ (باگ Y2K) شنیده‌اید. این مشکل از ذخیره کردن دو رقم آخر سال (به جای هر چهار رقم آن) ناشی شده بود. ویژال بیسیک درون خود، همیشه از چهار رقم برای نمایش سال استفاده می‌کند. بنابراین، برنامه‌هایی که با ویژال بیسیک نوشته شده بودند، در پشت سر گذاشتن هزاره‌ی دوم میلادی و ورود به هزاره‌ی سوم دچار هیچ مشکلی نشدند.

۲-۳ متغیرها

متغیر (Variable) مکانی در حافظه است که برای نگهداری یک مقدار، مورد استفاده قرار می‌گیرد. مقداری که در متغیر قرار داده می‌شود، قابل تغییر است (نام آن هم بر همین ویژگی دلالت دارد). وقتی مقداری را در یک متغیر قرار می‌دهید، مقدار قبلی آن از بین خواهد رفت. متغیرها با نامشان شناخته می‌شوند، بنابراین در یک قسمت از برنامه (روال) استفاده از دو متغیر با یک نام مجاز نیست، زیرا ویژوال بیسیک قادر به تشخیص آنها نخواهد بود. برخلاف کنترل‌ها که ویژوال بیسیک نام پیش‌فرضی را برای آنها در نظر می‌گیرد، نام‌گذاری متغیرها از همان ابتدا بر عهده‌ی شما (برنامه‌نویس) است. قبل از استفاده از یک متغیر بهتر است آن را اعلان (declare) کنید. اعلان یک متغیر، یعنی نام‌گذاری آن و تعیین نوع مقداری که می‌تواند بگیرد. متغیرها فقط می‌توانند از همان نوعی که اعلان شده‌اند مقدار بگیرند (به‌استثنای متغیرهای Variant که می‌توانند از تمام انواع داده مقدار بگیرند).

۲-۳-۱ اعلان متغیرها

برای اعلان یک متغیر (نام‌گذاری و تعیین نوع آن) از کلمه‌ی کلیدی `Dim`^۱ استفاده می‌شود. قبل از استفاده از یک متغیر حتماً باید آن را اعلان کرد. البته ویژوال بیسیک اجازه می‌دهد که این قاعده‌ی کلی را زیر پا بگذارید ولی تخلف از این قاعده می‌تواند به سردرگمی منجر شود. الزام (یا عدم الزام) به اعلان متغیرها را می‌توانید در منوی `Tools|Options`، زبانه‌ی `Editor` و گزینه‌ی `Require Variable Declaration` مشخص کنید. اگر این گزینه انتخاب شود، در قسمت تعاریف پنجره‌ی کد^۲، دستور `Option Explicit` به صورت پیش‌فرض نوشته می‌شود. این دستور به ویژوال بیسیک می‌گوید که در کل مدول مورد بحث، متغیرها قبل از استفاده باید تعریف شوند. در چنین مدول‌هایی هر گاه نام یک متغیر را اشتباه بنویسید، ویژوال بیسیک آن را به شما گوشزد خواهد کرد. اما اگر این گزینه را غیرفعال کرده باشید، ویژوال بیسیک اشتباه در نوشتن نام یک متغیر را متغیر جدیدی تلقی کرده و به کار خود ادامه خواهد داد. در این حالت تمام متغیرهایی که اعلان نشوند، از نوع `Variant` در نظر گرفته خواهند شد. شکل کلی استفاده از دستور `Dim` برای اعلان یک متغیر چنین است:

```
Dim VarName As DataType
```

```
Dim نام متغیر As نوع داده
```

۱. کلماتی است که در زبان‌های برنامه‌نویسی رزرو شده هستند، مانند نام دستورات و توابع VB.

۲. پنجره‌ای است که در آن می‌توان کد برنامه را نوشت.

یا

Dim VarName نوع پسوند

Dim A%

که در آن VarName نام متغیر و DataType یکی از انواع داده‌ی ویژوال بیسیک (جدول‌های ۲-۱ و ۲-۳) است. متغیرهای مورد استفاده در یک روال باید در همان ابتدای روال تعریف شوند. متغیرهای تعریف شده در یک روال فقط در همان روال قابل استفاده‌اند و در هیچ روال دیگری قابل دسترس نیستند؛ به این قبیل متغیرها، متغیر محلی (Local Variable) گفته می‌شود. متغیرهایی که در قسمت تعاریف مدول یعنی بخش General تعریف شوند، در تمام روال‌های آن مدول قابل دسترس خواهند بود؛ به این‌گونه متغیرها، متغیرهای عمومی (Global Variable) می‌گویند. متغیرهای عمومی فقط در همان مدولی که اعلان شده‌اند، دیده می‌شوند. متغیرها را می‌توان به گونه‌ای اعلان کرد که در تمام مدول‌های پروژه قابل دسترس باشند که در ادامه، مورد بحث قرار خواهد گرفت.

به دلیل اینکه نام‌گذاری متغیرها به عهده‌ی برنامه‌نویس است، باید قواعد نام‌گذاری آنها را بدانید:

- نام متغیر باید با یکی از حروف الفبا شروع شود.
 - استفاده از حروف و اعداد در نام متغیرها مجاز است.
 - نام یک متغیر می‌تواند تا ۲۵۵ نویسه طول داشته باشد.
 - سعی کنید تا حد امکان از حروف خاص (غیرالفبایی-عددی) استفاده نکنید؛ زیرخط (_) در این میان یک استثناست. بدین ترتیب دیگر نیازی نیست نگران باشید که کدام حروف خاص مجازند و کدام حروف غیرمجاز.
 - فاصله در نام متغیرها مجاز نیست.
 - در نام‌گذاری متغیر، استفاده از کلمات کلیدی مجاز نیست.
- علاوه بر قواعد الزامی فوق، سعی کنید هنگام نام‌گذاری متغیرها نکات زیر را هم رعایت کنید:
- در نام متغیرها از پیشنوندهایی استفاده کنید که نوع آن را مشخص کند. بدین ترتیب دیگر نیازی نیست که مدام برای اطلاع از نوع یک متغیر به قسمت اعلان متغیرها مراجعه کنید. جدول ۲-۴ پیشنهاد انواع داده‌های متعارف ویژوال بیسیک را نشان می‌دهد.

جدول ۲-۴ پیشوند نام متغیرها

| مثال | نوع داده | پیشوند |
|--------------------|----------|--------|
| blnButtonEnabled | Boolean | bln |
| bytLenght | Byte | byt |
| curSales98 | Currency | cur |
| dteOverdue | Date | dte |
| dblScientificAmt | Double | dbl |
| intYear1998 | Integer | int |
| lngWeatherDistance | Long | lng |
| objWorksheetAcct99 | Object | obj |
| sngSales1stQte | Single | sng |
| strFirstName | String | str |
| vntValue | Variant | vnt |

- اگر از نام‌های بامعنی استفاده کنید، برنامه‌ی شما قابل فهم‌تر خواهد بود و به مستندسازی کمتری نیاز خواهد داشت.
- برای جدا کردن قسمت‌های نام متغیر از حروف بزرگ استفاده کنید، مانند curHighSales به مثال‌های زیر توجه کنید:

```
Dim intTotal As integer
Dim curSales99 As Currency
Dim dteFinal As Date
Dim strName As String
Dim blnLsChecked As Boolean
```

سعی کنید هنگام نام‌گذاری متغیرهای Boolean از سئوالاتی استفاده کنید که بتوان به آنها جواب بلی یا خیر داد (به آخرین خط مثال بالا توجه کنید).
در هر دستور Dim می‌توان بیش از یک متغیر را اعلان کرد و آنها را با کاما (,) از هم جدا کرد:

```
Dim intTotal As integer, CurSales99 As Currency
```

اگر نوع داده‌ی یک متغیر ذکر نشود، ویژوال بیسیک آن را از نوع Variant اعلان خواهد کرد؛ بنابراین هر دو دستور زیر معادل یکدیگرند:

Dim vntControlVal As Variant

Dim vntControlVal

۲-۳-۲ متغیرهای رشته‌ای

هنگام تعریف رشته‌ها باید نوع متغیر رشته‌ای را تعیین کرد، زیرا دو نوع داده‌ی String وجود دارد:

- با طول ثابت
- با طول متغیر

متداول‌ترین رشته‌ها، رشته‌های با طول متغیر هستند، زیرا روش تعریف آنها مانند تعریف سایر متغیرهاست. در مثال‌های زیر دو رشته با طول متغیر تعریف شده‌اند:

Dim strCityName As String

Dim strStateName As String

هر دو این متغیرها می‌توانند رشته‌هایی با طول‌های متفاوت را در خود نگه دارند. مثلاً اگر ابتدا در متغیر strCityName رشته‌ی "Tehran" و سپس رشته‌ی "Yazd" را ذخیره کنیم، این متغیر طول خود را متناسب با آن تغییر خواهد داد. در اکثر موارد این همان چیزی است که ما می‌خواهیم، ولی گاهی پیش می‌آید (مثلاً هنگام کار با پرونده‌ها) که بخواهیم طول رشته‌ها ثابت بماند. در این موارد باید طول آنها را مشخص کنیم:

Dim VarName As String* Length

که در آن Length به ویژوال بیسیک می‌گوید که طول این رشته ثابت است و هرگز نمی‌تواند از آن بزرگ‌تر شود. در مثال زیر، متغیری تعریف شده است که حداکثر می‌تواند ۵ نویسه بگیرد:

Dim strZipCode As String * 5

اگر رشته‌ای که طول آن بیش از پنج نویسه است، به متغیر strZipCode نسبت داده شود، ویژوال بیسیک فقط پنج نویسه‌ی اول آن را ذخیره می‌کند و مابقی را نادیده می‌گیرد.

۲-۳-۳ مقدار دادن به متغیرها

بعد از اعلان یک متغیر، می‌توان داده‌ها را در آن ذخیره کرد. ساده‌ترین راه برای ذخیره کردن یک مقدار در یک متغیر، دستور انتساب (assignment statement) است. شکل کلی

این دستور چنین است:

VarName = Expression

که در آن VarName نام یک متغیر (یا مشخصه) است و Expression می‌تواند یکی از موارد زیر باشد:

- عبارت محاسباتی
- مقدار
- عبارت منطقی یا رشته‌ای
- مقدار مشخصه‌ی یک کنترل (مشخصه‌ی کنترل‌ها از نوع Variant هستند ولی ویژوال بیسیک هنگام ذخیره کردن آنها در یک متغیر، نوع آنها را تبدیل خواهد کرد).
- ترکیبی از عبارت‌های محاسباتی یا منطقی، متغیرها و مقدار مشخصه‌ی کنترل‌ها

هر چیزی که بتواند یک مقدار تولید کند، عبارت (expression) است. به مثال‌های زیر توجه کنید:

curSales = 571275

strFirstName = "Ali"

blnPassedTest = **True**

blnLsEnabled = LblTitle.Enabled

intCount = intNumber

dteOld = #4/1/92#

sngOld97Total = sngNew98Total - 1000.00

مهم‌ترین نکته در باره‌ی یک عبارت آن است که مقدار سمت راست عبارت به متغیر سمت چپ آن نسبت داده می‌شود. توجه کنید که مقدار سمت راست عبارت باید با نوع متغیر سمت چپ عبارت متناسب باشد، یا اینکه امکان تبدیل آن برای ویژوال بیسیک وجود داشته باشد. مثلاً، ویژوال بیسیک می‌تواند یک عدد صحیح کوچک‌تر را در متغیری از نوع Long (که قاعدتاً برای اعداد بزرگ به‌کار می‌رود) قرار دهد، ولی نمی‌تواند یک رشته را به متغیر عددی نسبت دهد. در ویژوال بیسیک برای حفظ سازگاری با بیسیک‌های قدیمی، می‌توان از کلمه‌ی کلیدی Let برای مقدار دادن به متغیرها استفاده کرد؛ دو دستور زیر معادل یکدیگرند:

```
Let intCount = 1
```

```
intCount = 1
```

از دستور انتساب برای مقدار دادن به خواص کنترل‌ها نیز می‌توان استفاده کرد؛ شکل کلی انجام این کار به صورت زیر است:

مقدار = مشخصه.نام کنترل

به مثال زیر توجه کنید:

```
lblTitle.Caption = "The task is completed"
```

هر کنترل ویژوال بیسیک دارای یک مشخصه‌ی پیش فرض است که اگر مشخصه‌ای را ذکر نکنید، ویژوال بیسیک آن را در نظر خواهد گرفت. مشخصه‌ی پیش فرض برچسب، Caption است، یعنی به جای دستور فوق، می‌توانستید بنویسید:

```
lblTitle = "The task is completed"
```

با آنکه این روش ساده‌تر است، ولی از وضوح برنامه‌ها می‌کاهد. بنابراین سعی کنید نام مشخصه را ذکر کنید، حتی اگر مشخصه‌ی پیش فرض کنترل باشد.

نکته

در مقداردهی به متغیرهای منطقی می‌توان از مقدار صفر برای False و هر مقداری غیر از صفر برای True استفاده کرد.

نکته

متغیرهای منطقی در هنگام تعریف به طور خودکار، مقدار False، متغیرهای تاریخ و ساعت، مقدار 00:00:00، متغیرهای عددی، مقدار صفر و متغیرهای رشته‌ای به صورت یک رشته‌ی خالی ("") مقداردهی اولیه می‌شوند.

۲-۴ ثابت‌ها (Constants)

متغیرها تنها روش ذخیره‌ی اطلاعات در حافظه نیستند. روش دیگر، استفاده از ثابت‌هاست. ثابت‌ها مقادیری هستند که در برنامه‌ای که تعریف می‌شوند، مورد استفاده قرار می‌گیرند و

مقدار آنها در طول برنامه تغییر نمی‌کند. ثابت‌ها اغلب برای جایگزینی مقادیری که به‌خاطر سپردن آنها مشکل است یا برای پرهیز از نوشتن مکرر رشته‌های طولانی استفاده می‌شوند. همچنین با استفاده از ثابت‌ها می‌توان تغییرات مقدار را به سادگی، اعمال کرد. ثابت‌ها ممکن است از نوع عددی یا رشته‌ای باشند. ثابت‌های رشته‌ای، از تمام نویسه‌ها بجز نویسه‌ی نقل‌قول ("") تشکیل شده‌اند (اگر نیاز باشد در یک رشته نویسه (")) درج شود باید دوبار ذکر شود) و ثابت‌های عددی شامل کلبه‌ی اعداد مثبت و منفی هستند. "IRAN" و "987" از نوع ثابت‌های رشته‌ای و ۲۰۳۵۷۶۲۰۳ و ۵۶۹/۵۷ از ثابت‌های عددی هستند. ثابت‌های عددی نیز می‌توانند دارای یکی از انواع داده‌های عددی موجود در ویژوال بیسیک باشند. برای تعریف ثابت، از کلمه‌ی کلیدی Const استفاده می‌شود.

```
Const CONSTANT_NAME [As ConstantType] = value
```

CONSTANT_NAME نام ثابت، ConstantType نوع داده‌ی ثابت که اختیاری است و value مقداری است که در ثابت ذخیره می‌شود.

```
Const conPi As Single = 3.1415
```

```
Const conName As String = "Afshin"
```

```
Const conBirthDate As Date = #05/15/1968#
```

۲-۵ عملگرهای ویژوال بیسیک

ویژوال بیسیک از عملگرهای (Operators) محاسباتی، مقایسه‌ای، منطقی و رشته‌ای متعددی پشتیبانی می‌کند. جدول ۲-۵ عملگرهای محاسباتی و رشته‌ای متداول ویژوال بیسیک را نشان می‌دهد. عملگر ابزاری برای ترکیب داده‌های موردنظر است.

عملگر توان (exponentiation) یک عدد را به توان عدد دیگر می‌رساند؛ 2^3 یعنی ۲ به توان ۳، عملگرهای ضرب، تقسیم، جمع و تفریق هم دقیقاً همان کاری را می‌کنند که در محاسبات معمول با آن آشنا هستید. عملگر Mod (باقیمانده) برای محاسبه‌ی باقیمانده‌ی یک تقسیم است؛ بنابراین $11 \text{ Mod } 3$ معادل 2 خواهد بود. عملگر Mod فقط برای اعداد صحیح است و اگر از اعداد اعشاری استفاده کنید، ویژوال بیسیک ابتدا آنها را به عدد صحیح تبدیل کرده و سپس باقیمانده را محاسبه خواهد کرد. عملگر تقسیم صحیح (\) خارج قسمت صحیح تقسیم را برمی‌گرداند و از باقیمانده‌ی تقسیم صرف‌نظر می‌کند و مشابه Mod مخصوص اعداد صحیح است.

جدول ۲-۵ عملگرهای محاسباتی و رشته‌ای و ویژوال بیسیک

| عملگر | مفهوم | مثال | نتیجه |
|--------|---------------|---------------------|-------------|
| ^ | توان | $2^3 = 2^3$ | 8 |
| * | ضرب | $2*3$ | 6 |
| / | تقسیم | $6/2$ | 3 |
| + | جمع | $2+3$ | 5 |
| - | تفریق | $6-3$ | 3 |
| Mod | باقیمانده | $11 \text{ Mod } 3$ | 2 |
| \ | تقسیم صحیح | $11 \setminus 3$ | 3 |
| & یا + | ترکیب رشته‌ها | "Hi," & "There" | "Hi, There" |

$$\begin{array}{r} 11 \quad | \quad 3 \\ 9 \quad \textcircled{3} \\ \hline \textcircled{2} \end{array} \rightarrow \text{حاصل تقسیم صحیح (خارج قسمت)}$$

← باقیمانده

نکته‌ی جالب در جدول ۲-۵ آن است که عملگر + دو کار متفاوت انجام می‌دهد: جمع معمولی و به هم چسباندن (ترکیب) رشته‌ها. این عملگر با توجه به محلی که مورد استفاده قرار گرفته است (بین دو عدد یا بین دو رشته) واکنش مناسب را نشان می‌دهد. هنگام ترکیب رشته‌ها، ویژوال بیسیک هیچ چیز به آنها اضافه نخواهد کرد. بنابراین اگر می‌خواهید بین دو رشته یک فاصله وجود داشته باشد باید خودتان آن را اضافه کنید. به مثال زیر توجه کنید:

```
strCompleteName = lblFirst.Caption & " " & lblLast.Caption
```

عملگرهای منطقی و مقایسه‌ای را در ادامه‌ی کتاب مورد بررسی قرار خواهیم داد.

۲-۵-۱-۱ مقدم عملگرها

ویژوال بیسیک اعمال ریاضی را به ترتیب خاصی انجام می‌دهد. اعمال درون پرانتز ابتدا اجرا می‌شوند و سپس بالاترین تقدم را توان (^) دارد و بعد نوبت به ضرب و تقسیم می‌رسد و بعد از آن نوبت جمع و تفریق است (جدول ۲-۶).

اگر از پرانتزها استفاده نکنید، ویژوال بیسیک همیشه ابتدا توان، سپس ضرب و تقسیم و بعد از آن جمع و تفریق را انجام خواهد داد. اگر در یک عبارت، عملگرهایی با تقدم یکسان

جدول ۲-۶ ترتیب عملگرهای محاسباتی

| عملگر | ترتیب |
|-------------------|-------|
| \wedge | ۱ |
| $-$ (تفریق یکانی) | ۲ |
| $/, *$ | ۳ |
| \backslash | ۴ |
| Mod | ۵ |
| $-, +$ | ۶ |

وجود داشته باشند، ویزوال بیسیک محاسبات را از چپ به راست انجام می‌دهد. به مثال زیر توجه کنید:

$$10/2*3$$

در این عبارت به دلیل اینکه ضرب و تقسیم دارای تقدم یکسان هستند، ویزوال بیسیک ابتدا تقسیم را انجام داده و سپس حاصل تقسیم را در ۳ ضرب خواهد کرد؛ بدین ترتیب حاصل عبارت فوق ۱۵ خواهد شد. اگر می‌خواهید ترتیب انجام محاسبات را تغییر دهید باید از پرانتزها استفاده کنید. وجود پرانتزها اولویت عملگرها را تغییر می‌دهند. در پرانتزهای تودرتو، ویزوال بیسیک از داخلی‌ترین زوج پرانتز شروع کرده و رو به بیرون حرکت می‌کند. به عنوان مثال در عبارت

$$(10 + 2 - (8 - 3)) + 1$$

ویزوال بیسیک قبل از هر کاری (8-3) را محاسبه خواهد کرد. حاصل این عبارت برابر با ۸ خواهد بود.

نکته

هر پرانتز باز شده باید بسته شود.

۲-۶ دکمه‌ی فرمان (Command Button)

از این کنترل می‌توان برای شروع، توقف و پایان یک فرآیند استفاده کرد. به کمک این کنترل می‌توان به کاربر امکان داد عملیات مختلفی را که با مقداردهی مشخصه‌ها قابل اجرا نیست انجام دهد؛ مانند بستن یک فرم، بازکردن فرم دیگر، ذخیره یا صرف نظر کردن از نتایج ویرایش و ...

یکی از متداول‌ترین جاهایی که می‌توان کدی را برای عملیات خاصی قرار داد، رویداد Click دکمه‌ی فرمان است.

می‌توانید مشخصه‌ی Default دکمه‌ی فرمان را با True مقداردهی کنید تا دکمه‌ی پیش‌فرض شود. دکمه‌ی پیش‌فرض نسبت به سایر دکمه‌های فرمان حاشیه‌ی نازک سیاه رنگی دارد. اگر دکمه‌ای پیش‌فرض باشد، هنگامی که کاربر کلید Enter را فشار دهد، رویداد Click آن دکمه اجرا می‌شود.

مشخصه‌های دکمه‌ی فرمان که معمولاً در زمان طراحی تنظیم می‌شوند، در جدول ۲-۷ نشان داده شده است.

جدول ۲-۷

| مشخصه | توضیح |
|-----------------|---|
| Cancel | تعیین می‌کند که کد مربوط به رویداد Click دکمه‌ی فرمان، هنگام فشار دادن کلید ESC اجرا شود. |
| Caption | متنی است که روی دکمه نمایش داده می‌شود. با قرار دادن نویسه‌ی & قبل از یک نویسه در این مشخصه، می‌توان آن نویسه را به‌عنوان کلید میانبر تعریف کرد (به‌عنوان مثال Print &). در زمان اجرا، می‌توان با فشار دادن کلیدهای Alt+P دکمه‌ی فرمان را انتخاب کرد. |
| DisabledPicture | پرونده‌ی تصویری هنگام غیرفعال شدن دکمه نمایش داده می‌شود. |
| DownPicture | پرونده‌ی تصویری هنگام فشار دادن دکمه نمایش داده می‌شود. |
| Enabled | تعیین می‌کند آیا دکمه می‌تواند انتخاب شود. |
| Font | قلم متن روی دکمه‌ی فرمان |
| Picture | پرونده‌ی تصویری روی دکمه نمایش داده می‌شود. |

نکته

برای نمایش تصویر روی دکمه‌ی فرمان، باید مشخصه‌ی Graphical آن style یا ۲ تنظیم شود.

۲-۷ نمایش متن روی فرم و کادر تصویر

برای نمایش متن روی یک فرم یا کادر تصویر، از متد Print که بعد از نام فرم یا کادر تصویر قرار می‌گیرد، استفاده کنید.

شکل کلی متد Print به صورت زیر است:

```
[object.] Print [outputlist] [{; | ,}]
```

آرگومان Object اختیاری است و در صورتی که نوشته نشود، خروجی این متد روی فرم جاری نمایش داده می شود.

به عنوان مثال، عبارت های زیر پیامی را چاپ می کنند:

- روی فرمی با نام MyForm:

```
MyForm.Print "This is a form."
```

- روی کادر تصویری به نام picMiniMsg:

```
picMiniMsg.Print "This is a picture box."
```

- روی فرم جاری:

```
Print "This is the current form."
```

- روی شیء Printer به وسیله چاپگر:

```
Printer.Print "This text is going to the printer."
```

آرگومان outputlist متنی است که روی فرم یا کادر تصویر ظاهر می شود. اگر پس از دستور PRINT، هیچ عبارتی (مقدار ثابت یا متغیر) نوشته نشود، سبب خواهد شد که روی شیء جاری، یک خط خالی نشان داده شود. پس از متد PRINT می توان یک یا چند عبارت نوشت. برای جدا کردن این عبارت ها می توان از علامت کاما (,) یا سمی کولن (;) استفاده کرد.

تمرین

تفاوت (,) و (;) را در دو عبارت زیر بررسی کنید:

```
Print "computer", "book"
```

```
Print "computer"; "book"
```

۲-۸ متد cls

به کمک این متد می توان متن یا گرافیک موجود روی فرم یا کادر تصویر را در زمان اجرا پاک کرد. این متد تأثیری روی کنترل ها و تصاویر مربوط به مشخصه ی Picture فرم نخواهد

داشت. شکل کلی این متد به صورت زیر است:

[object].cls

آرگومان Object اختیاری بوده و شبیه متد Print عمل می‌کند.

۹-۲ توابع داخلی^۱

تابع را در حکم ماشین در نظر بگیرید که مقداری را به‌عنوان عامل (آرگومان) دریافت می‌کند و پس از انجام عملیاتی بر روی آن، مقدار دیگری را به‌عنوان خروجی تابع برمی‌گرداند.

توابع داخلی، در واقع تعدادی زیر برنامه‌ی آماده هستند که به همراه هر زبان برنامه‌نویسی ارائه می‌شوند تا وظایف و عملیات عمومی و رایج را انجام دهند. با فراخوانی یک تابع، می‌توان از تابع خواست که عمل مورد نظر را انجام دهد. اغلب توابع، مقدار یا مقادیری را به‌عنوان پارامتر (آرگومان) دریافت کرده، و پس از انجام عملیات بر روی آن پارامتر، مقدار جدیدی را به‌عنوان خروجی تابع برمی‌گردانند. تمام توابع دارای ویژگی‌های مشترکی به شرح زیر هستند:

- معمولاً برای آنکه تابع کار خود را انجام دهد، باید یک یا چند مقدار به آن داده شود (البته توابعی هم وجود دارند که ورودی نمی‌گیرند). به این مقادیر **آرگومان (argument)** گفته می‌شود.
- نام تابع همیشه دارای () است (حتی اگر هیچ آرگومانی نداشته باشد).
- آرگومان‌های تابع در داخل پرانتز قرار می‌گیرند و با کاما (,) از هم جدا می‌شوند.

برخی از توابعی که چند آرگومان می‌گیرند، اجازه می‌دهند که تعدادی از این آرگومان‌ها را قید نکنید. به این آرگومان‌ها، آرگومان‌های اختیاری (Optional) می‌گویند. نکته‌ی مهم در مورد آرگومان‌های تابع این است که نوع داده‌ی آرگومان‌ها باید با آنچه که تابع انتظار دارد، یکسان باشند. ترتیب ارسال آرگومان‌ها به تابع هم بسیار مهم است و باید آنها را به همان ترتیبی که تابع می‌خواهد، به آن ارسال کنید (البته در این مورد، استثنایی هم وجود دارد که بعدها به آن خواهیم پرداخت).

۱. بخشی از تابع‌های داخلی دیگر را در بخش‌های دیگر این کتاب مطالعه خواهید کرد.

۲-۹-۱ تابع SPC()

همانطور که می‌دانیم یک عبارت می‌تواند ترکیبی از ثابت‌ها، متغیرها و توابع باشد. تابع را در حکم ماشین در نظر بگیرید که مقداری را به‌عنوان عامل (آرگومان) دریافت می‌کند و پس از انجام عملیاتی بر روی آن، مقدار دیگری را به‌عنوان خروجی تابع برمی‌گرداند. توابع دارای انواع مختلفی هستند که در برنامه‌سازی ۲ درباره‌ی آنها توضیح خواهیم داد.

تابع SPC() در متد PRINT برای ایجاد فاصله بین خروجی‌ها مورد استفاده قرار می‌گیرد و به‌صورت کلی SPC(n) است که در آن n تعداد فضاهای خالی است که در خروجی متد PRINT باید در نظر گرفته شود. n می‌تواند یک عدد صحیح در بازه‌ی صفر تا ۳۲۷۶۷ باشد.

۲-۹-۲ تابع MsgBox()

MsgBox() تابعی است که یک کادر پیام به کاربر نشان می‌دهد (شکل ۲-۱).



شکل ۲-۱ تابع MsgBox() یک پیام نمایش داده و اجازه می‌دهد که بعد از خواندن پیام آن را ببندید.

همانطور که مشاهده می‌کنید، کادر پیام دارای یک نشانه، یک پیام و حداقل یک دکمه است؛ این دکمه به کاربر امکان می‌دهد تا بعد از خواندن پیام، کادر پیام را ببندد. با آنکه کاربر می‌تواند کادر پیام را جابه‌جا کند ولی نمی‌تواند هیچ تغییر دیگری در آن بدهد. نشانه کادر پیام، پیامی که نمایش می‌دهد و تعداد و نوع دکمه‌های آن آرگومان‌هایی هستند که تابع MsgBox() می‌گیرد. مشاهده می‌کنید که برنامه‌نویس بر آنچه کاربر مشاهده خواهد کرد، تسلط کامل دارد. مقدار برگشتی تابع MsgBox()، دکمه‌ای است که کاربر با کلیک کردن روی آن، کادر پیام را بسته است. برنامه می‌تواند با بررسی مقدار برگشتی این تابع (انتخاب کاربر)، عکس‌العمل مناسبی نشان دهد.

در ویرایش‌های قبلی ویژوال بیسیک دستوری به نام MsgBox وجود داشت که مقدار

برگشتی نداشت و برنامه نمی‌توانست تشخیص دهد که کدام دکمه کلیک شده است. این دستور (برای حفظ سازگاری با گذشته) هم‌چنان در Visual Basic 6.0 وجود دارد و در بعضی از مثال‌ها از این دستور استفاده کرده‌ایم.

شکل کلی تابع `MsgBox()` چنین است:

```
intResponse = MsgBox (strPrompt [, intStyle] [, strTitle])
```

تابع `MsgBox()` یک آرگومان اجباری (`strPrompt`) و دو آرگومان اختیاری (`intStyle`) و (`strTitle`) دارد. از پیشوند نام این آرگومان‌ها می‌توانید حدس بزنید که نوع آنها چیست. آرگومان اول (`strPrompt`)، پیامی است که باید کادر پیام نشان دهد. آرگومان دوم (`intStyle`)، تعداد و نوع دکمه‌های کادر پیام را تعیین خواهد کرد. آرگومان سوم (`strTitle`)، عنوان پنجره‌ی کادر پیام خواهد بود. مقدار برگشتی تابع، در متغیر صحیح `intResponse` قرار خواهد گرفت. وقتی کادر پیام ظاهر شود، اجرای برنامه موقتاً متوقف می‌شود تا کاربر دکمه‌ای از کادر پیام را کلیک کند. بعد از آن، اجرای برنامه از دستور بعد از تابع `MsgBox()` از سر گرفته خواهد شد (اگر جمله‌ای که نمایش می‌دهید بسیار طولانی باشد، خود ویژوال بیسیک آن را به چند خط تقسیم خواهد کرد).

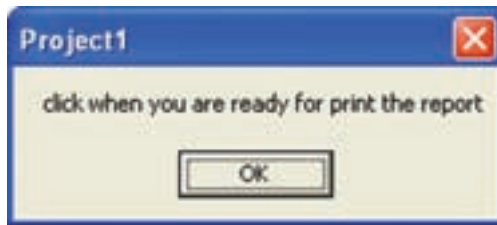
فرض کنید در برنامه‌ای قبیل از چاپ گزارش، کاربر باید آمادگی خود را اعلام کند. دستور ساده‌ی زیر این کار را انجام می‌دهد:

```
intResponse = MsgBox ("click when you are ready for print the report")
```

البته باید قبلاً متغیر `intResponse` را در قسمت تعاریف برنامه اعلان کرده باشید. اگر آرگومان دوم را قید نکرده باشید، ویژوال بیسیک کادر پیام را فقط با یک دکمه‌ی OK نمایش خواهد داد. شکل ۲-۲ این کادر پیام ساده را نشان می‌دهد. در این حالت دیگر مقدار برگشتی تابع اهمیت چندانی ندارد و می‌توان از روش دستوری این تابع استفاده کرد که نیازی به مقدار برگشتی ندارد. به‌عنوان مثال، تابع فوق را به‌صورت زیر نیز می‌توان نوشت:

```
MsgBox "click when you are ready for print the report"
```

توجه کنید که اگر آرگومان اختیاری دوم قید نشود، ویژوال بیسیک از نام پروژه در عنوان پنجره‌ی پیام استفاده می‌کند. این شاید چندان جالب نباشد و در آینده خواهید دید که چگونه می‌توان آن را اصلاح کرد.



شکل ۲-۲ تابع MsgBox() در حالت پیش فرض پیامی را با یک دکمه نمایش می دهد.

نکته

در صورتی که بخواهید پیامی در بیش از یک سطر نمایش پیدا کند لازم است از ثابت VBCrLf یا VBNewline به شکل زیر استفاده کنید.

MsgBox "click when you are ready" + VBNewline + "for print the report"

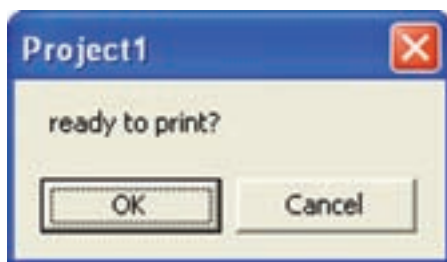
برای کنترل بیشتر دکمه های کادر پیام باید از آرگومان اختیاری اول (intStyle)، استفاده کنید. وقتی دکمه های کادر پیام بیش از یکی باشد، مقدار برگشتی MsgBox() مفهوم پیدا می کند. جدول ۲-۸ مقادیر ممکن آرگومان دوم تابع MsgBox() را نشان می دهد.

جدول ۲-۸ برای تعیین تعداد و نوع دکمه های کادر پیام از اعداد صحیح استفاده کنید.

| مقدار | نام ثابت | توضیح |
|-------|--------------------|-------------------------------|
| ۰ | VbOKOnly | دکمه ی OK |
| ۱ | VbOKCancel | دکمه های OK و Cancel |
| ۲ | VbAbortRetryIgnore | دکمه های Abort، Retry، Ignore |
| ۳ | VbYesNoCancel | دکمه های Yes، No، Cancel |
| ۴ | VbYesNo | دکمه های Yes، No |
| ۵ | VbRetryCancel | دکمه های Retry و Cancel |

شکل ۲-۳ کادر پیام حاصل از دستور زیر را نشان می‌دهد:

```
intResponse = MsgBox ("Ready to print?", 1)
```



شکل ۲-۳ دکمه‌ای که کاربر کلیک کرده است، عکس‌العمل برنامه را تعیین می‌کند.

عدد ۱ در آرگومان دوم تعیین می‌کند که این کادر پیام دارای دو دکمه‌ی OK و Cancel خواهد بود؛ این کادر برای حالت‌هایی مناسب است که کاربر باید امکان لغو عملیات را هم داشته باشد. جدول ۲-۹ مقادیر برگشتی کادر پیام را نشان می‌دهد.

جدول ۲-۹ با بررسی مقدار برگشتی می‌توان متوجه شد که کاربر روی کدام دکمه کلیک کرده است.

| مقدار | نام ثابت | توضیح |
|-------|----------|--------------------------------|
| ۱ | vbOK | کاربر OK را کلیک کرده است. |
| ۲ | vbCancel | کاربر Cancel را کلیک کرده است. |
| ۳ | vbAbort | کاربر Abort را کلیک کرده است. |
| ۴ | vbRetry | کاربر Retry را کلیک کرده است. |
| ۵ | vbIgnore | کاربر Ignore را کلیک کرده است. |
| ۶ | vbYes | کاربر Yes را کلیک کرده است. |
| ۷ | vbNo | کاربر No را کلیک کرده است. |

در کادر پیام، زدن کلید Esc معادل کلیک کردن روی دکمه‌ی Cancel است. توجه داشته باشید که در هر حال کاربر فقط می‌تواند یکی از دکمه‌ها را کلیک کند، چون به محض کلیک کردن اولین دکمه، کادر پیام ناپدید شده و مقدار مناسب برگشت داده می‌شود.

استفاده از ثابت‌های نام‌دار

اگر به جدول ۲-۸ و ۲-۹ دقت کرده باشید ستونی تحت عنوان «نام ثابت» در آنها مشاهده می‌کنید. ویژوال بیسیک صدها ثابت (constant) دارد که آنها را به نام می‌شناسد؛ به اینها ثابت نام‌دار (named constant) می‌گویند. مقدار این ثابت‌ها همیشه ثابت است (و غیر از این هم نمی‌تواند باشد!) و نمی‌توان آنها را تغییر داد. استفاده از ثابت‌های نام‌دار، وضوح برنامه را افزایش خواهد داد. به مثال‌های زیر توجه کنید (هر دوی آنها در واقع یکی هستند):

```
intResponse = MsgBox ("Ready to print?", 1)
```

```
intResponse = MsgBox ("Ready to print?", vbOKCancel)
```

وقتی از ثابت‌های نام‌دار استفاده می‌کنید، دیگر نیازی نیست مقدار آنها را حفظ کنید و دفعه‌ی بعد که آنها را ببینید، درک عملکرد آنها ساده‌تر خواهد بود. ویژوال بیسیک هنگام نوشتن توابع، لیستی از ثابت‌های نام‌دار را نشان می‌دهد و دیگر حتی زحمت نوشتن آنها را هم نخواهید داشت. بنابراین سعی کنید تا حد امکان از ثابت‌های نام‌دار استفاده کنید.

دکمه‌های پیش‌فرض

همیشه اولین دکمه‌ی کادر پیام (از سمت چپ) دکمه‌ی پیش‌فرض است، یعنی زدن Enter معادل کلیک کردن آن دکمه خواهد بود. اما با استفاده از مقادیر جدول ۲-۱۰ می‌توانید این رفتار را عوض کنید.

جدول ۲-۱۰ مقادیری که با آنها می‌توانید دکمه‌ی پیش‌فرض کادر پیام را تغییر دهید.

| مقدار | نام ثابت | توضیح |
|-------|------------------|--------------------------------|
| ۰ | VbDefaultButton1 | دکمه‌ی اول دکمه‌ی پیش‌فرض است. |
| ۲۵۶ | VbDefaultButton2 | دکمه‌ی دوم دکمه‌ی پیش‌فرض است. |
| ۵۱۲ | VbDefaultButton3 | دکمه‌ی سوم دکمه‌ی پیش‌فرض است. |

ثابت‌های نام‌دار، علیرغم طولانی بودن آنها، درک و نگهداری برنامه را ساده‌تر می‌کند و برنامه را خودمستند خواهد کرد. به دستور زیر نگاه کنید:

```
intResponse = MsgBox ("Is the printer on?", vbYesNoCancel +  
vbDefaultButton1)
```

در این مثال، دکمه‌ی Yes به‌عنوان دکمه‌ی پیش‌فرض در نظر گرفته شده است. در اعمال بحرانی، مانند حذف فایل‌ها، دکمه‌ی Cancel را دکمه‌ی پیش‌فرض کادر پیام قرار دهید تا اگر کاربر تصادفاً کلید Enter را فشار داد، عملیات لغو شود.

تعیین نشانه‌ی کادر پیام

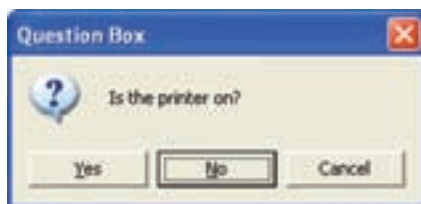
با استفاده از آرگومان دوم تابع `MsgBox()` می‌توانید نشانه‌ی کادر پیام را هم تغییر دهید. تا اینجا کادر پیام‌هایی که ایجاد کردیم هیچ‌یک نشانه‌ای نداشتند. در جدول ۲-۱۱ نشانه‌های کادر پیام را مشاهده می‌کنید.

جدول ۲-۱۱ با این مقادیر می‌توانید نشانه‌ی کادر پیام را تغییر دهید.

| مقدار | نام ثابت | توضیح | نشانه‌ی |
|-------|---------------|---------------------|--|
| ۱۶ | VbCritical | نشانه‌ی پیام بحرانی |  |
| ۳۲ | VbQuestion | نشانه‌ی علامت سؤال |  |
| ۴۸ | VbExclamation | نشانه‌ی اخطار |  |
| ۶۴ | vbInformation | نشانه‌ی اطلاعات |  |

دستور زیر، کادر پیامی با سه دکمه‌ی Yes، No، Cancel و نشانه‌ی علامت سؤال را ایجاد می‌کند. شکل ۲-۴ این کادر پیام را نشان می‌دهد.

```
intResponse = MsgBox("Is the printer on?", vbYesNoCancel +  
vbQuestion + VbDefaultButton2, "Question Box")
```



شکل ۲-۴ انتخاب کاربر، مسیر برنامه را تعیین خواهد کرد.

۲-۹-۳ تابع InputBox()

در بعضی موارد برنامه باید پرسشی را برای کاربر مطرح کند و پاسخ آن را بگیرد، ولی برنامه‌نویس برای این منظور قرار دادن یک کادر متن روی فرم برنامه را مناسب نمی‌بیند. آیا راه دیگری برای ارتباط با کاربر وجود دارد؟ بله، تابع InputBox(). این تابع، هم پیامی به کاربر ارائه می‌دهد و هم امکان می‌دهد تا کاربر جواب خود را وارد کند. در شکل ۲-۵ یک کادر ورودی (Input box) را مشاهده می‌کنید.



شکل ۲-۵ در کادر ورودی، فیلدی برای دریافت پاسخ کاربر وجود دارد.

کادر ورودی تمام خصوصیات کادر پیام را دارد و فقط دارای یک فیلد اضافی برای گرفتن جواب کاربر است. البته برنامه‌نویس هیچ کنترلی روی دکمه‌های کادر ورودی ندارد و یک کادر ورودی همیشه دارای دو دکمه‌ی OK و Cancel خواهد بود. کادر ورودی، نشانه هم نمی‌تواند داشته باشد. شکل کلی تابع InputBox() چنین است:

```
strAnswer = InputBox(strPrompt[, strTitle] [, strDefault] [, intXpos]
[, intYpos])
```

مقدار برگشتی تابع InputBox() از نوع Variant است و همیشه می‌توان آن را به صورت یک رشته به کار برد. به این دلیل که مقدار برگشتی این تابع Variant است آن را حتی به مشخصه‌های کنترل‌ها هم می‌توان نسبت داد. از میان تمام آرگومان‌های تابع InputBox() فقط اولین آرگومان اجباری است.

- strPrompt: پیام یا پرسشی است که در کادر ورودی مشاهده می‌شود. حداکثر طول

این پیام، می‌تواند ۱۰۲۴ نویسه باشد. سعی کنید که تا حد امکان این پیام جنبه‌ی پرسشی داشته باشد.

- `strTitle`: عنوان پنجره‌ی کادر ورودی. اگر این آرگومان قید نشود، ویژوال بیسیک به‌جای آن از نام پروژه استفاده خواهد کرد.
- `strDefault`: مقداری که به‌صورت پیش‌فرض در فیلد ورودی ظاهر خواهد شد. کاربر می‌تواند این مقدار را قبول کند یا اینکه آن را تغییر دهد. استفاده از این مقدار پیش‌فرض می‌تواند زحمت تایپ کاربر را کمتر کند.
- `intXpos` و `intYpos`: مختصات ظاهر شدن پنجره‌ی کادر ورودی روی صفحه. اگر این آرگومان‌ها قید نشوند، ویژوال بیسیک کادر ورودی را وسط صفحه‌ی نمایش قرار خواهد داد.

کادر پیامی که در شکل ۵-۲ مشاهده می‌کنید، حاصل دستور زیر است:

```
strAnswer = InputBox("What is the customer's name?", "Get name")
```

اگر می‌خواهید این کادر ورودی مقدار پیش‌فرض داشته باشد و در محل خاصی ظاهر شود، از دستور زیر استفاده کنید:

```
strAnswer = InputBox("What is the customer's name?", "Get name",  
"Ali", 500, 750)
```

اما باید روشی وجود داشته باشد تا برنامه بدانند که کاربر کدام دکمه را کلیک کرده است. اگر کاربر روی `Cancel` کلیک کند، تابع `InputBox()` رشته‌ای به طول صفر ("") برمی‌گرداند و کلیک کردن روی `OK` سبب برگشت رشته‌ی داخل فیلد ورودی خواهد شد.

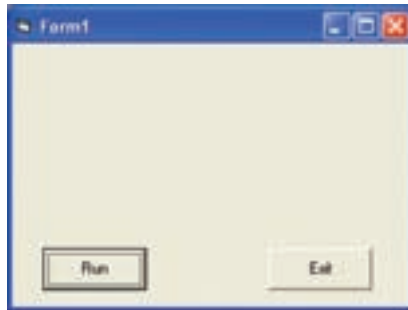
مثال ۲-۱

برنامه‌ی زیر دو عدد را از ورودی دریافت و سپس حاصل‌ضرب آنها را با کلیک روی دکمه‌ی `Run` محاسبه و چاپ می‌کند.

برنامه‌ی ویژوال بیسیک را اجرا کنید و از کادر محاوره‌ای `New Project` گزینه‌ی `Standard EXE` را انتخاب کرده و `OK` را فشار دهید تا وارد محیط اصلی ویژوال

بیسیک شوید. در پنجره‌ی مشخصه‌ها مراحل زیر را انجام دهید:

۱. مشخصه‌ی Name فرم را به frmExample1 تغییر دهید.
 ۲. مشخصه‌ی Caption فرم را به First Example تغییر دهید.
 ۳. از جعبه‌ابزار، دو شیء CommandButton را به فرم اضافه کرده و مشخصه‌ی Name آنها را به ترتیب cmdRun و cmdExit قرار داده و مشخصه‌ی Caption آنها را نیز به ترتیب Run و Exit قرار دهید.
- فرم شبیه شکل ۲-۶ خواهد بود.



شکل ۲-۶

۴. حال لازم است کد برنامه را اضافه کنیم. برای این کار روی دکمه‌ی Run دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub cmdRun _Click()
```

و

```
End Sub
```

دستورهایی زیر را وارد کنید:

```
intNum1 = InputBox("Enter First Number", "Get Number 1")
intNum2 = InputBox("Enter Second Number", "Get Number 2")
intResult = intNum1 * intNum2
MsgBox intNum1 & "*" & intNum2 & "=" & intResult, ,
"Result"
```

۵. در پنجره‌ی پروژه روی دکمه‌ی View Object کلیک کنید تا فرم دوباره نمایش یابد. این بار روی دکمه‌ی Exit دوبار کلیک کنید و بین دو دستور:

```
Private Sub cmdExit_Click()
```

و

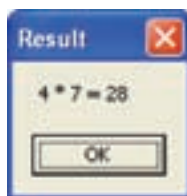
```
End Sub
```

دستور End را وارد کنید (دستور End در ویژوال بیسیک برای خاتمه‌ی اجرای برنامه به کار می‌رود).

۶. با فشار دادن کلید F5 برنامه را اجرا کنید.

دکمه‌ی Run را کلیک کنید و دو عدد دلخواه (۴ و ۷) را به ترتیب در پاسخ کادرهای محاوره‌ای وارد کنید.

کادری به شکل ۲-۷ نمایان شده و نتیجه را نشان می‌دهد.



شکل ۲-۷

پروژه‌ی خود را ذخیره کنید. برای این کار ابتدا برنامه را متوقف کنید (روی دکمه‌ی Stop در نوار ابزار کلیک کنید) و از منوی File گزینه‌ی Save Project As... را انتخاب کنید. سپس در محل موردنظر، پوشه‌ای به نام Exam1 ایجاد کرده و روی دکمه‌ی Save کلیک کنید. باید توجه داشت که ابتدا فرم با نام frmExample.frm و سپس با کلیک مجدد روی دکمه‌ی Save، پروژه با نام پیش فرض Project1.vbp ذخیره می‌شود.

مثال ۲-۲

برنامه‌ی زیر شعاع یک دایره را از ورودی دریافت و سپس مساحت آن را محاسبه و چاپ می‌کند.

از منوی File گزینه‌ی New Project را انتخاب کنید و از کادر محاوره‌ای New Project

گزینه‌ی Standard EXE را انتخاب کرده و OK را فشار دهید تا وارد محیط اصلی ویژوال بیسیک شوید. در پنجره‌ی مشخصه‌ها مراحل زیر را انجام دهید:

۱. مشخصه‌ی Name فرم را به frmExample2 تغییر دهید.
۲. مشخصه‌ی Caption فرم را به Second Example تغییر دهید.
۳. از جعبه ابزار، دو شیء CommandButton را به فرم اضافه کرده و مشخصه‌ی Name آنها را به ترتیب cmdRun و cmdExit قرار داده و مشخصه‌ی Caption آنها را نیز به ترتیب Exit و Run قرار دهید.
۴. حال لازم است کد برنامه را اضافه کنیم. برای انجام این عمل، روی دکمه‌ی Run دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub cmdRun_Click()
```

و

```
End Sub
```

دستورهای زیر را وارد کنید:

```
Const PI As Single = 3.141592
```

```
Dim sngRadius As Single, sngArea As Single
```

```
sngRadius = InputBox("Enter radius:", "Get Radius")
```

```
sngArea = PI * sngRadius ^ 2
```

```
MsgBox "Area of Circle with radius =" & sngRadius & _  
"is:" & sngArea, , "Result"
```

۵. در پنجره‌ی پروژه روی دکمه‌ی View Object کلیک کنید تا فرم مجدداً نمایش یابد. این بار روی دکمه‌ی Exit کلیک کنید و بین دو دستور:

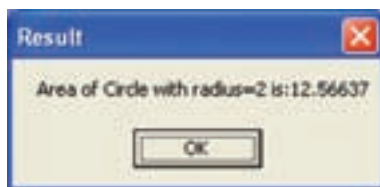
```
Private Sub cmdExit_Click()
```

و

```
End Sub
```

دستور End را وارد کنید.

با فشار دکمه‌ی F5 برنامه را اجرا کنید. روی دکمه‌ی Run کلیک کنید و در پاسخ کادر محاوره‌ای، شعاع موردنظر را وارد کنید (به‌عنوان مثال، عدد ۲). کادری به شکل ۲-۸ ظاهر شده و نتیجه را نشان می‌دهد.



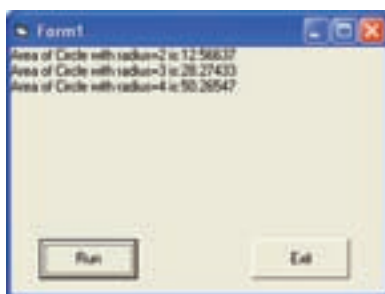
شکل ۲-۸

اگر بخواهیم به جای MsgBox از متد Print استفاده کنیم، کافی است دستور MsgBox را به صورت زیر تغییر دهیم:

```
Print "Area of Circle with radius = " & sngRadius &_
```

```
"is:" & sngArea
```

در این صورت خروجی پس از سه بار اجرا با شعاع‌های ۲، ۳ و ۴ به صورت شکل ۲-۹ خواهد بود.



شکل ۲-۹

برای آنکه در هر بار عمل چاپ، صفحه‌ی فرم پاک شده و فقط نتیجه در یک سطر نمایش یابد، می‌توان از متد cls قبل از متد Print استفاده کرد.

تمرین

گام به‌کارگیری برنامه در مثال‌های ۱-۱ و ۱-۲ فصل اول را انجام دهید.

نکته

اگر کاربر به جای ورود عدد، رشته‌ای وارد کند یا دکمه‌ی Cancel را بزند، خطایی تولید شده و پنجره‌ای به صورت شکل ۱۰-۲ نمایش می‌یابد که در درس برنامه‌سازی ۲ نحوه‌ی رفع این گونه خطاها را خواهید آموخت.



شکل ۱۰-۲

خلاصه‌ی فصل

داده‌ها به دو دسته‌ی عددی و غیرعددی تقسیم می‌شوند. داده‌های عددی را نیز به دو گروه صحیح و اعشاری تقسیم می‌کنند. داده‌های غیرعددی می‌توانند شامل داده‌های منطقی، رشته‌ای، تاریخ و زمان و شیء باشند.

متغیر، مکانی از حافظه است که می‌توان یک نوع داده را در آن ذخیره کرد. طبیعی است که هر متغیر باید دارای نامی باشد تا بتوان به آن رجوع کرد. نوع متغیر به نوع داده‌ای که قرار است در آن ذخیره شود بستگی دارد. برای اعلان متغیرها از شکل کلی زیر استفاده کنید:

Dim VarName As DataType

Dim نوع داده As نام متغیر

در نام‌گذاری متغیرها باید از قواعد خاصی پیروی کرد.

برای مقداردهی متغیرها یا مشخصه‌ی کنترل‌ها، آنها را در سمت چپ یک عبارت نوشته و بعد علامت مساوی قرار دهید. سپس در سمت راست علامت مساوی، یک مقدار یا عبارت محاسباتی را بنویسید.

ثابت‌ها مقادیری هستند که در برنامه‌ها تعریف می‌شوند و مورد استفاده قرار می‌گیرند و مقدار آنها در طول برنامه تغییر نمی‌کند. ثابت‌ها اغلب برای جایگزینی مقادیری که به خاطر سپردن آنها سخت است یا برای پرهیز از نوشتن مکرر رشته‌های طولانی استفاده می‌شوند.

برای نوشتن عبارت‌ها و ترکیب داده‌ها، به مفهومی به نام عملگر نیاز داریم. عملگرها دارای انواع مختلفی مثل محاسباتی، رشته‌ای، منطقی و رابطه‌ای هستند. از عملگرهای محاسباتی مثل توان و چهار عمل اصلی، برای نوشتن عبارت‌های محاسباتی استفاده می‌شود.

دکمه‌ی فرمان، یکی از کنترل‌های ویژال بیسیک است که معمولاً با رویداد کلیک فعال شده و رفتاری را که برنامه‌نویس تعیین کرده است، انجام می‌دهد.

برای نمایش داده‌ها روی فرم یا کادر تصویر، از متد Print استفاده می‌شود. در این متد، از توابع SPC و TAB برای فاصله‌گذاری خروجی‌ها استفاده می‌شود.

از متد cls برای پاک کردن فرم یا کادر تصویر استفاده می‌شود.

از تابع یا دستور MsgBox برای نمایش یک کادر پیام استفاده می‌شود. در صورتی که شکل تابع آن را به کار ببرید، می‌توانید نوع دکمه‌ای را که کاربر فشار داده است، تشخیص دهید.

برای دریافت داده‌ها از کاربر، از تابع InputBox استفاده کنید.

خودآزمایی

۱. عبارت‌های زیر را از فرم ریاضی به فرم معادل و یژوال بیسیک تبدیل کنید.

$$\text{الف) } a^b + 4b^2c \quad \text{ج) } \sqrt{b^2 - 4ac}$$

$$\text{ب) } -3^4 + x^3 - \frac{4z^2}{x+y} \quad \text{د) } \frac{x + v^z}{3^5 y^2}$$

۲. کدام نام زیر برای یک متغیر غیرمجاز است. با ذکر علت، آن را مشخص کنید.

$$\text{الف) } 3AB \quad \text{ج) } A3*2 \quad \text{ب) } \text{Ali Reza} \quad \text{د) } \text{Block 23}$$

۳. حاصل عبارت‌های زیر را به دست آورید.

$$\text{الف) } 3^2 * 5 + 6/3 - 2 \quad \text{ب) } 41 \text{ Mod } 2 * 3 + (81 \setminus 4 * 2)$$

$$\text{ج) } 36 + 8 \text{ Mod } 3 + 42/7 \setminus 2 \quad \text{د) } 26 + 2^4/21 \setminus 3 * 8$$

۴. برنامه‌ای بنویسید که طول و عرض یک مستطیل را خوانده و محیط و مساحت آن را نشان دهد.

۵. برنامه‌ای بنویسید که ساعت، دقیقه و ثانیه را خوانده و زمان معادل را فقط برحسب ثانیه حساب کند.

۶. برنامه‌ای بنویسید که یک عدد ۳ رقمی را بخواند و مقلوب آن را چاپ کند.

۷. برنامه‌ای بنویسید که ۵ عدد از ورودی بخواند و میانگین آنها را چاپ کند.

۸. برنامه‌ای بنویسید که درجه‌ی هوا را برحسب سانتی‌گراد خوانده و معادل آن را به فارنهایت نمایش دهد.

$$(-32 - \text{فارنهایت}) = \frac{5}{9} \text{ سانتی‌گراد}$$

۹. برنامه‌ای بنویسید که یک عدد ۲ رقمی را بخواند و مجموع ارقام آن را چاپ کند.

۱۰. برنامه‌ای بنویسید که شعاع قاعده و ارتفاع یک استوانه را دریافت و مساحت جانبی، مساحت کل و حجم استوانه را محاسبه و چاپ کند.

توضیح: برای برنامه‌های ۴ تا ۱۰، گام‌های طراحی را به ترتیب انجام دهید (بر اساس مطالب فصل اول).

فصل سوم

کنترل برنامه

در این فصل با چند **عملگر** دیگر ویروال بیسیک آشنا خواهید شد که محاسباتی نیستند. عملگرهایی که در این فصل مشاهده خواهید کرد، **عملگرهای رابطه‌ای یا مقایسه‌ای و منطقی** هستند. به کمک این عملگرها می‌توانید در یک برنامه، شرایطی را بررسی کرده یا کارهای تکراری انجام دهید.

پس از پایان این فصل، انتظار می‌رود که فراگیر بتواند:

- عملگرهای مقایسه‌ای را توضیح دهد.
- عملگرهای منطقی را شرح دهد.
- با دستور If برنامه بنویسد.
- از توابع If(), Choose() و Switch() در برنامه‌های خود استفاده کند.
- از کنترل‌های Option Button، Check Box، Text Box، Label و Frame در برنامه‌های خود استفاده کند.

۳-۱ عملگرهای رابطه‌ای یا مقایسه‌ای

فرض کنید از شما خواسته‌اند که یک برنامه‌ی حسابداری بنویسید. این برنامه باید مطالبات هر شرکت را جمع کرده و یک چک در وجه هر کدام چاپ کند. اما اگر طی یک مدت خاص با یکی از شرکت‌ها هیچ معامله‌ای صورت نگرفته باشد، چه باید کرد؟ آیا برنامه باید یک چک به مبلغ صفر چاپ کند؟ یقیناً خیر. تا اینجا در برنامه‌هایی که مشاهده کردید، دستورها یکی بعد از دیگری اجرا می‌شدند. با عملگرهای رابطه‌ای که در این فصل با آنها آشنا خواهید شد،

می‌توانید ترتیب اجرای دستورهای برنامه را مطابق داده‌های ورودی برنامه تغییر دهید. بدین ترتیب برنامه‌ی حسابداری شما خواهد توانست فقط برای شرکت‌هایی که مطالبات واقعی دارند، چک چاپ کند.

در جدول ۱-۳ عملگرهای رابطه‌ای و بیژوال بیسیک را مشاهده می‌کنید. این عملگرها هیچگونه عملیات ریاضی انجام نمی‌دهند، بلکه داده‌ها را مقایسه می‌کنند. با این عملگرها، برنامه هوشمندتر خواهد شد و خواهد توانست داده‌ها را مقایسه کرده و بر اساس نتایج آن، عملکرد مناسب را در پیش گیرد.

جدول ۱-۳ عملگرهای رابطه‌ای و بیژوال بیسیک

| عملگر | توضیح | مثال | نتیجه |
|-------|------------------|------------|--------|
| > | بزرگ‌تر از | $6 > 3$ | درست |
| < | کوچک‌تر از | $5 < 11$ | درست |
| >= | بزرگ‌تر یا مساوی | $23 >= 23$ | درست |
| <= | کوچک‌تر یا مساوی | $4 <= 21$ | درست |
| = | تساوی | $7 = 2$ | نادرست |
| <> | نامساوی | $3 < > 3$ | نادرست |

توجه کنید که نتیجه‌ی عمل عملگرهای رابطه‌ای همیشه False (نادرست) یا True (درست) خواهد بود. از فصل قبل به یاد دارید که اینها مقادیر Boolean هستند. نکته‌ی دیگر جدول ۱-۳ آن است که عملگر = هم می‌تواند در دستور انتساب مورد استفاده قرار گیرد، و هم در عبارت‌های مقایسه و این و بیژوال بیسیک است که کارکرد آن را تشخیص داده، به طور مناسب عمل خواهد کرد.

عملگرهای رابطه‌ای روی عبارت‌ها، متغیرها، مشخصه‌ها، کنترل‌ها یا ترکیبی از آنها عمل می‌کنند. با توجه به تنوع داده‌ها در و بیژوال بیسیک، این برنامه‌نویس است که باید تصمیم بگیرد داده‌ها را چگونه مقایسه کرده و چگونه نتیجه‌گیری کند.

عملگرهای رابطه‌ای، علاوه بر مقادیر عددی، می‌توانند رشته‌ها را هم مقایسه کنند. در هنگام مقایسه‌ی رشته‌ها به نکات زیر توجه داشته باشید:

- حروف بزرگ، کوچک‌تر از حروف کوچک هستند، یعنی "IRAN" قبل از "iran" قرار خواهد گرفت.

- حروف الفبا به همان ترتیبی که هستند مقایسه می‌شوند، یعنی "A" کوچک‌تر از "B" است و نام "Ahmad" قبل از "Ali" قرار خواهد گرفت.
- رقم‌ها کوچک‌تر از حروف هستند، یعنی "3" از "Three" کوچک‌تر خواهد بود.

برای درک بهتر این نکات، فقط کافی است بدانید که ویژوال بیسیک رشته‌ها را مانند یک لغت‌نامه مرتب خواهد کرد. با این قابلیت، برنامه‌ی شما می‌تواند روی انواع مختلف داده‌ها عمل کند.

ویژوال بیسیک هنگام مقایسه‌ی رشته‌ها به بزرگ یا کوچک بودن حروف هم توجه دارد یعنی بین آنها تفاوت قابل خواهد شد.^۱ اگر می‌خواهید ویژوال بیسیک حروف بزرگ و کوچک را یکسان فرض کند، می‌توانید در قسمت تعاریف پنجره‌ی کد از دستور Option Compare Text استفاده کنید. اما اگر این دستور وجود نداشته باشد، ویژوال بیسیک برای مرتب کردن رشته‌ها از جدول ASCII استفاده خواهد کرد. به عبارت‌های مقایسه‌ای زیر توجه کنید:

"abcdef" > "ABCDEF"

"Yes!" < "Yes?"

"Computers are fun!" = "Computers are fun!"

"PC" < > "pc"

"Books, Books, Books" > = "Books, Books"

ارزش تمام عبارت‌های فوق مقدار True را بر می‌گردانند.

ویژوال بیسیک عملگر رابطه‌ای دیگری به نام Like دارد که با آن می‌توان مقایسه‌های کلی‌تری انجام داد که در ادامه‌ی درس برنامه‌سازی با آن بیشتر آشنا خواهید شد. هنگام مقایسه‌ی دو مقدار، باید نوع آنها با هم سازگار باشد. به‌عنوان مثال، دو عدد یا دو رشته را می‌توان با هم مقایسه کرد، ولی نمی‌توان یک عدد را با یک رشته یا یک مقدار منطقی (Boolean) مقایسه کرد.

ویژوال بیسیک هنگام مقایسه‌ی داده‌های Variant (که مشخصه‌های کنترل‌ها هم از این نوع است) بسیار خوب عمل می‌کند. ابتدا آنها را به نوع مناسب تبدیل کرده و سپس مقایسه را انجام خواهد داد. اما مقایسه‌ی داده‌های ناهمگن همیشه می‌تواند با خطا همراه باشد.^۱ این ویژگی را حساس به حرف یا Case Sensitive گویند.

۳-۲ عملگرهای منطقی

عملگرهای منطقی (logical operator) که در ویژوال بیسیک وجود دارند به ترتیب تقدم عبارت اند از:

۱. Not، ۲. And، ۳. Or، ۴. Xor، ۵. Eqv

عملگرهای منطقی عبارت‌های موردنظر را بیت به بیت با یکدیگر مقایسه کرده، دو ارزش منطقی T(TRUE) یا F(FALSE) را برای آنها مشخص می‌کنند. اگر P و Q دو عبارت منطقی باشند، جدول ۳-۲ نحوه عملکرد هر یک از عملگرها را نشان می‌دهد.

جدول ۳-۲ عملکرد عملگرهای منطقی

| P Eqv Q | P Xor Q | P Or Q | P And Q | Not P | Q | P |
|---------------|---------------|--------------|---------------|----------|---|---|
| T | F | T | T | F | T | T |
| F | T | T | F | F | F | T |
| F | T | T | F | T | T | F |
| T | F | F | F | T | F | F |

همانطور که از جدول ۳-۲ برمی‌آید، عملگر Not، روی یک عبارت عمل می‌کند و ارزش آن را نقیض می‌کند. اگر عبارت دارای ارزش T باشد، نقیض آن دارای ارزش F است و برعکس.

عملگر And بر روی دو عبارت عمل می‌کند. ارزش نتیجه وقتی T است که ارزش دو عبارت T باشد و در غیر این صورت، F است.

عملگر Or روی دو عبارت عمل می‌کند و ارزش نتیجه وقتی F است که ارزش هر دو عبارت F باشد و در غیر این صورت، ارزش آن T است.

عملگر Xor روی دو عبارت عمل می‌کند و ارزش نتیجه وقتی T است که ارزش یکی از دو عبارت، T و ارزش دیگری F باشد. با Xor می‌توان دو گزینه‌ی انحصاری را مقایسه کرد. اگر هر دو گزینه‌ی Xor درست یا هر دو نادرست باشد، کل عبارت نادرست خواهد شد.

عملگر Not برای نقیض یک عبارت است.

۱. درست

۲. نادرست

عملگر Eqv روی دو عبارت عمل می‌کند و ارزش نتیجه وقتی T است که هر دو عبارت دارای ارزش یکسان باشند یعنی هر دو دارای ارزش T یا F باشند.

۳-۳ ترکیب عملگرهای رابطه‌ای و منطقی

از لحاظ تئوری، شش عملگر رابطه‌ای و ویژوال بیسیک قدرت کافی برای هر نوع مقایسه‌ای را دارند، ولی می‌توان قدرت آنها را با استفاده از عملگرهای منطقی افزایش داد. جدول ۳-۳ مثال‌هایی از عملگرهای منطقی و ویژوال بیسیک را نشان می‌دهد.

جدول ۳-۳ مثال‌هایی از عملگرهای منطقی و ویژوال بیسیک

| نتیجه | مثال | عملگر |
|-------|-------------------------|-------|
| True | $(2 < 3)$ And $(4 < 5)$ | And |
| True | $(2 < 3)$ Or $(6 < 7)$ | Or |
| False | $(2 < 3)$ Xor $(7 > 4)$ | Xor |
| False | Not $(3 = 3)$ | Not |

با عملگرهای منطقی می‌توان دو یا چند مقایسه‌ی شرطی را با هم ترکیب کرد. کاربرد And و Or بیشتر از سایر عملگرهاست.

به عبارت زیر که ترکیب دو عملگر شرطی با And منطقی است، توجه کنید:

$(\text{curSales} < \text{curMinSales}) \text{ And } (\text{intYrsEmp} > 10)$

در این عبارت، اگر فروش سال جاری یک کارمند از حداقل فروش تعیین شده کمتر باشد و تعداد سال‌های کاری وی هم از ۱۰ سال بیشتر باشد (به انتخاب نام متغیرها دقت کنید!) کل عبارت درست خواهد شد. البته می‌توانستید این دو عبارت را جداگانه مقایسه کنید، ولی And امکان می‌دهد تا آنها را در یک عبارت متمرکز کنید. البته ترکیب بیش از حد عبارت‌ها، آنها را پیچیده خواهد کرد. به عنوان مثال، عبارت زیر شاید حتی برای نویسنده‌ی آن هم چندان مفهوم نباشد:

$(a > 6) \text{ And } (b < 1) \text{ Or Not } (1 = c) \text{ Xor } (d = 4)$

تقدم عملگرها بر اجرای آنها تأثیر بسزایی دارد. عبارت زیر را در نظر بگیرید:

$$\text{curSales} * \text{sngCommission} > \text{curHighSales} / 10$$

ویژوال بیسیک کدام عمل را زودتر انجام خواهد داد؟ آیا ابتدا `sngCommission` را با `curHighSales` مقایسه کرده و حاصل آن را در `curSales` ضرب و سپس بر ۱۰ تقسیم خواهد کرد؟ البته این کار بی‌معنی است، چون حاصل مقایسه‌ی `True` یا `False` است و نمی‌توان روی آن اعمال محاسباتی انجام داد. در جدول ۳-۴ تقدم عملگرها را مشاهده می‌کنید. با توجه به این جدول حدس نتیجه‌ی عبارت فوق، ساده خواهد بود.

جدول ۳-۴ ترتیب عملگرهای محاسباتی، مقایسه‌ای و منطقی

| عملگر | ترتیب |
|------------------------|-------|
| ^ | ۱ |
| - (تقریبی یکانی) | ۲ |
| *, / | ۳ |
| \ | ۴ |
| Mod | ۵ |
| -, + | ۶ |
| =, <, >, <=, >= | ۷ |
| Not, And, Or, Xor, Eqv | ۸ |

نکته

ترتیب اجرای عملگرهای منطقی مطابق با جدول ۳-۴، از چپ به راست است.

باز هم تکرار می‌کنیم که برای وضوح بیشتر و تغییر اولویت‌ها در عبارت، تا آنجا که امکان دارد از پرانتزها استفاده کنید. عبارت فوق هم با استفاده از پرانتز می‌تواند قابل فهم‌تر شود:

$$(\text{curSales} * \text{sngCommission}) > (\text{curHighSales} / 10)$$

۳-۴ کنترل Label

از این کنترل برای **نمایش متن** استفاده می‌شود که این متن قابل ویرایش به وسیله‌ی کاربر نیست. اغلب از کنترل برجسب (Label) برای شناسایی شیء‌های روی فرم استفاده می‌شود (به‌عنوان مثال، در صورتی که روی آن کلیک شود، توضیحی درباره‌ی یک کنترل ارائه می‌کند). همچنین می‌توان کدی نوشت که متن نمایش یافته به وسیله‌ی برجسب را در زمان اجرا و در پاسخ به یک رویداد یا عملی، تغییر دهد.

به دلیل اینکه این کنترل، فوکوس دریافت نمی‌کند، می‌توان از آن، برای ایجاد کلیدهای دسترسی به سایر کنترل‌ها استفاده کرد.

۳-۴-۱ مقداردهی توضیح برجسب

برای تغییر متن نمایش یافته در کنترل برجسب، از مشخصه‌ی Caption استفاده کنید. در زمان طراحی، می‌توان این مشخصه را با انتخاب آن از پنجره‌ی Properties کنترل، تنظیم کرد. طول مشخصه‌ی Caption را می‌توان حداکثر تا ۱۰۲۴ بایت تعیین کرد.

۳-۴-۲ ترازبندی متن

مشخصه‌ی Alignment امکان تعیین ترازبندی متن داخل کنترل برجسب را به صورت چپ چین (°)، وسط چین (۲) و راست چین (۱) فراهم می‌کند.

۳-۴-۳ مشخصه‌های AutoSize و WordWrap

به‌طور پیش‌فرض، هنگامی که متنی در مشخصه‌ی Caption وارد می‌شود، به پهنای کنترل اضافه نمی‌شود و متن به خط بعدی شکسته می‌شود و در صورت افزایش بلندی کنترل، متن نمایش داده می‌شود.

برای اینکه کنترل بتواند اندازه‌ی خودش را با محتوا تنظیم کند، مشخصه‌ی AutoSize را با True مقداردهی کنید. کنترل به صورت افقی بزرگ خواهد شد تا کل محتوای مشخصه‌ی Caption را در خود جای دهد. برای شکستن محتوا به خط بعدی و بزرگ کردن عمودی کنترل، مشخصه‌ی WordWrap را با True مقداردهی کنید.

۴-۳ کاربرد برچسب‌ها برای ایجاد کلیدهای دسترسی

اگر می‌خواهید نویسه‌ای را در مشخصه‌ی Caption برچسب، به‌عنوان کلید دسترسی تعریف کنید، مشخصه‌ی UseMnemonic را با True مقداردهی کنید. هنگامی که در کنترل برچسب، یک کلید دسترسی تعریف می‌کنید، کاربر می‌تواند کلیدهای Alt و نویسه‌ی موردنظر را فشار دهد تا فوکوس به کنترل بعدی منتقل شود. همچنین می‌توان برای سایر کنترل‌هایی که دارای مشخصه‌ی Caption هستند، کلید دسترسی ایجاد کرد. برای انجام این کار، قبل از نویسه‌ی موردنظر به‌عنوان کلید دسترسی، علامت امپرسند (&) را اضافه کنید. برای تعیین کلید دسترسی کنترل‌هایی که Caption ندارند، از یک برچسب به همراه کنترل استفاده کنید. به‌دلیل اینکه برچسب‌ها نمی‌توانند فوکوس را دریافت کنند، فوکوس به‌طور خودکار به کنترل بعدی در ترتیب پرش منتقل می‌شود. از این تکنیک می‌توان برای تعیین کلید دسترسی به کادرهای متن، کادرهای تصویر، کادرهای ترکیبی، کادرهای لیست و ... استفاده کرد.

نکته

اگر می‌خواهید علامت & را در یک برچسب نمایش دهید، مشخصه‌ی UseMnemonic را با False مقداردهی کنید.

۵-۳ کنترل Text Box (کادر متن)

از کنترل کادر متن برای نمایش اطلاعات وارد شده به‌وسیله‌ی کاربر در زمان اجرا یا متن تعیین شده برای مشخصه‌ی Text در زمان طراحی یا اجرا، استفاده می‌شود.

به‌طور کلی از این کنترل برای متن قابل ویرایش استفاده خواهد شد، اگرچه می‌توان مشخصه‌ی Locked آن را با True مقداردهی کرد تا فقط خواندنی شود. همچنین کادرهای متن، امکان نمایش متن در چندین خط، تغییر اندازه برای جاگیری متن و افزودن قالب‌بندی‌های اصلی را ارائه می‌کنند.

۳-۵-۱ مشخصه‌ی Text

متن وارد شده در کنترل کادر متن، در مشخصه‌ی Text قرار دارد. به‌طور پیش‌فرض، می‌توان در یک کادر متن تا ۲۰۴۸ نویسه وارد کرد. در صورتی که مشخصه‌ی Multiline را با True

مقداردهی کنید، می‌توانید متن وارد شده را در چند سطر نمایش داده و حداکثر تا ۳۲k نویسه وارد کنید.

۳-۵-۲ قالب‌بندی متن

با True قرار دادن مشخصه‌ی Multiline، متن به محض رسیدن به حاشیه‌ی کادر متن، به خط بعدی شکسته می‌شود و با تنظیم مشخصه‌ی ScrollBars می‌توان نوار لغزان‌های افقی، عمودی یا هر دو را به کادر متن اضافه کرد. به کمک این نوار لغزان‌ها می‌توان کل متن را مشاهده کرد. می‌توان متن را با استفاده از مشخصه‌ی Alignment به صورت چپ‌چین، وسط‌چین یا راست‌چین تنظیم کرد. به طور پیش فرض، متن چپ‌چین است.

۳-۵-۳ ایجاد کادر متن گذرواژه

یک **گذرواژه**، کادر متنی است که امکان تایپ گذرواژه را برای کاربر فراهم می‌کند که گذرواژه با نویسه‌های جایگزین، مثل ستاره نمایش داده می‌شود. برای این منظور، ویژوال بیسیک دو مشخصه (PasswordChar و Maxlength) برای کادر متن ارائه کرده است که ایجاد کادر متن گذرواژه را ساده می‌کنند.

مشخصه‌ی PasswordChar، نویسه‌ای را که در کادر متن نمایش داده می‌شود، تعیین می‌کند. به عنوان مثال، اگر بخواهید در کادر گذرواژه، ستاره نمایش داده شود، در مشخصه‌ی PasswordChar از پنجره‌ی Properties باید مقدار * را تعیین کنید. بدین ترتیب، به ازای تایپ هر نویسه‌ای در کادر متن، یک ستاره نمایش داده خواهد شد. اگر کادر متن در حالت چندخطی قرارگیرد، این مشخصه عمل نخواهد کرد.

تعیین تعداد نویسه‌هایی که می‌توان در کادر متن تایپ کرد به کمک مشخصه‌ی Maxlength امکان‌پذیر است. با انجام این کار، رایانه‌ی شخصی بعد از تعداد تعیین شده، نویسه‌ی اضافی را قبول نکرده و بوق می‌زند.

۳-۶ دستور If

یکی از مهم‌ترین دستورهای زبان برنامه‌نویسی ویژوال بیسیک، دستور If است. شکل کلی دستور If چنین است:

```
If conditional Then
    (دستورهای ویژوال بیسیک) ...
End If
```

که در آن conditional یک عبارت یا متغیر شرطی است که True یا False را برمی‌گرداند. البته اگر دستور بدنه‌ی If فقط یکی باشد، می‌توان از شکل خلاصه‌شده‌ی If استفاده کرد:

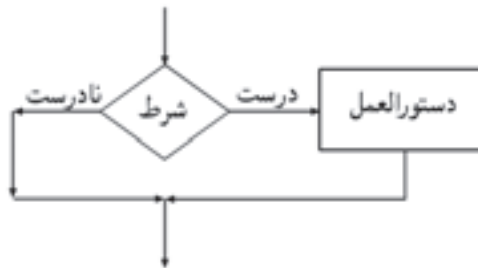
(دستورهای ویژه‌ی بیسیک) If conditional Then

برای وضوح برنامه بهتر است که بدنه‌ی If را جلوتر از If ... End If بنویسید تا تشخیص دستورهای درون If از دستورهای بیرون آن ساده‌تر باشد. مردم در زندگی روزمره‌ی خود به‌دفعات از If استفاده می‌کنند:

If (کارم زودتر تمام خواهد شد) Then (زودتر سرکار بروم)

If (نمره‌ی قبولی خواهم گرفت) Then (درس را خوب بخوانم)

ویژوال بیسیک هم از دستورهای If به‌همین ترتیب استفاده می‌کند. نحوه‌ی اجرای دستورهای If چنین است: اگر conditional درست باشد، دستورهای بدنه‌ی If اجرا خواهد شد. این دقیقاً همان چیزی است که مردم از جملات شرطی می‌فهمند. به دستورهای If برنامه‌ی ۱-۳ توجه کنید.



برنامه‌ی ۱-۳ مقایسه‌ی داده‌ها با If

1. Dim DblAge As Double

2. DblAge = 15

3. **If** (DblAge < 20) **Then**

بررسی سن کاربر

4. lblMessage.Caption = "you're young"

5. lblMessage.BackColor = Red

6. lblMessage.FontBold = True

7. **End If**

8. ادامه‌ی کد در اینجا آورده می‌شود.

اگر مقدار `curSales` از `curSalesGoal` بیشتر باشد، چهار دستور خطوط ۳ تا ۶ اجرا خواهند شد؛ اما اگر چنین نباشد، دستورهای خطوط ۳ تا ۶ اجرا نخواهند شد. در هر حال (چه بدنه‌ی `If` اجرا شود، چه نشود) خط بعد از دستور `If` (خط ۸) اجرا خواهد شد. در اینجا داده‌ها هستند که نحوه‌ی اجرای برنامه را تعیین می‌کنند و در واقع، برنامه تصمیم‌گیری می‌کند^۱ (پرانتهای اطراف شرط `If` الزامی نیستند ولی به خوانایی این دستور کمک می‌کنند).

۱-۶-۳ کامل کردن `If` با `Else`

در قسمت قبل با دستور `If` آشنا شدید، ولی برنامه‌نویسان معمولاً از شکل کامل `If` استفاده می‌کنند:

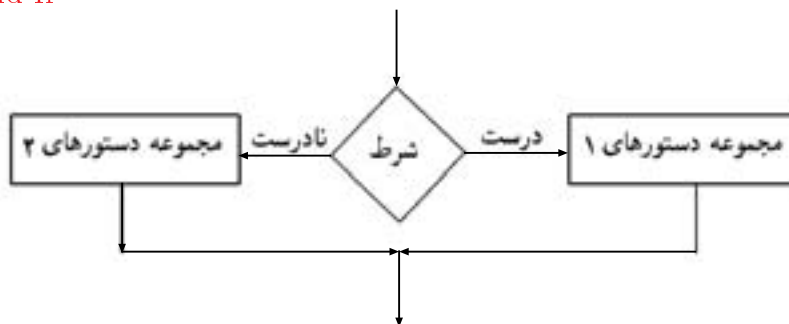
`If conditional Then`

... (دستورهای ویژه‌ی بیسیک)

`Else`

... (دستورهای ویژه‌ی بیسیک)

`End If`



در نوع ساده‌ی دستور `If`، فقط پردازش حالت درست `conditional` (شرط) انجام می‌شد ولی در نوع کامل `If`، حالت نادرست شرط هم می‌تواند پردازش شود. قسمت `Else` اجرای حالت نادرست شرط را بر عهده دارد.

اگر متن موجود در کادر متن `txtPassword` برابر با رشته‌ی "Afshin" بود
دوبار صدای بوق رایانه را به صدا درآور

اعلان برجسب `lblPrompt` را برابر با رشته‌ی "پول را به من نمایش بده" قرار بده

۱. عبارت‌هایی که بعد از علامت آپستروف (?) می‌آیند به عنوان توضیح (comment) در نظر گرفته می‌شوند و اجرا نمی‌شوند.

در غیر این صورت

اعلان برجسب lblPrompt را برابر با رشته‌ی "گذرواژه اشتباه است - دوباره سعی کنید" قرار بده

محتوای کادر متن txtPassword را پاک کن

فوکوس را به کادر متن txtPassword انتقال بده

برنامه‌ی ۲-۳ بررسی گذرواژه

1. **If** txtPassword.Text = "Afshin" **Then**
2. گذرواژه معتبر است؛
3. **Beep**
4. **Beep**
5. lblPrompt.Caption = "Show me the money"
6. **Else**
7. lblPrompt.Caption = "Wrong password – Try Again"
8. txtPassword.Text = "" حذف گذرواژه‌ی قبلی؛
9. txtPassword.SetFocus انتقال فوکوس به کادر متن؛
10. **End If**

خط ۱، محل مقایسه‌ی محتویات کادر متن با گذرواژه است. اگر این مقایسه به نتیجه‌ی True منجر شود، دستورهای بعد از If (خط ۲ تا ۵) اجرا می‌شوند؛ رایانه‌ی شخصی دوبار بوق می‌زند و عنوان برجسب هم پیام درستی گذرواژه را نشان می‌دهد. اما اگر مقایسه False شود، قسمت بعد از Else (خط ۷ تا ۹) اجرا خواهد شد و سیستم پیام نادرست بودن گذرواژه و شروع مجدد عملیات را خواهد داد.

تمرین

برنامه‌ای بنویسید که یک گذرواژه را از ورودی دریافت کرده و در صورت درست بودن آن، دکمه‌ی فرمانی به نام Exit را فعال کند و در غیر این صورت، آن را غیرفعال نماید.

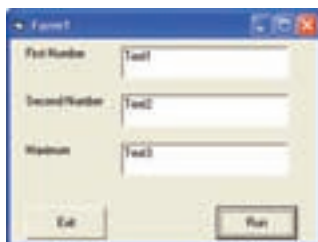
مثال ۳-۱

برنامه‌ی زیر، دو عدد را از ورودی دریافت کرده و با فشار دادن دکمه‌ی Run، بزرگ‌ترین مقدار بین آنها را در یک TextBox نمایش می‌دهد.

مراحل کار:

برنامه‌ی ویژوال بیسیک را اجرا کنید و از کادر محاوره‌ای New Project گزینه‌ی Standard EXE را انتخاب کرده و OK را فشار دهید:

۱. فرمی به صورت شکل ۳-۱ طراحی کنید و کد مربوط به دکمه‌ی Exit آن را نیز بنویسید.



شکل ۳-۱

۲. حال لازم است کد برنامه را اضافه کنیم. برای این کار روی فرم دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub Form_Load( )
```

و

```
End Sub
```

دستورهای زیر را وارد کنید:

```
Text1.Text = ""
```

```
Text2.Text = ""
```

```
Text3.Text = ""
```

همانطور که می‌دانید پس از اجرای برنامه، ابتدا رویداد Load مربوط به فرم اجرا می‌شود و این دستورها سبب می‌شوند که کادرهای متن قبل از شروع کار، پاک شوند.

۳. روی دکمه‌ی Run دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub CmdRun_Click()
```

```
End Sub
```

و

دستورهای زیر را وارد کنید:

اگر مقدار کادر متن اول بزرگ‌تر از مقدار کادر متن دوم بود

مقدار کادر متن اول را در کادر متن سوم قرار بده

در غیر این صورت

مقدار کادر متن دوم را در کادر متن سوم قرار بده

```
If Val(Text1.Text) > Val(Text2.Text) Then
```

```
    Text3.Text = Text1.Text
```

```
Else
```

```
    Text3.Text = Text2.Text
```

```
End If
```

۴. برنامه را اجرا کرده و دو عدد در کادر متن اول و دوم بنویسید و روی دکمه‌ی Run

کلیک کنید. در این صورت، عدد بزرگ‌تر در کادر متن سوم نمایش می‌یابد.

لیست کامل برنامه به صورت زیر است:

Option Explicit

```
Private Sub CmdExit_Click()
```

```
    End
```

```
End Sub
```

```
Private Sub CmdRun_Click()
```

```
    If Val(Text1.Text) > Val(Text2.Text) Then
```

```
        Text3.Text = Text1.Text
```

```
    Else
```

```
        Text3.Text = Text2.Text
```

```
    End If
```

```
End Sub
```

```
Private Sub Form_Load()
```

```
Text1.Text = "" ' خالی کردن کادرهای متن در ابتدا'
```

```
Text2.Text = ""
```

```
Text3.Text = ""
```

```
End Sub
```

نکته

از آنجایی که مشخصه‌ی Text کنترل کادر متن از نوع رشته‌ای است، نمی‌توانیم از آنها برای مقایسه‌ی اعداد استفاده کنیم. در نتیجه باید به‌وسیله‌ی تابعی داخلی به نام Val ابتدا ارزش عددی آنها را به دست آورده و سپس عمل مقایسه را انجام دهیم. شکل کلی این تابع به صورت زیر است:

```
Val (string)
```

باید توجه داشت که نوع خروجی این تابع به عددی که بعد از تبدیل به دست می‌آید، وابسته است.

```
intR=VAL("123")
```

```
lngX=VAL("12391873")
```

```
sgnY=VAL("123.876")
```

(توضیحات بیشتر را در ادامه‌ی کتاب مشاهده خواهید کرد.)

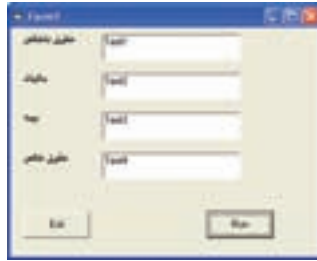
مثال ۲-۳

برنامه‌ی زیر، حقوق کارمندی را از ورودی دریافت کرده و با کلیک روی دکمه‌ی Run، بیمه، مالیات و حقوق خالص او را نمایش می‌دهد. فرمول‌های محاسبه به صورت زیر است:

| | | |
|--------------------------------------|------------------------|-----------------------------|
| مالیات | شرط | بیمه = ۹٪ از حقوق ناخالص |
| ° | حقوق >= ۳۰۰۰۰۰ | = حقوق خالص |
| $(\text{حقوق} - ۳۰۰۰۰۰) \times ۱۵\%$ | $\text{حقوق} > ۳۰۰۰۰۰$ | مالیات - بیمه - حقوق ناخالص |

مراحل کار:

برنامه‌ی ویژوال بیسیک را اجرا کرده و پروژه‌ی جدیدی ایجاد کنید. فرمی به صورت شکل ۳-۲ طراحی کنید و کد مربوط به دکمه‌ی Exit آن را نیز بنویسید.



شکل ۳-۲

۱. حال لازم است کد برنامه را اضافه کنیم. برای این کار روی فرم دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub Form_Load( )
```

و

```
End Sub
```

دستورهای زیر را وارد کنید:

```
Text1.Text = ""
```

```
Text2.Text = ""
```

```
Text3.Text = ""
```

```
Text4.Text = ""
```

۲. روی دکمه‌ی Run دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub CmdRun_Click()
```

و

```
End Sub
```

دستورهای زیر را وارد کنید:

اگر مقدار کادر متن اول بزرگ‌تر یا مساوی ۳۰۰۰۰۰ بود

کادر متن دوم $\rightarrow ۱۰۰ \div ۱۵ \times (۳۰۰۰۰۰ - \text{مقدار کادر متن اول})$

در غیر این صورت

کادر متن دوم \rightarrow \circ

کادر متن سوم $\rightarrow 100 \div 9 \times$ مقدار کادر متن اول

کادر متن چهارم \rightarrow مقدار کادر متن سوم - مقدار کادر متن دوم - مقدار کادر متن اول

If Val(Text1.Text) >= 300000 **Then**

Text2.Text = (Val(Text1.Text) - 300000) * 15/100

Else

Text2.Text = 0

End If

Text3.Text = Val(Text1.Text) * 9/100

Text4.Text = Val(Text1.Text) - Val(Text2.Text) -

Val(Text3.Text)

برنامه را اجرا کرده و حقوق ناخالص را وارد کنید و روی دکمه‌ی Run کلیک کنید.

در این صورت مالیات، بیمه و حقوق خالص نمایش می‌یابد.

لیست کامل برنامه به صورت زیر است:

Option Explicit

Private Sub CmdExit_Click()

End

End Sub

Private Sub CmdRun_Click() *'Tax calculation*

If Val(Text1.Text) >= 300000 **Then**

Text2.Text = (Val(Text1.Text) - 300000) * 15/100

Else

Text2.Text = 0

End If

Text3.Text = Val(Text1.Text) * 9/100

Text4.Text = Val(Text1.Text) - Val(Text2.Text) -

Val(Text3.Text)

```

End Sub
Private Sub Form_Load()
    Text1.Text = ""
    Text2.Text = ""
    Text3.Text = ""
    Text4.Text = ""
End Sub

```

۲-۶-۳ خروج زودرس

بسته به شرایط یک روال، ممکن است بخواهید روال از حالت معمول، زودتر پایان یابد. برای این منظور باید از دستور Exit استفاده کنید. شکل کلی دستور Exit چنین است:

Exit Sub | Function | Do | For

خطوط عمودی بین کلمات، نشان‌دهنده‌ی آن است که در هر دستور Exit یکی از این کلمات را می‌توان به‌کار برد. به‌عنوان مثال، برای خروج از یک روال رویداد باید از دستور Exit Sub و برای خروجی از یک تابع باید از دستور Exit Function استفاده کنید. قبل از پایان فصل بعدی، محل استفاده از Exit Do و Exit For را هم خواهید دید. در برنامه‌ی ۳-۳، اگر شرط If درست باشد روال خاتمه خواهد یافت (خط ۳).

برنامه‌ی ۳-۳ استفاده از Exit Sub برای خاتمه‌ی زودرس یک روال

```

Private Sub cmdButton_Click ()
    If (txtPoint < 17) Then بررسی نمره‌ی دانش‌آموز
        Exit Sub
    Else
        lblMessage.Caption = "آفرین"
    End If
End Sub

```

کنجکاوی

تفاوت روال و تابع چیست؟

۳-۶-۳ دستورهای If ... Else متداخل

اگر یک دستور If را بعد از Else، دستور If دیگری قرار دهید، به آنها دستورهای If تودرتو (nested If) می‌گویند. به برنامه‌ی ۳-۴ توجه کنید.

برنامه‌ی ۳-۴ با ElseIf می‌توان دو دستور If ... Else را در داخل هم قرار داد.

اگر ساعت کاری کمتر یا مساوی ۴۰ ساعت بود،

اضافه‌کاری \rightarrow

در غیر این صورت اگر ساعت کاری کمتر یا مساوی ۵۰ بود،

اضافه‌کاری \rightarrow حقوق هر ساعت کاری $\times 1.5 \times (ساعت کاری - 40)$

در غیر این صورت

اضافه‌کاری \rightarrow حقوق هر ساعت کاری $((10 \times 1.5) + 2 \times (ساعت کاری - 50))$

1. **If** (intHours <= 40) **Then**
2. curOverTime = 0.0
3. **ElseIf** (intHours <=50) **Then**
4. curOverTime = (intHours - 40) * 1.5 * sngRate
5. **Else**
6. curOverTime = ((intHours - 50) * 2 + (10 * 1.5)) * sngRate
7. **End If**

با دستور ElseIf در خط ۳، یک بلوک جدید If ... Else شروع می‌شود. اگر ساعت‌های کار کمتر از ۴۰ ساعت نباشد (خط ۱)، باید بیش از ۴۰ باشد. بنابراین، خط ۳ بررسی می‌کند که آیا ساعت‌های کار بین ۴۰ تا ۵۰ ساعت است (اگر ساعت‌های کار کمتر از ۴۰ باشد، خط ۳ هرگز اجرا نخواهد شد). در این صورت اضافه‌کاری بر مبنای ۱/۵ برابر دستمزد پایه محاسبه

می‌شود. اگر خط ۳ هم False شود، یعنی ساعت‌های کار بیش از 5° ساعت است. خط ۶ یک دستور پیچیده است که اضافه‌کاری را در این حالت به این صورت محاسبه می‌کند که ساعت‌های بین 4° تا 5° (10° ساعت) بر مبنای $1/5$ برابر دستمزد پایه و بیش از 5° ساعت بر مبنای ۲ برابر دستمزد پایه محاسبه خواهد شد.

دستورهای If ... ElseIf ... End If (حتی در ساده‌ترین شکل، مانند مثال بالا) پیچیده‌اند و امکان بروز خطا در آنها بسیار زیاد است. در ادامه‌ی فصل، مشاهده خواهید کرد که چگونه می‌توان با دستور Select Case به راه‌حل بهتری دست یافت.

مثال ۳-۳

برنامه‌ی زیر، نمره‌ی یک دانش‌آموز را از ورودی دریافت کرده و سپس بر اساس جدول زیر رتبه‌ی وی را نمایش می‌دهد.

اگر نمره‌ی دانش‌آموز بزرگ‌تر از 20° یا کوچک‌تر از صفر بود، پیغام خطا در ورود عدد را نمایش بده

از برنامه خارج شو

اگر نمره‌ی دانش‌آموز بزرگ‌تر یا مساوی ۱۸ و کوچک‌تر یا مساوی 20° بود، A را در کادر متن قرار بده

اگر نمره‌ی دانش‌آموز بزرگ‌تر یا مساوی ۱۶ و کوچک‌تر از ۱۸ بود، B را در کادر متن قرار بده

اگر نمره‌ی دانش‌آموز بزرگ‌تر یا مساوی ۱۴ و کوچک‌تر از ۱۶ بود، C را در کادر متن قرار بده

اگر نمره‌ی دانش‌آموز بزرگ‌تر یا مساوی ۱۲ و کوچک‌تر از ۱۴ بود، D را در کادر متن قرار بده

اگر نمره‌ی دانش‌آموز بزرگ‌تر یا مساوی 10° و کوچک‌تر از ۱۲ بود، E را در کادر متن قرار بده

اگر نمره‌ی دانش‌آموز کوچک‌تر از 10° بود، F را در کادر متن قرار بده

| رتبه | نمره |
|------|------------------------------|
| A | $18 \leq \text{Num} \leq 20$ |
| B | $16 \leq \text{Num} < 18$ |
| C | $14 \leq \text{Num} < 16$ |
| D | $12 \leq \text{Num} < 14$ |
| E | $10 \leq \text{Num} < 12$ |
| F | $\text{Num} < 10$ |

مراحل کار:

پروژه‌ی جدیدی را شروع کنید:

۱. فرمی به صورت شکل ۳-۳ طراحی کنید و کد مربوط به دکمه‌ی Exit آن را نیز بنویسید.



شکل ۳-۳

۲. کد برنامه را اضافه کنید. برای انجام این کار، روی فرم دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub Form_Load( )
```

و

```
End Sub
```

دستورهای زیر را وارد کنید:

```
Text1.Text = ""
```

```
Text2.Text = ""
```

۳. روی دکمه‌ی Run دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub cmdRun_Click()
```

و

```
End Sub
```

دستورهای زیر را وارد کنید:

```
Dim sngNum As Single
```

```
sngNum = Val(Text1)
```

```
If sngNum > 20 Or sngNum < 0 Then
```

```
Text2 = "Error in Number"
```

```

Exit Sub
End If
If sngNum >= 18 And sngNum <= 20 Then
    Text2 = "A"
ElseIf sngNum >= 16 And sngNum < 18 Then
    Text2 = "B"
ElseIf sngNum >= 14 And sngNum < 16 Then
    Text2 = "C"
ElseIf sngNum >= 12 And sngNum < 14 Then
    Text2 = "D"
ElseIf sngNum >= 10 And sngNum < 12 Then
    Text2 = "E"
ElseIf sngNum < 10 Then
    Text2 = "F"
End If

```

۴. برنامه را اجرا و نمره‌ای را در کادر متن اول وارد کرده و روی دکمه‌ی Run کلیک کنید. در این صورت، رتبه‌ی دانش‌آموز در کادر متن دوم نمایش می‌یابد.

نکته

در ابتدای برنامه، نمره‌ی دانش‌آموز در متغیری به نام sngNum ذخیره می‌شود. سپس اگر نمره از محدوده‌ی ۰ تا ۲۰ خارج بود، پیام خطایی چاپ کرده و از روال خارج می‌شود و در غیر این صورت، مراحل بررسی چند قسمتی آغاز می‌شود.

لیست کامل برنامه به صورت زیر است:

```

Option Explicit
Private Sub cmdExit_Click()
    End
End Sub

```

```
Private Sub cmdRun_Click()  
    Dim sngNum As Single  
    sngNum = Val(Text1)  
    If sngNum > 20 Or sngNum < 0 Then  
        Text2 = "Error in Number"  
        Exit Sub  
    End If  
    If sngNum >= 18 And sngNum <= 20 Then  
        Text2 = "A"  
    ElseIf sngNum >= 16 And sngNum < 18 Then  
        Text2 = "B"  
    ElseIf sngNum >= 14 And sngNum < 16 Then  
        Text2 = "C"  
    ElseIf sngNum >= 12 And sngNum < 14 Then  
        Text2 = "D"  
    ElseIf sngNum >= 10 And sngNum < 12 Then  
        Text2 = "E"  
    ElseIf sngNum < 10 Then  
        Text2 = "F"  
    End If  
End Sub  
Private Sub Form_Load()  
    Text1.Text = ""  
    Text2.Text = ""  
End Sub
```

برنامه‌ی فوق را می‌توان به صورت زیر بازنویسی کرد:

```
Option Explicit  
Private Sub cmdExit_Click()
```



```
End
End Sub
Private Sub cmdRun_Click()
Dim sngNum As Single
sngNum = Val(Text1)
If sngNum > 20 Or sngNum < 0 Then
    Text2 = "Error in Number"
    Exit Sub
End If
If sngNum >= 18 Then
    Text2 = "A"
ElseIf sngNum >= 16 Then
    Text2 = "B"
ElseIf sngNum >= 14 Then
    Text2 = "C"
ElseIf sngNum >= 12 Then
    Text2 = "D"
ElseIf sngNum >= 10 Then
    Text2 = "E"
Else
    Text2 = "F"
End If
End Sub
Private Sub Form_Load()
Text1 = ""
Text2 = ""
End Sub
```

تحقیق ۳-۱

۱. آیا نتیجه‌ی دو برنامه یکی است؟
۲. چرا شرط‌های If در مثال دوم به صورت یک شرطی نوشته شده است؟

۳-۷ انتخاب با Select Case

بهترین روش برای بررسی شرط‌های چندگانه، دستور **Select Case** است. اگر تعداد دستورهای If تودرتو از سه یا چهار عدد بیشتر شود، برنامه بسیار پیچیده خواهد شد. به شکل کلی دستور Select Case توجه کنید:

عبارت موردنظر Select Case

مقدار ۱ Case

یک یا چند دستور

[مقدار ۲ Case

] یک یا چند دستور

[مقدار ۳ Case

] یک یا چند دستور

...

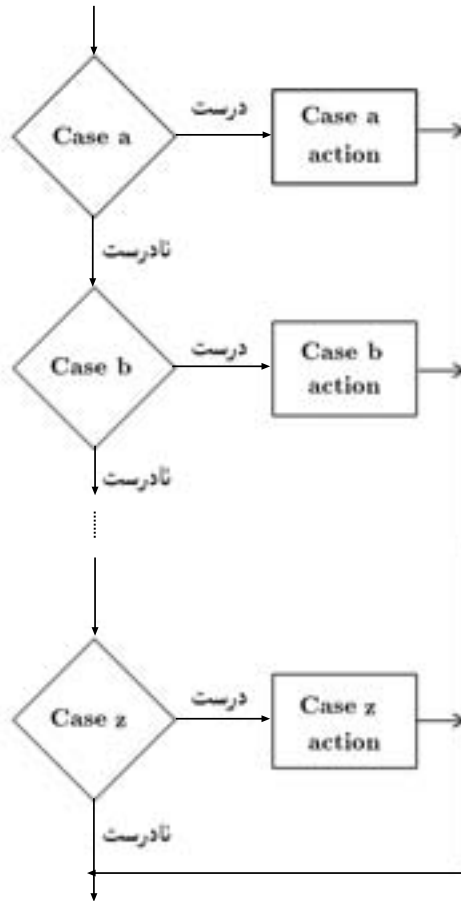
[مقدار N Case

] یک یا چند دستور

[Case Else

] یک یا چند دستور

End Select



این دستور یکی از شرط‌های متعدد را انتخاب می‌کند. تعداد دستورهای Case، به تعداد شرط‌های مسأله بستگی دارد. اگر هیچ یک از شرط‌های مسأله واقع نشوند، دستورهای قسمت Case Else اجرا خواهند شد. توجه داشته باشید که نوع داده‌ی عبارت موردنظر (در قسمت Select Case) باید با مقادیر (در قسمت‌های Case) یکسان باشد. به برنامه‌ی ۳-۵ توجه کنید.

برنامه‌ی ۳-۵ انتخاب از میان چند مقدار از دستورهای Select Case

اگر مقدار کادر متن برابر با یکی از مقادیر A تا F بود، پیغام مربوطه را نمایش بده، در غیر این صورت پیغام خطا را نمایش بده.

1. بررسی رتبه‌ی حرفی دانش‌آموز؛
2. `Select Case txtGrade.Text`
3. `Case "A"`

4. `lblAnnounce.Caption = "Perfect!"`
5. `Case "B"`
6. `lblAnnounce.Caption = "Great!"`
7. `Case "C"`
8. `lblAnnounce.Caption = "Study harder!"`
9. `Case "D"`
10. `lblAnnounce.Caption = "Get help!"`
11. `Case "F"`
12. `lblAnnounce.Caption = "Back to basics!"`
13. `Case Else`
14. `lblAnnounce.Caption = "Error in grade"`
15. `End Select`

اگر کادر متن txtGrade حاوی حرف A باشد، خط ۳ و ۴ اجرا شده و پس از آن برنامه از دستور بعد از End Select ادامه خواهد یافت. اگر کادر متن txtGrade حاوی حرف B باشد، خط ۵ و ۶ اجرا شده و دستور Select Case پایان خواهد یافت و ... دستور Case می‌تواند چندین دستور داشته باشد. ویژوال بیسیک بعد از بررسی عبارت موردنظر، تمام دستوراتی را که بین Case متناظر و Case بعد از آن بیاید، اجرا خواهد کرد. در مثال فوق، دستورهای Case ساده و یک خطی هستند. اگر کادر متن txtGrade حاوی یکی از حروف A و B، C، D یا F نباشد، دستور بعد از Case Else اجرا خواهد شد.

ویژوال بیسیک نوع دیگری از Select Case دارد که در آن از عملگر شرطی Is برای بررسی شرطها استفاده می‌کند. در برنامه‌ی ۳-۶ نوع تکامل یافته‌ی برنامه‌ی ۳-۵ را مشاهده می‌کنید.

برنامه‌ی ۳-۶ نوع دیگری از Select Case

1. بررسی نمره‌ی عددی دانش‌آموز '،
2. `Select Case txtGrade.Text`
3. `Case Is >= 90`
4. `lblAnnounce.Caption = "Perfect!"`
5. `Case Is > = 80`
6. `lblAnnounce.Caption = "Great!"`

7. **Case Is** >= 70
8. lblAnnounce.Caption = "Study harder!"
9. **Case Is** > = 60
10. lblAnnounce.Caption = "Get help!"
11. **Case Else**
12. lblAnnounce.Caption = "Back to basics!"
13. **End Select**

در این برنامه، بررسی نه بر اساس یک مقدار، بلکه بر اساس یک محدوده صورت خواهد گرفت و این پیشرفت محسوسی است. (توجه داشته باشید که در بررسی دستور Select Case نمی‌توان از عملگرهای منطقی استفاده کرد و این محدودیت جدی برای این دستور است. اگر به بررسی‌های پیچیده و شامل عملگرهای منطقی نیاز دارید، باید از همان If ... ElseIf ... End If استفاده کنید.)

نکته

اگر بعد از مقدار Case فقط یک دستور داشته باشیم، آن دستور را می‌توان بعد از شرط جلوی Case بعد از علامت : قرار داد.

مثال ۳-۴

برنامه‌ی زیر، همان برنامه‌ی مثال ۳-۳ است با این تفاوت که از Select به جای If استفاده کرده‌ایم.

```
Private Sub cmdRun_Click()
Dim sngNum As Single
sngNum = Val(Text1)
If sngNum > 20 Or sngNum < 0 Then
    Text2 = "Error in Number"
Exit Sub
End If
Select Case sngNum
```

```
Case Is >= 18
```

```
Text2 = "A"
```

```
Case Is >= 16
```

```
Text2 = "B"
```

```
Case Is >= 14
```

```
Text2 = "C"
```

```
Case Is >= 12
```

```
Text2 = "D"
```

```
Case Is >= 10
```

```
Text2 = "E"
```

```
Case Else
```

```
Text2 = "F"
```

```
End Select
```

```
End Sub
```

در این برنامه، ابتدا نمره‌ی دانش‌آموز را بررسی کرده و اگر از محدوده‌ی مجاز خارج باشد، از روال خارج می‌شویم.

سپس نمره‌ی دانش‌آموز را به‌وسیله‌ی شرط Is بررسی می‌کنیم و اگر نمره در هر یک از محدوده‌ها باشد، رتبه‌ی مربوطه را نشان می‌دهیم. باید توجه داشت که اگر یکی از شرط‌ها درست باشد، بقیه‌ی شرط‌ها بررسی نمی‌شوند و این نکته، سبب کاهش زمان اجرای برنامه می‌شود.

تحقیق ۲-۳

آیا می‌توانید بدون شرط Is برنامه‌ی مثال ۳-۴ را بازنویسی کنید؟ چگونه؟

مثال ۳-۵

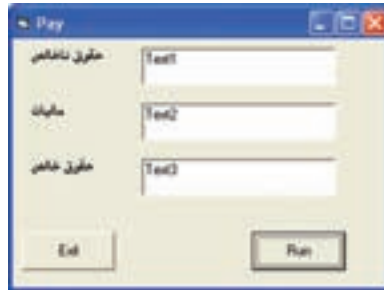
برنامه‌ی زیر، حقوق ناخالص یک کارمند را از ورودی دریافت کرده و با کلیک روی دکمه‌ی Run، میزان مالیات و حقوق خالص کارمند را در دو TextBox نمایش می‌دهد. جدول بعد، نحوه‌ی محاسبه‌ی مالیات را نشان می‌دهد.

| مالیات | حقوق ناخالص |
|---|------------------------|
| Tax=0 | $H \leq 10000$ |
| $Tax = (h - 10000) * \%5$ | $10000 < H \leq 25000$ |
| $Tax = (h - 25000) * \%10 + (25000 - 10000) * \%5$ | $25000 < H \leq 45000$ |
| $Tax = (h - 45000) * \%15 + (45000 - 25000) * \%10 + (25000 - 10000) * \%5$ | $H > 45000$ |

مراحل کار:

پروژه‌ی جدیدی را شروع کنید.

۱. فرمی به صورت شکل ۳-۴ طراحی کنید و کد مربوط به دکمه‌ی Exit آن را نیز بنویسید.



شکل ۳-۴

۲. با دوبار کلیک روی فرم، در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub Form_Load( )
```

و

```
End Sub
```

دستورهای زیر را وارد کنید:

```
Text1.Text = ""
```

```
Text2.Text = ""
```

```
Text3.Text = ""
```

۳. روی دکمه‌ی Run دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub cmdRun_Click()
```

و

```
End Sub
```

دستورهای زیر را وارد کنید:

متغیرهای sngH و sngTax را از نوع اعداد اعشاری در نظر بگیر

sngH → مقدار کادر متن اول

اگر sngH کوچک‌تر یا مساوی ۱۰۰۰۰ بود، sngTax →

اگر sngH کوچک‌تر یا مساوی ۲۵۰۰۰ بود، sngTax → $(\text{sngH} - 10000) \times 0.05$

اگر sngH کوچک‌تر یا مساوی ۴۵۰۰۰ بود،

sngTax → $(\text{sngH} - 25000) \times 0.1 + (25000 - 10000) \times 0.05$

اگر sngH بزرگ‌تر از ۴۵۰۰۰ بود،

sngTax → $(\text{sngH} - 45000) \times 0.15 + (45000 - 25000) \times 0.1 + (25000 -$

$10000) \times 0.05$

کادر متن دوم → sngTax

کادر متن سوم → sngH - sngTax

Dim sngH As Single, sngTax As Single

sngH = Val(Text1)

Select Case sngH

Case Is <= 10000

sngTax = 0

Case Is <= 25000

sngTax = (sngH - 10000) * 0.05

Case Is <= 45000

sngTax = (sngH - 25000) * 0.1 + (25000 - 10000) * 0.05

Case Is > 45000

sngTax = (sngH - 45000) * 0.15 + (45000 - 25000) *

0.1 + (25000 - 10000) * 0.05

End Select

Text2 = sngTax

Text3 = sngH - sngTax

۴. برنامه را اجرا کرده و عددی را به عنوان حقوق در کادر متن اول بنویسید و روی دکمه‌ی Run کلیک کنید. در این صورت مالیات و حقوق خالص دریافتی در دو کادر متن دیگر نمایش می‌یابد.
لیست کامل برنامه به صورت زیر است:

Option Explicit

```
Private Sub cmdExit_Click()
```

```
End
```

```
End Sub
```

```
Private Sub cmdRun_Click()
```

```
Dim sngH As Single, sngTax As Single
```

```
sngH = Val(Text1)
```

```
Select Case sngH
```

```
Case Is < 10000
```

```
sngTax = 0
```

```
Case Is <= 25000
```

```
sngTax = (sngH - 10000) * 0.05
```

```
Case Is <= 45000
```

```
sngTax = (sngH - 25000) * 0.1 + (25000 - 10000) * 0.05
```

```
Case Is > 45000
```

```
sngTax = (sngH - 45000) * 0.15 + (45000 - 25000) *
```

```
0.1 + (25000 - 10000) * 0.05
```

```
End Select
```

```
Text2 = sngTax
```

```
Text3 = sngH - sngTax
```

```
End Sub
```

```
Private Sub Form_Load()
```

```
Text1 = "" 'Clear Text boxes in Start
```

```
Text2 = ""
```

```
Text3 = ""
```

```
End Sub
```

نوع دیگر بررسی محدوده‌ها در دستور Select Case، استفاده از عملگر To است. برنامه‌ی ۳-۷ همان برنامه‌ی ۳-۶ را (این بار با استفاده از عملگر To) نشان می‌دهد.

برنامه‌ی ۳-۷ بررسی محدوده‌ها در Select Case

1. بررسی نمره‌ی عددی دانش‌آموز؛
2. **Select Case** txtGrade.Text
3. **Case 0 To 59**
4. lblAnnounce.Caption = "Back to Basics"
5. **Case 60 To 69**
6. lblAnnounce.Caption = "Get help!"
7. **Case 70 To 79**
8. lblAnnounce.Caption = "Study harder!"
9. **Case 80 To 89**
10. lblAnnounce.Caption = "Great!"
11. **Case Else**
12. lblAnnounce.Caption = "Perfect!"
13. **End Select**

دستور Select Case در برنامه‌ی ۳-۷ تفاوتی با دستور Select Case برنامه‌ی ۳-۶ ندارد؛ فقط کار را از کمترین حالت شروع کرده است. عملگر To علاوه بر اعداد می‌تواند روی رشته‌ها نیز عمل کند، مشروط بر اینکه بررسی از پایین جدول ASCII شروع شود و به سمت بالا حرکت کند.

سه نوع دستور Select Case را می‌توان با هم ترکیب کرد. به دستور زیر توجه کنید:

```
Case 101, 102, 201 To 205, is > 300
```

اگر شرط مسأله معادل ۱۰۱، ۱۰۲، ۲۰۱، ۲۰۲، ۲۰۳، ۲۰۴، ۲۰۵ و بزرگ‌تر از ۳۰۰ باشد، دستورهای این Case اجرا خواهند شد.

۳-۸ تابع IIf()

اگر دستور `If ... Else` چندان پیچیده نباشد، می‌توانید به جای آن از تابع `IIf()` استفاده کنید که شکل کلی آن به صورت زیر است:

`IIf (Condition, TrueBody, FalseBody)`

`Condition` شرط موردنظر، `TrueBody` عبارتی است که در صورت درست بودن شرط، و `FalseBody` عبارتی است که در صورت نادرست بودن شرط، برگردانده خواهد شد. به عنوان مثال:

If (`CurSales < 5000`) **Then**

`CurBonus = 0`

Else

`CurBonus = 75`

End If

به دلیل اینکه در هر قسمت `If` و `Else` یک دستور وجود دارد، می‌توان به جای آن از تابع `IIf()` به شکل زیر استفاده کرد:

`CurBonus = IIf (CurSales < 5000, 0, 75)`

به دلیل اینکه تابع `IIf()` بسیار محدود است، بهتر است از دستور `If ... Else` استفاده کنید مگر در مواقعی که شرط دارای دو حالت باشد.

در مثال زیر برای اطمینان از عدم تقسیم بر صفر (که از نظر ریاضی تعریف نشده است) از یک دستور `IIf()` استفاده شده است:

`curAveSales = IIf (intQty > 0, curTotalSales / intQty, Null)`

از آنجایی که ویژوال بیسیک همیشه صفر را به عنوان `False` تعبیر می‌کند، دستور فوق را حتی ساده تر هم می‌توان نوشت:

`curAveSales = IIf (intQty, curTotalSales / intQty, Null)`

تمرین

فرمی طراحی کنید که دارای یک کادر متن و یک دکمه‌ی فرمان باشد. به کمک تابع `IIf()` کدی بنویسید که با کلیک روی دکمه‌ی فرمان تشخیص دهد آیا مقدار واردشده در کادر متن برابر با کلمه‌ی `computer` است یا نه؟ (در خروجی پیغام مناسبی نمایش داده شود).

۳-۹ تابع `Choose()`

همانطور که `If ... Else` یک نوع ساده شده دارد، تابع `Choose()` هم نوع ساده شده‌ی دستور `Select Case` است. این تابع بر اساس مقدار اولین آرگومان، یکی از آرگومان‌های بعدی را انتخاب می‌کند. یعنی اگر آرگومان اول ۱ باشد، عبارت اول، اگر ۲ باشد، عبارت دوم و ...:

`Choose (intIndexNum, expression1 [, expression2]...)`

این تابع می‌تواند به تعداد دلخواهی آرگومان داشته باشد (البته آرگومان‌های اول و دوم الزامی هستند). اولین آرگومان یعنی `intIndexNum`، باید متغیری باشد که از ۱ تا آخرین آرگومان موجود، مقدار می‌گیرد. برای تهیه‌ی لیست‌های ساده، تابع `Choose()` مؤثرتر از دستورهای `If` یا `Select` است. ولی چون اولین آرگومان آن فقط می‌تواند عدد صحیح باشد، محدودیت زیادی دارد. اگر `intIndexNum` بین ۱ و شماره‌ی آخرین آرگومان نباشد، تابع `Choose()` مقدار `Null` برمی‌گرداند.

اولین آرگومان `Choose()` می‌تواند یک عبارت باشد و برنامه‌نویس باید مراقب باشد که مقدار عبارت از محدوده‌ی مجاز خارج نشود. توجه داشته باشید که این اندیس باید از ۱ شروع شود تا تابع `Choose()` بتواند درست عمل کند.

فرض کنید می‌خواهید یک لیست کد قیمت بسازید. کاربر با وارد کردن یک محصول جدید باید یک کد قیمت هم وارد کند و برنامه بر اساس فهرست زیر عمل کند:

۲. ۵٪ تخفیف

۱. قیمت کامل

۴. سفارش ویژه

۳. ۱۰٪ تخفیف

۵. سفارش پستی

تابع Choose() زیر، می‌تواند بر اساس کد قیمت، توضیح مناسب را انتخاب کند.
 Descript = Choose(lblProductCode, "Full markup", "5% discount",
 "10% discount", "Special order", "Mail order")

مثال ۳-۶

برنامه‌ی زیر، عددی را به‌عنوان شماره‌ی روز سال از ورودی می‌گیرد و با فرض اینکه اولین روز سال شنبه است، نام روز هفته در آن روز خاص را نمایش می‌دهد.

مراحل کار:

پروژه‌ی جدیدی را شروع کنید.

۱. فرمی به‌صورت شکل ۳-۵ طراحی کنید و کد مربوط به دکمه‌ی Exit آن را نیز بنویسید.



شکل ۳-۵

۲. حال لازم است کد برنامه را اضافه کنیم. برای این کار روی فرم دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub Form_Load( )
```

و

```
End Sub
```

دستورهای زیر را وارد کنید:

```
Text1.Text = ""
```

```
Text2.Text = ""
```

۳. روی دکمه‌ی Run دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub cmdRun_Click()
```

و

```
End Sub
```

دستورهای زیر را وارد کنید:

```
Dim intDay As Integer, strWeekDayName As String
```

```
intDay = Val(Text1) Mod 7
```

```
strWeekDayName=Choose(intDay+1, "Fri","Sat","Sun", _  
"Mon","Tue","Wed","Thu")
```

```
Text2 = strWeekDayName
```

۴. برنامه را اجرا کرده و شماری روز سال را در کادر متن اول وارد کنید و روی دکمه Run کلیک کنید. در این صورت، در کادر متن دوم، نام روز هفته نمایش می‌یابد.

نکته

همانطور که می‌دانید اگر ایام هفته را شماره‌گذاری کنیم، متوجه می‌شویم که هر هفت روز یک‌بار ایام هفته از ابتدا تکرار می‌شوند و به همین دلیل در ابتدای برنامه، شماری روز سال را دریافت و بر هفت تقسیم کرده و باقیمانده‌ی آن را برمی‌داریم. خارج قسمت این تقسیم نشان‌دهنده‌ی تعداد هفته‌های کامل گذشته و باقیمانده‌ی تقسیم نشان‌دهنده‌ی هفته‌ی جاری است که هنوز تمام نشده است. در این صورت اگر باقیمانده یک باشد، یعنی اولین روز هفته، اگر دو باشد یعنی دومین روز هفته و ... و اگر صفر شود، یعنی آخرین روز هفته. در این برنامه از Choose استفاده کرده‌ایم. همانطور که می‌دانید عدد ورودی در پارامتر اول این تابع باید از یک آغاز شود. در نتیجه، ما هم در این تابع یک واحد به intDay اضافه می‌کنیم که در این صورت لازم است نام آخرین روز هفته را به‌عنوان اولین پارامتر انتخاب کرده و در این فرمان بنویسیم.

تحقیق ۳-۳

۱. آیا می‌توانید نام اولین روز سال را نیز دریافت کنید.
۲. اگر نام اولین روز سال دوشنبه باشد، چه تغییری باید در برنامه ایجاد شود تا به درستی کار کند؟

۳-۱۰ تابع Switch()

این تابع لیستی از عبارات را ارزیابی کرده و مقدار مربوط به اولین عبارت درست را به صورت variant برمی‌گرداند. شکل کلی این تابع به صورت زیر است:

Switch (مقدار n, عبارت n, ... [مقدار ۲, عبارت ۲], [مقدار ۱, عبارت ۱])

مثال زیر را در نظر بگیرید:

```
Matchup = Switch(CityName = "Tehran", "Persian", CityName = "Rome", "Italian", CityName = "Paris", "French")
```

در این مثال، در صورتی که مقدار متغیر CityName برابر با Tehran باشد، متغیر Matchup با Persian مقداردهی می‌شود و ...

تمرین

برنامه‌ای بنویسید که با فرض وجود ۱۰ دانش‌آموز و شماره‌ی دانش‌آموز (از ۱ تا ۱۰) شماره‌ی دانش‌آموز را از ورودی دریافت کند و نام آن دانش‌آموز را نمایش دهد. این تمرین را با هر یک از دستورات if, case, switch و choose بنویسید.

۳-۱۱ کنترل Check Box (کادر علامت)

در صورتی که این کنترل انتخاب شود، یک علامت ✓ را نمایش می‌دهد. از این کنترل معمولاً برای ارایه‌ی انتخاب Yes/No یا True/False به کاربر استفاده می‌شود. کنترل کادر علامت را می‌توان به صورت گروهی نیز به‌کار برد تا چندین انتخاب را به کاربر ارایه دهد و کاربر می‌تواند یک یا چند کادر علامت را انتخاب کند.

۳-۱۱-۱ مشخصه Value

مشخصه Value کنترل کادر علامت، تعیین می‌کند که آیا کادر علامت به وسیله کاربر انتخاب شده است یا نه و آیا این کادر علامت غیرفعال است؟ هنگامی که کادر علامت انتخاب شود، مقدار این مشخصه، ۱ خواهد بود.

جدول ۳-۵، مقادیر و ثابت‌های معادل آنها در ویژوال بیسیک را برای مشخصه Value کادر علامت نشان می‌دهد.

جدول ۳-۵

| String | Value | Constant |
|-------------|-------|-------------|
| Unchecked | 0 | vbUnchecked |
| Checked | 1 | vbChecked |
| Unavailable | 2 | vbGrayed |

با کلیک کاربر روی کادر علامت، این کنترل انتخاب شده یا از انتخاب خارج می‌شود. سپس می‌توان در برنامه، وضعیت این کنترل را بررسی کرده و عمل خاصی را بر اساس این اطلاعات انجام داد.

به‌طور پیش‌فرض، کنترل کادر متن با vbUnchecked مقداردهی می‌شود. در صورتی که می‌خواهید تعدادی از کادرهای علامت را از قبل انتخاب کنید، می‌توانید مشخصه Value آنها را در روای‌های Form_Load، Form_Initialize یا Form_Activate با vbchecked یا ۱ مقداردهی کنید.

همچنین برای غیرفعال کردن کادر علامت، مشخصه Value را با vbGrayed مقداردهی کنید. به‌عنوان مثال، ممکن است بخواهید کادر علامت را تا زمانی که شرط خاصی واقع شود، غیرفعال کنید.

۳-۱۱-۲ رویداد Click

هر زمانی که کاربر روی کنترل کادر علامت کلیک کند، رویداد Click فعال می‌شود. سپس می‌توان در برنامه، بر اساس وضعیت کادر علامت، عمل خاصی را انجام داد. در مثال زیر، مشخصه Caption کنترل کادر علامت با هر بار کلیک روی کنترل، تغییر یافته و وضعیت انتخاب یا عدم انتخاب آن را نشان می‌دهد:

```
Private Sub Check1_Click( )
```

```
    If Check1.value = vbChecked Then
```



```
Check1.Caption = "Checked"
```

```
ElseIf Check1.value = vbUnchecked Then
```

```
Check1.Caption = "Unchecked"
```

```
End If
```

```
End Sub
```

نکته

کنترل کادر علامت از رویداد Double-Click پشتیبانی نمی‌کند و اگر کاربر این عمل را انجام دهد، هر کلیک را به‌طور جداگانه اجرا خواهد کرد.

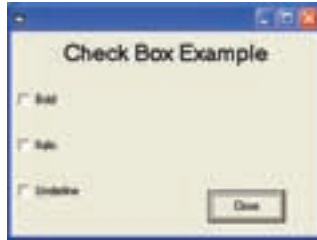
همچنین رویداد Click کادر علامت را می‌توان با انتقال فوکوس به وسیله‌ی کلید Tab به کادر علامت و سپس فشار دادن کلید SPACEBAR نیز فعال کرد. با اضافه کردن یک علامت امپرسند (&) قبل از یک حرف در مشخصه‌ی Caption کادر علامت، می‌توان کلید میانبری را ایجاد کرد. شبیه کنترل دکمه‌ی فرمان، می‌توان ظاهر کادر علامت را با تنظیم مشخصه‌ی Style و سپس استفاده از مشخصه‌های Picture، DownPicture و DisabledPicture تغییر داد. به‌عنوان مثال، ممکن است بخواهید آیکن یا تصویر نقش‌بیتی^۱ را به کادر علامت اضافه کنید یا تصویر متفاوتی را هنگامی که روی کنترل کلیک می‌شود یا غیرفعال می‌گردد، نمایش دهید. اگر می‌خواهید روی یک فرم، چند کادر علامت داشته باشید که در هر لحظه فقط یکی از آنها انتخاب شده باشد، ویژوال بیسیک استفاده از دکمه‌های انتخاب را پیشنهاد می‌دهد.

مثال ۳-۷

در برنامه‌ی زیر، می‌توان با تغییر کادر علامت، نحوه‌ی نمایش یک برجسب را تغییر داد.

مراحل کار:

۱. پروژه‌ی جدیدی را شروع کنید.
۲. فرمی به‌صورت شکل ۳-۶ طراحی کنید و کد مربوط به دکمه‌ی Close آن را نیز بنویسید.



شکل ۳-۶

۳. نام سه CheckBox را به ترتیب chkBold و chkItalic، chkUnderLine قرار دهید.

۴. نام برجسب را lbl1 قرار داده و پیام درون Caption آن را مطابق شکل تنظیم کنید.

۵. روی chkBold دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub chkBold_Click()
```

و

```
End Sub
```

دستورهای زیر را وارد کنید:

```
If chkBold.Value = 1 Then
```

```
    lbl1.FontBold = True
```

```
Else
```

```
    lbl1.FontBold = False
```

```
End If
```

۶. روی chkItalic دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub chkItalic_Click()
```

و

```
End Sub
```

دستورهای زیر را وارد کنید:

```
If chkItalic.Value = 1 Then
```

```
    lbl1.FontItalic = True
```

```
Else
```

```
    lbl1.FontItalic = False
```

```
End If
```

۷. روی chkUnderLine دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub chkUnderLine_Click()
```

و

```
End Sub
```

دستورهای زیر را وارد کنید:

```
If chkUnderLine.Value = 1 Then
```

```
    lbl1.FontUnderline = True
```

```
Else
```

```
    lbl1.FontUnderline = False
```

```
End If
```

۸. برنامه را اجرا کنید. با انتخاب هر Check Box، تغییرات را در برجسب مشاهده می‌کنید.

تحقیق ۳-۴

آیا مشخصه‌های دیگری برای Font یک برجسب وجود دارد؟ آنها را نام ببرید.

۱۲-۳ کنترل Option Button

دکمه‌ی انتخاب (Option Button) برای انتخاب انحصاری یک گزینه از میان چندگزینه به کار می‌رود. ویژگی‌های بیسیک اجازه نمی‌دهد که در هر لحظه، بیش از یکی از دکمه‌ها انتخاب شود (شکل ۳-۷ را مشاهده کنید). در این شکل، سه دکمه‌ی انتخاب وجود دارد که یکی از آنها انتخاب



شکل ۳-۷ مثالی از فرم با دکمه‌های انتخاب

شده است. اگر کار بر روی یکی از دکمه‌ها کلیک کند، وژوال بیسیک آن را انتخاب و سایر دکمه‌ها را غیرفعال خواهد کرد (وظیفه‌ای که در کادرهای علامت برعهده‌ی برنامه‌نویس گذاشته شده است). به دکمه‌های انتخاب، دکمه‌های رادیویی (Radio button) هم گفته می‌شود، چون رفتار آنها مانند دکمه‌های فشاری در رادیوهای قدیمی است.

اگر روی فرمی چند دکمه‌ی انتخاب وجود داشته باشد، در روال رویداد (Form_Load) می‌توان مشخصه‌ی Value تمام آنها را False کرد؛ بدین ترتیب در شروع برنامه هیچ کدام انتخاب شده نخواهند بود. بعد از آن کاربر می‌تواند یکی از آنها را انتخاب کند. به یاد داشته باشید که دکمه‌های انتخاب هم می‌توانند کلید دسترسی سریع داشته باشند. هرگز یک دکمه‌ی انتخاب را به تنهایی روی یک فرم قرار ندهید، زیرا فعال کردن آن ممکن است، ولی دیگر امکان غیرفعال کردن آن وجود نخواهد داشت. دکمه‌های انتخاب فقط وقتی غیرفعال می‌شوند که کاربر روی دکمه‌ی انتخاب دیگری کلیک کند.

مثال ۳-۸

این برنامه، شبیه همان برنامه‌ی قبلی است با این تفاوت که سه دکمه‌ی انتخاب برای انتخاب رنگ قلم مربوط به برجسب به آن اضافه شده است. ۱. فرم مثال قبل را به صورت شکل ۳-۸ تغییر دهید.



شکل ۳-۸ مثالی از مقایسه‌ی دکمه‌های Option با Check Box

۲. نام سه Option Button را به ترتیب OptBlue، OptGreen و OptRed قرار دهید.

۳. روی فرم دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub Form_Load()
```

```
End Sub
```

دستورهای زیر را وارد کنید:

```
OptRed.Value = True
lbl1.ForeColor = vbRed
```

نکته

به وسیله‌ی مشخصه‌ی `ForeColor` می‌توان رنگ قلم مربوط به یک شیء را تغییر داد. در ضمن با ثابت‌های نام‌دار `vbBlue`، `vbGreen` و `vbRed` می‌توان رنگ قلم را عوض کرد.

۴. روی `OptRed` دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub OptRed_Click()
```

و

```
End Sub
```

دستور زیر را وارد کنید:

```
lbl1.ForeColor = vbRed
```

۵. روی `OptGreen` دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub OptGreen_Click()
```

و

```
End Sub
```

دستور زیر را وارد کنید:

```
lbl1.ForeColor = vbGreen
```

۶. روی `OptBlue` دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub OptBlue_Click()
```

و

```
End Sub
```

دستور زیر را وارد کنید:

```
lbl1.ForeColor = vbBlue
```

۷. برنامه را اجرا کنید. با انتخاب هر Option Button تغییرات را در برجسب مشاهده می‌کنید.

لیست کامل برنامه به صورت زیر است:

Option Explicit

اگر کادر علامت برابر یک بود، مشخصه‌ی Bold برجسب ۱ را فعال کن در غیر این صورت،
مشخصه‌ی Bold را غیرفعال کن

```
Private Sub chkBold_Click()
```

```
    If chkBold.Value = 1 Then
```

```
        lbl1.FontBold = True
```

```
    Else
```

```
        lbl1.FontBold = False
```

```
    End If
```

```
End Sub
```

اگر کادر علامت برابر یک بود، مشخصه‌ی ایتالیک قلم را فعال و در غیر این صورت، آن
را غیرفعال کن

```
Private Sub chkItalic_Click()
```

```
    If chkItalic.Value = 1 Then
```

```
        lbl1.FontItalic = True
```

```
    Else
```

```
        lbl1.FontItalic = False
```

```
    End If
```

```
End Sub
```

اگر کادر علامت برابر یک بود، مشخصه‌ی زیرخط‌دار برجسب را فعال و در غیر این صورت،
آن را غیرفعال کن

```
Private Sub chkUnderLine_Click()
```

```
    If chkUnderLine.Value = 1 Then
```

```
        lbl1.FontUnderline = True
```

```
    Else
```

```
        lbl1.FontUnderline = False
```

```
    End If
```

```
End Sub
```

```
Private Sub cmdClose_Click()
```

```

End
End Sub
Private Sub Form_Load()
    OptRed.Value = True
    lbl1.ForeColor = vbRed
End Sub
Private Sub OptBlue_Click()
    lbl1.ForeColor = vbBlue
End Sub
Private Sub OptGreen_Click()
    lbl1.ForeColor = vbGreen
End Sub
Private Sub OptRed_Click()
    lbl1.ForeColor = vbRed
End Sub

```

۱۳-۳ کنترل Frame

مشاهده کردید که در هر لحظه فقط یکی از دکمه‌های انتخاب را می‌توان انتخاب (فعال) کرد، ولی از نظر تکنیکی می‌توان در یک لحظه، روی یک فرم چندین دکمه‌ی انتخاب فعال داشت. این کار با استفاده از کنترل فریم (Frame) امکان‌پذیر است. **وظیفه‌ی کنترل فریم دسته‌بندی ظاهری و منطقی کنترل‌های دیگر است.** فریم که به آن ظرف (container) هم می‌گویند، مکانی برای قرار گرفتن کنترل‌های دیگر است. چند دکمه‌ی انتخاب که روی یک فریم قرار داشته باشند، از لحاظ منطقی جزء یک گروه به حساب می‌آیند (روی فریم می‌توان کنترل‌های دیگری هم قرار داد). در شکل ۹-۳، دو گروه دکمه‌ی انتخاب وجود دارد که چون یک دسته از آنها در داخل فریم قرار دارند، می‌توان همزمان دو تا از آنها را انتخاب کرد. اگر فریم وجود نداشت، در هر لحظه فقط یکی از این پنج کنترل می‌توانست فعال باشد. به یاد داشته باشید که فریم باید قبل از دکمه‌های گزینه روی فرم قرار داده شود.

قرار دادن یک فریم روی فرم برنامه بسیار ساده است. مشخصه‌های مهم فریم‌ها عبارت‌اند از:



شکل ۳-۹

- *BorderStyle*: 0-None یا 1-Fixed Single: اگر مقدار این خاصیت 0 باشد، فریم نامرئی خواهد شد و نمی‌توان آن را از سایر قسمت‌های فرم تشخیص داد. با اینکه یک فریم نامرئی همچنان به وظیفه‌اش (دسته‌بندی کنترل‌ها) عمل خواهد کرد، برای کاربر تشخیص این مسأله که کنترل‌ها به چند دسته‌ی جداگانه تعلق دارند، مشکل خواهد بود.
- *Caption*: عنوانی که در بالای فریم ظاهر خواهد شد.
- *Font*: قلم عنوان فریم.

در مثال بعدی مشاهده خواهید کرد که چگونه می‌توان از فریم‌ها برای دسته‌بندی دکمه‌های انتخاب استفاده کرد. برای آنکه ویژوال بیسیک بتواند متوجه شود که یک دکمه‌ی انتخاب باید داخل فریم قرار گیرد (نه روی فرم) بایستی آن را روی فریم رسم کنید (روی ابزار Option Button در جعبه ابزار کلیک کرده و سپس روی فریم، آن را به اندازه‌ی موردنظر رسم کنید). اگر روی ابزار مزبور دوبار کلیک کنید، ویژوال بیسیک آن را روی فرم قرار می‌دهد، نه روی فریم.

مثال ۳-۹

این برنامه، سه دکمه‌ی انتخاب برای انتخاب رنگ قلم و سه دکمه‌ی انتخاب برای انتخاب رنگ زمینه‌ی مربوط به برجسب دارد که با تغییر آنها، رنگ نوشته را تغییر می‌دهد.

مراحل کار:

۱. پروژه‌ی جدیدی را شروع کنید.
۲. فرمی به صورت شکل ۱۰-۳ طراحی کنید و کد مربوط به دکمه‌ی Close آن را نیز بنویسید.



شکل ۱۰-۳

برای انجام این کار ابتدا:

- دو فریم به اسامی ForeColor و BackColor ایجاد کنید.
- در هر فریم سه OptionButton قرار دهید. برای انجام این کار، می‌توان Option Button را ابتدا روی فرم قرار داده و سپس با Cut و Paste آن را درون فریم قرار داد یا بر روی نشانه Option Button کلیک کرده و آن را درون فریم ترسیم کرد.

نکته

عناصر درون فریم را نمی‌توان از درون فریم خارج کرد.

۳. نام سه Option Button مربوط به رنگ قلم (ForeColor) را به ترتیب OptGreenF، OptBlueF و OptRedF قرار دهید.

۴. نام سه Option Button مربوط به رنگ زمینه (BackColor) را به ترتیب OptWhiteB، OptBlackB و OptYellowB قرار دهید.

۵. برجستگی به نام lbl1 مطابق شکل ۱۰-۳ روی فرم قرار داده و متن درون آن را تعیین کنید.

۶. روی فرم دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub Form_Load()
```

```
End Sub
```

و

دستورهای زیر را وارد کنید:

```
OptBlackB.Value = True
OptRedF.Value = True
lbl1.ForeColor = vbRed
lbl1.BackColor = vbBlack
```

نکته

به وسیله‌ی مشخصه‌ی BackColor می‌توان رنگ زمینه‌ی مربوط به یک شیء را تغییر داد. در ضمن ثابت‌های نام‌دار رنگ در جدول زیر مشخص شده است.

| ثابت | مقدار | توضیحات |
|-----------|----------|-----------|
| vbBlack | &H0 | سیاه |
| vbRed | &HFF | قرمز |
| vbGreen | &HFF00 | سبز |
| vbYellow | &HFFFF | زرد |
| vbBlue | &HFF0000 | آبی |
| vbMagenta | &HFF00FF | بنفش |
| vbCyan | &HFFFF00 | فیروزه‌ای |
| vbWhite | &HFFFFFF | سفید |

۷. روی OptRedF دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub OptRedF_Click()
```

و

```
End Sub
```

دستور زیر را وارد کنید:

```
lbl1.ForeColor = vbRed
```

۸. روی OptGreenF دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub OptGreenF_Click()
```

و

```
End Sub
```

دستور زیر را وارد کنید:

```
lbl1.ForeColor = vbGreen
```

۹. روی OptBlueF دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub OptBlueF_Click()
```

و

```
End Sub
```

دستور زیر را وارد کنید:

```
lbl1.ForeColor = vbBlue
```

۱۰. روی OptBlackB دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub OptBlackB_Click()
```

و

```
End Sub
```

دستور زیر را وارد کنید:

```
lbl1.BackColor = vbBlack
```

۱۱. روی OptYellowB دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو

دستور:

```
Private Sub OptYellowB_Click()
```

و

```
End Sub
```

دستور زیر را وارد کنید:

```
lbl1.BackColor = vbYellow
```

۱۲. روی OptWhiteB دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub OptWhiteB_Click()
```

و

```
End Sub
```

دستور زیر را وارد کنید:

```
lbl1.BackColor = vbWhite
```

۱۳. برنامه را اجرا کنید. با انتخاب هر Option Button، تغییرات را در برجسب مشاهده می‌کنید.

لیست کامل برنامه به صورت زیر است:

Option Explicit

```
Private Sub cmdClose_Click()
```

```
End
```

```
End Sub
```

```
Private Sub Form_Load()
```

```
OptBlackB.Value = True
```

```
OptRedF.Value = True
```

```
lbl1.ForeColor = vbRed
```

```
lbl1.BackColor = vbBlack
```

```
End Sub
```

```
Private Sub OptBlackB_Click()
```

```
lbl1.BackColor = vbBlack
```

```
End Sub
```

```
Private Sub OptBlueF_Click()
```

```
lbl1.ForeColor = vbBlue
```

```
End Sub
```

```
Private Sub OptGreenF_Click()
```

```
lbl1.ForeColor = vbGreen
```

```
End Sub
```

```
Private Sub OptRedF_Click()
```

```
lbl1.ForeColor = vbRed
```

```
End Sub
```

```
Private Sub OptWhiteB_Click()
```

```
    lbl1.BackColor = vbWhite
```

```
End Sub
```

```
Private Sub OptYellowB_Click()
```

```
    lbl1.BackColor = vbYellow
```

```
End Sub
```

خلاصه‌ی فصل

از عملگرهای رابطه‌ای برای مقایسه‌ی دو عبارت و از عملگرهای منطقی برای ترکیب دو عبارت منطقی استفاده می‌شود.

از کنترل Label برای نمایش متن استفاده می‌شود که این متن قابل ویرایش به وسیله‌ی کاربر نیست. اغلب از کنترل برجسب (Label) برای شناسایی شی‌های روی فرم استفاده می‌شود که به عنوان مثال، توضیحی درباره‌ی یک کنترل ارائه می‌کند. این کنترل دارای مشخصه‌هایی مانند Caption, Autosize, WordWrap, Alignment و Caption است. به کمک مشخصه‌ی UseMnemonic می‌توان برای کنترل‌های بعدی، کلید دسترسی ایجاد کرد (باید مشخصه‌ی True مقداردهی شود).

از کنترل کادر متن برای نمایش اطلاعات وارد شده به وسیله‌ی کاربر در زمان اجرا، یا متن تعیین شده برای مشخصه‌ی Text در زمان طراحی یا اجرا استفاده می‌شود. با مشخصه‌ی Text می‌توان متنی با حداکثر ۲۰۴۸ نویسه را نگه داشت.

برای بررسی درستی یا نادرستی یک عبارت منطقی، از دستور If استفاده می‌شود. در صورت درست بودن (True) شرط، دستورهای بعد از Then و در غیر این صورت، دستورهای بعد از Else اجرا خواهند شد. دستور If را می‌توان به شکل متداخل نیز به کار برد که در این صورت، If داخلی بعد از Else قرار می‌گیرد. برای بررسی شرط‌های چندگانه، از دستور Select Case استفاده می‌شود. اگر عبارت با یکی از مقادیر Caseها برابر باشد، دستورهای بعد از آن، و در غیر این صورت، دستورهای بعد از Case Else اجرا می‌شوند.

از توابع Choose, IIF و Switch نیز می‌توان برای بررسی درستی یا نادرستی یک یا چند شرط استفاده کرد.

برای دریافت پاسخ بلی/خیر کاربر، از کنترل خاصی به نام CheckBox استفاده می‌شود که مهم‌ترین مشخصه‌ی آن، Value است که از نوع منطقی است. رویداد اصلی این کنترل، Click است.

در صورتی که می‌خواهید کاربر از بین چند گزینه یکی را انتخاب کند، از کنترل Option Button استفاده کنید. مهم‌ترین مشخصه‌ی این کنترل، Value و رویداد آن، Click است. برای دسته‌بندی کنترل‌ها به ویژه کنترل Option Button، از کنترلی به نام Frame استفاده کنید.

خودآزمایی

۱. یک دستور If بنویسید که تساوی سه عدد را بررسی کند.
۲. برنامه‌ای بنویسید که سه عدد را دریافت و بزرگ‌ترین مقدار بین آنها را چاپ کند.
۳. برنامه‌ای بنویسید که سه عدد از ورودی بگیرد و تعیین کند که آیا با این سه عدد می‌توان یک مثلث ساخت؟
۴. برنامه‌ای بنویسید که سه عدد از ورودی بگیرد و تعیین کند که آیا با این سه عدد می‌توان یک مثلث قائم‌الزاویه ساخت یا نه؟
۵. فروشنده‌ای با صاحب یک فروشگاه قرارداد بسته است که اجناس وی را بفروشد و درصدی از فروش را به‌عنوان حق‌الزحمه دریافت کند. درصد حق‌الزحمه، به میزان فروش و جنس کالا بستگی دارد؟ کالاهای فروشگاه از نوع درجه‌ی ۱، درجه‌ی ۲ و درجه‌ی ۳ هستند. حق‌الزحمه به شرح زیر تعیین می‌شود:
 - الف) اگر کالا درجه‌ی ۱ باشد، به‌ازای فروش کمتر از ۵۰۰۰۰ ریال، ۱۰ درصد و به‌ازای فروش بیشتر از ۵۰۰۰۰ ریال، ۱۵ درصد حق‌الزحمه تعلق می‌گیرد.
 - ب) اگر کالا درجه‌ی ۲ باشد، به‌ازای فروش کمتر از ۵۰۰۰۰ ریال، ۴ درصد و به‌ازای فروش بیشتر از ۵۰۰۰۰ ریال، ۱۰ درصد حق‌الزحمه تعلق می‌گیرد.
 - ج) اگر کالا درجه‌ی ۳ باشد، به‌ازای هر میزان فروش، ۷ درصد حق‌الزحمه تعلق می‌گیرد.
- برنامه‌ای بنویسید که حق‌الزحمه‌ی فروشنده را با دریافت میزان فروش هر کدام از کالاها محاسبه کرده و نمایش دهد.
۶. برنامه‌ای بنویسید که ضرایب معادله‌ی درجه دوم را بگیرد و ریشه‌های آن را روی فرم چاپ کند.
۷. برنامه‌ای بنویسید که یک عدد ۴ رقمی از ورودی بگیرد و اگر آن عدد با مقلوب خود برابر بود کلمه‌ی "Yes" و در غیر این صورت "No" را چاپ کند.
۸. برنامه‌ای بنویسید که عددی را به‌عنوان ثانیه بخواند و ساعت و دقیقه و ثانیه‌ی معادل آن را چاپ کند.
۹. برنامه‌ای بنویسید که ۴ عدد از ورودی بگیرد و پیام مناسبی چاپ کند (چهار عدد مساوی هستند، سه عدد مساوی هستند، دو عدد مساوی هستند، هیچ یک مساوی نیستند).

فصل چهارم

ساختارهای تکرار

پس از پایان این فصل، انتظار می‌رود که فراگیر بتواند:

- ساختار حلقه‌های تکرار For و Do را شرح دهد.
- تفاوت بین حلقه‌های تکرار بیان شده را ذکر کند.
- با استفاده از حلقه‌های تکرار، برنامه بنویسد.
- از حلقه‌های متداخل در برنامه‌نویسی استفاده کند.

مفهوم ساختارهای تکرار را با ذکر چند مثال شرح می‌دهیم. فرض کنید در یک کارخانه‌ی تولیدی، کارگری را استخدام کرده و به او می‌گویند که برچسب خاصی را بر روی جعبه‌ی محصولات کارخانه بچسباند. ممکن است در روز هزاران محصول جدید تولید شود و این کارگر وظیفه‌ی خود را انجام دهد. نیازی نیست که برای چسباندن هر برچسب، به کارگر بگویید که چه کاری را انجام دهد. فقط کافی است همان بار اول، این عمل را به او تفهیم کنید تا بتواند در طول روز و روزهای بعد، این عمل تکراری را انجام دهد. مثال‌هایی از این قبیل، در زندگی روزمره‌ی هر فرد نیز به وفور یافت می‌شود.

در انجام عملیات رایانه‌ی نیز، بعضی از اعمال به دفعات تکرار می‌شوند و بسیاری از کارها هستند که ماهیت آنها تکراری است. به‌عنوان مثال، خواندن نام و معدل ۵۰ دانش‌آموز از ورودی، خواندن ساعت اضافه‌کاری ۱۰۰ کارمند، خواندن نمرات دانش‌آموز و محاسبه‌ی معدل وی و هزاران مثال دیگر، نمونه‌هایی از عملیات تکراری هستند.

برای انجام این‌گونه عملیات، در تمام زبان‌های برنامه‌نویسی، ساختارهایی وجود دارند که سبب تکرار اجرای دستورها می‌شوند. به این‌گونه ساختارها، «حلقه‌های تکرار» می‌گویند. حلقه‌های تکرار را می‌توان به دو گروه عمده تقسیم کرد: **حلقه‌های تکرار معین و حلقه‌های تکرار نامعین.**

حلقه‌ی تکرار معین، حلقه‌ای است که تعداد دفعات تکرار آن مشخص است و با مقادیر متغیری به نام شمارنده تعیین می‌شود؛ مانند حلقه‌ی For...Next در زبان ویژوال بیسیک. **حلقه‌ی تکرار نامعین**، حلقه‌ای است که تعداد دفعات تکرار آن مشخص نیست و بر اساس درستی یا نادرستی شرطی، حلقه ادامه می‌یابد؛ مانند حلقه‌ی Do...Loop در زبان ویژوال بیسیک.

همانطور که قبلاً نیز مشاهده کردید، می‌توان بدون استفاده از کادر متن با کاربر ارتباط برقرار کرد و اطلاعاتی را از او گرفت. برای این قبیل اعمال، باید از حلقه استفاده کنید چون هیچ اطمینانی وجود ندارد که کاربر اطلاعات صحیح وارد کند. مثلاً، فرض کنید از یک کاربر سن وی را پرسیده‌اید و او جواب داده است: ۲۹۱! در اینجا واضح است که کاربر اشتباه کرده (یا قصد شوخی داشته) است. با استفاده از حلقه می‌توان تا زمانی که کاربر اطلاعات صحیح نداده است، پردازش را متوقف کرد (البته در مثال فوق، برنامه از قبل نمی‌تواند سن کاربر را حدس بزند ولی حداقل حدود منطقی آن را می‌داند!).

۴-۱ حلقه‌ی تکرار For...Next

برای اجرای دستورهای مشخصی از برنامه به دفعات معین، از ساختار حلقه‌ی For...Next کمک بگیرید. هنگامی حلقه‌ی تکرار For...Next را به کار می‌بریم که تعداد دفعات تکرار مشخص باشد.

شکل کلی این دستور به صورت زیر است:

For intCounter = intStart To intEnd [Step intIncrement]

مجموعه دستورها

Next [intCounter]

intCounter یک متغیر عددی است که شمارنده‌ی حلقه نامیده می‌شود. ویژوال بیسیک قبل از شروع حلقه، intCounter را با intStart مقداردهی می‌کند. مقدار intStart معمولاً ۱ است ولی می‌تواند هر عدد (یا متغیر) دیگری هم باشد. با هر تکرار حلقه، مقدار intIncrement به intCounter اضافه می‌شود. اگر قسمت Step حذف شود، حلقه‌ی For مقدار ۱ را برای intIncrement فرض خواهد کرد. عدد intEnd پایان حلقه را کنترل می‌کند. هرگاه intCounter از intEnd بیشتر شود، ویژوال بیسیک دیگر حلقه را تکرار نمی‌کند و دستورهایی بعد از Next اجرا می‌شوند. Next در واقع نقطه‌ی انتهایی حلقه است. یادآوری: تمام قسمت‌هایی که داخل [] نوشته شده‌اند اختیاری هستند، یعنی ویژوال بیسیک بدون آنها هم می‌تواند کار خودش را انجام دهد.

مثال ۴-۱

برنامه‌ی زیر با فشار دادن دکمه‌ی Run، اعداد ۱ تا ۱۰ را روی فرم چاپ می‌کند:

۱- متغیر intI را از نوع اعداد صحیح در نظر بگیر

۲- $intI \rightarrow 1$

۳- مقدار intI را نمایش بده

۴- $intI + 1 \rightarrow intI$

۵- اگر $intI \leq 10$ بود، برو به مرحله‌ی ۳

```
Private Sub cmdRun_Click( )
```

```
Dim intI as Integer
```

```
For intI=1 To 10
```

```
Print intI
```

```
Next intI
```

```
End Sub
```

در این برنامه ابتدا متغیر intI با مقدار یک پر می‌شود، سپس بدنه‌ی حلقه اجرا شده و مقدار درون intI چاپ می‌شود. پس از رسیدن به دستور Next مقدار intI با مقدار نهایی یعنی ۱۰ مقایسه می‌شود و چون از ۱۰ کوچک‌تر است، یک واحد به intI اضافه شده و به ابتدای حلقه باز می‌گردد و مراحل قبل را تکرار می‌کند و این عمل را تا وقتی که intI از مقدار نهایی بزرگ‌تر نشده است، ادامه می‌دهد.

حلقه‌ی For اصلی‌ترین ابزار تکرار و بیژوال بیسیک است. توجه داشته باشید که اگر مقدار Step مثبت باشد، شمارش حلقه صعودی است ولی اگر مقدار Step منفی باشد، شمارش حلقه نزولی خواهد بود (در این حالت intStart باید از intEnd بزرگ‌تر باشد). تعداد دفعات تکرار این نوع حلقه با مقادیر اولیه و نهایی شمارنده‌ی حلقه و گام حرکت تعیین می‌شود. این مقادیر می‌توانند اعداد صحیح مثبت، منفی و اعشاری و یا حاصل یک عبارت محاسباتی باشند. اگر گام حرکت، برابر با عدد یک باشد، نیازی به نوشتن آن نیست و بیژوال بیسیک به‌طور پیش فرض، آن را یک مثبت در نظر خواهد گرفت. در صورتی که گام حرکت، عددی غیر از یک باشد، باید آن را تعیین کرد. اگر مقدار اولیه‌ی شمارنده از مقدار نهایی کوچک‌تر باشد، باید گام حرکت را مثبت در نظر گرفت و در غیر این صورت، گام حرکت منفی خواهد بود. شمارنده‌ی حلقه را می‌توان متغیری مثل intI، intCount و ... تعریف کرد.

مثال ۲-۴

قطعه برنامه‌ی زیر، ۱۰ تا ۱ را به صورت نزولی روی فرم چاپ می‌کند:

```
For intI=10 To 1 Step -1
  Print intI
Next intI
```

بین دو عبارت For و Next مجموعه‌ای از دستورها و عبارات نوشته می‌شوند که به عنوان بدنه‌ی حلقه شناسایی می‌گردند و به تعداد دفعات مشخص، اجرای آنها تکرار می‌شود. در زبان ویژوال بیسیک، می‌توان با بررسی شرط خاصی از حلقه‌ی تکرار For خارج شد و دستور بعد از Next را اجرا کرد. انجام این کار با استفاده از دستور Exit For صورت می‌گیرد که سبب خروج از حلقه‌ی تکرار خواهد شد. مثال‌های زیر، چگونگی تعریف و مقداردهی شمارنده‌ی حلقه را نشان می‌دهند:

```
For Count = 10 To 100 Step 4
```

```
For I = -10 To 45 Step 2
```

```
For J = 150 To 50 Step - 5.5
```

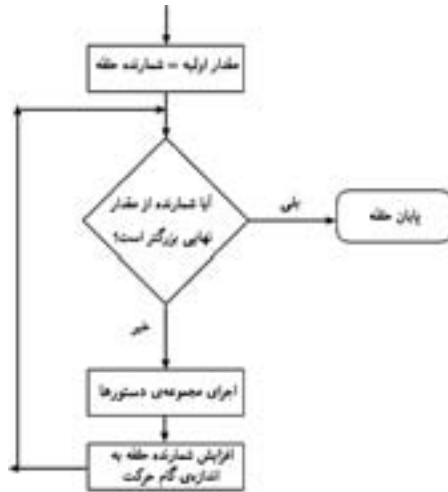
```
For K = A*2/6 To A+ 100 Step (A+K)/5
```

مثال ۳-۴

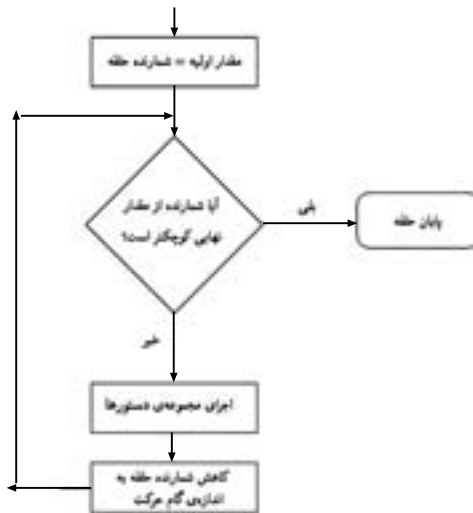
برنامه‌ای بنویسید که اولین عدد دو رقمی را که بر ۷ بخش پذیر است، نمایش دهد.

```
For intI=10 To 100
  if (I Mod 7) = 0 Then Exit For
Next intI
Label 1. caption = "The First Number is:" & i
```

نمودار گردش کار حلقه‌ی تکرار با گام حرکت مثبت و منفی در شکل‌های ۴-۱ و ۴-۲ نشان داده شده است:



شکل ۴-۱ نمودارگردشی حلقه‌ی تکرار For با گام حرکت مثبت



شکل ۴-۲ نمودارگردشی حلقه‌ی تکرار For با گام حرکت منفی

مثال ۴-۴

قطعه برنامه‌ی زیر، اعداد زوج ۲ تا ۳۰ را به صورت صعودی روی فرم چاپ می‌کند:

```
For intI = 2 To 30 Step 2
```

```
Print intI
```

```
Next
```

مثال ۵-۴

برنامه‌ای بنویسید که ۵ عدد را از ورودی دریافت کند و حاصل جمع و میانگین آنها را چاپ کند.

مراحل کار:

۱. پروژه‌ی جدیدی را شروع کنید.
۲. فرمی به صورت شکل ۳-۴ طراحی کنید و کد مربوط به دکمه‌ی Close آن را نیز بنویسید.



شکل ۳-۴

۳. روی دکمه‌ی Run دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub CmdRun_Click()
```

و

```
End Sub
```

دستورهای زیر را وارد کنید:

۱- متغیر `intI` را از نوع اعداد صحیح و متغیرهای `sngS` و `sngN` را از نوع اعداد اعشاری در نظر بگیرید

۲- `sngS → ۰`

۳- `intI → ۱`

۴- `InputBox ("Enter"& intI & "th Number:") → sngN`

۵- `sngS + sngN → sngS`

۶- `intI + ۱ → intI`

۷- اگر `intI ≤ ۵` بود، برو به مرحله‌ی ۴

۸- پیام `"Sum="` و مقدار `sngS` را نمایش بده

۹- پیام "Average=" و مقدار $sngS/5$ را نمایش بده

1. **Dim intI As Integer, sngS As Single, sngN As Single**
2. **sngS = 0**
3. **For intI = 1 To 5**
4. **sngN = InputBox ("Enter"& intI & "th Number:")**
5. **sngS = sngS + sngN**
6. **Next intI**
7. **Print "Sum=";sngS**
8. **Print "Average="; sngS/5**

۴. برنامه را اجرا کنید و روی دکمه‌ی Run کلیک کنید. ۵ عدد را یک به یک در پاسخ به کادر InputBox وارد کنید (به‌عنوان مثال ۹، ۸، ۷، ۶، ۵). در نتیجه روی فرم حاصل جمع و میانگین را به‌صورت شکل ۴-۴ مشاهده می‌کنید:



شکل ۴-۴

شرح کار برنامه:

در سطر ۱، ابتدا متغیر intI را به‌عنوان متغیر حلقه تعریف می‌کنیم تا از آن برای ۵ دور تکرار حلقه استفاده کنیم. سپس متغیر sngS را برای متغیر نتیجه‌ی جمع ۵ عدد و sngN را برای دریافت تک‌تک اعداد ورودی تعریف می‌کنیم.

در سطر ۲ متغیر sngS را برابر صفر قرار می‌دهیم زیرا لازم است قبل از ذخیره‌ی اعداد در آن، مقدار آن صفر باشد.

در سطر ۳، حلقه‌ی For را برای شمارش از ۱ تا ۵، یعنی ۵ دور تنظیم می‌کنیم.

در سطر ۴ به‌وسیله‌ی تابع InputBox اولین عدد را از ورودی گرفته و در متغیر sngN ذخیره می‌کنیم.

در سطر ۵ به‌وسیله‌ی رابطه‌ی $sngS = sngS + sngN$ مقدار دریافت شده را با $sngS$ جمع می‌کنیم.

نکته

همان‌طور که در فصل‌های قبل آموختیم، عملگر $=$ سبب می‌شود که ابتدا عبارت سمت راست محاسبه و سپس در عبارت سمت چپ ذخیره شود. در نتیجه، ابتدا مقدار جدید $sngN$ با مقدار قبلی $sngS$ جمع شده و سپس در همان $sngS$ ذخیره می‌شود که در این صورت، پس از چند دور حلقه، مقادیر یک به یک به $sngS$ اضافه می‌شود.

در سطر ۶، به شرط آنکه به انتها نرسیده باشیم، به متغیر حلقه یک واحد اضافه و کنترل اجرا به ابتدای حلقه منتقل می‌شود و مراحل قبل دوباره تکرار می‌گردد. در سطر ۷، یعنی پس از خروج از حلقه، مقدار $sngS$ که همان مجموع ۵ عدد است، چاپ می‌شود. در سطر ۸، نیز میانگین اعداد ورودی چاپ می‌شود.
 $\text{تعداد} / \text{مجموع اعداد} = \text{میانگین}$

مثال ۴-۶

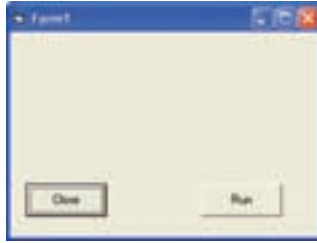
برنامه‌ی زیر، عددی را از ورودی می‌گیرد و مقسوم‌علیه‌های آن را چاپ می‌کند.

نکته

برای یافتن مقسوم‌علیه‌های یک عدد، باید آن را بر تمامی اعداد از ۱ تا نصف عدد تقسیم کرد و اگر باقیمانده صفر بود، یعنی بخش‌پذیر است و آن عدد را به‌عنوان مقسوم‌علیه چاپ می‌کنیم.

مراحل کار:

۱. پروژه‌ی جدیدی را شروع کنید.
۲. فرمی به‌صورت شکل ۴-۵ طراحی کنید و کد مربوط به دکمه‌ی Close آن را نیز بنویسید.



شکل ۴-۵

۳. روی دکمه‌ی Run دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub cmdRun_Click()
```

و

```
End Sub
```

دستورهای زیر را وارد کنید:

۱- متغیرهای $intI$ و $intN$ را از نوع اعداد صحیح در نظر بگیر

۲- صفحه‌ی نمایش خروجی را پاک کن

۳- مقداری را از ورودی دریافت (از طریق کادر ورودی) کن و در $intN$ قرار بده

۴- اگر $intN \leq 0$ بود، پیغامی مبنی بر اینکه عدد ورودی باید بزرگ‌تر از صفر باشد، نمایش بده و از روال خارج شو

۵- $intI \rightarrow 1$

۶- اگر باقیمانده‌ی تقسیم $intN$ بر $intI$ مساوی صفر بود (بخش‌پذیر بود)، مقدار

$intI$ را نمایش بده

۷- $intI + 1 \rightarrow intI$

۸- اگر $intI \leq intN/2$ بود، برو به مرحله‌ی ۶

```
Dim intI As Integer, intN As Integer
```

```
Cls
```

```
intN = InputBox("Enter an Integer Number:")
```

```
If intN <= 0 Then
```

```
    MsgBox "The number must be greater than 0", vbOKOnly_
```

```
    + vbCritical
```



```
Exit Sub
```

```
End If
```

```
For intI = 1 To intN\2
```

```
    If (intN Mod intI = 0) Then Print intI;
```

```
Next intI
```

۴. برنامه را اجرا کنید و دکمه Run را فشار دهید و در پاسخ به کادر InputBox عددی را وارد کنید (به عنوان مثال، ۱۰۲۴). در نتیجه روی فرم، مقسوم‌علیه‌های این عدد را مشاهده می‌کنید (شکل ۴-۶).



شکل ۴-۶

شرح برنامه:

ابتدا پس از تعریف متغیرها به وسیله‌ی تابع InputBox عددی را دریافت می‌کنیم. سپس آن عدد را با صفر مقایسه کرده و در صورت صفر بودن، یا کوچک‌تر از صفر بودن، پیام خطایی را چاپ کرده و از رویداد خارج می‌شویم. در بخش دوم، حلقه‌ای ایجاد کرده و محدوده‌ی آن را از ۱ تا نصف مقدار ورودی یعنی $\text{intN} \setminus 2$ مقداردهی می‌کنیم. در هر دور اجرای حلقه، مقدار ورودی یعنی intN را بر متغیر حلقه یعنی intI تقسیم و باقیمانده آن را به دست می‌آوریم. در سطر بعد، به وسیله‌ی دستور شرطی If باقیمانده را بررسی و به شرط صفر بودن، متغیر intI یا همان مقسوم‌علیه را چاپ می‌کنیم.

تمرین

۱. برنامه‌ی قبل را به گونه‌ای تغییر دهید که علاوه بر چاپ مقسوم‌علیه‌ها، تعداد آنها را نیز چاپ کند.
۲. برنامه‌ی قبل را به گونه‌ای تغییر دهید که مجموع مقسوم‌علیه‌ها را چاپ کند.

۴-۱-۱ نکاتی درباره‌ی حلقه‌ی FOR

در شکل ۴-۱، در هر بار اجرای دستورها، به اندازه‌ی گام حرکت، به مقدار جاری شمارنده اضافه می‌شود. در این حالت، اگر مقدار اولیه‌ی شمارنده از مقدار نهایی بزرگ‌تر باشد، حلقه‌ی تکرار اصلاً اجرا نخواهد شد.

در شکل ۴-۲، در هر بار اجرای دستورها به اندازه‌ی گام حرکت از مقدار جاری شمارنده کاسته می‌شود. در این حالت، اگر مقدار اولیه‌ی شمارنده از مقدار نهایی کوچک‌تر باشد، حلقه‌ی تکرار اصلاً اجرا نخواهد شد.

مقدار شمارنده‌ی حلقه، در بدنه‌ی حلقه معمولاً از سوی برنامه‌نویس تغییر نمی‌کند ولی برنامه‌نویس می‌تواند آن را تغییر دهد.

اگر مقدار نهایی شمارنده، با یک عبارت محاسباتی یا متغیری تعیین شده است، بدان معنی است که عدد ثابتی نیست ولی اگر این متغیر در بدنه‌ی حلقه تغییر کند، در تعداد دفعات تکرار تأثیری نخواهد داشت، زیرا مقدار نهایی شمارنده‌ی حلقه، فقط یک بار و آن هم هنگام شروع حلقه خوانده شده و در متغیر درونی دستور For ذخیره می‌شود.

پس از خارج شدن از حلقه‌ی تکرار، مقدار شمارنده، آخرین مقداری خواهد بود که در طول اجرای حلقه به خود اختصاص داده است و همواره بسته به صعودی یا نزولی بودن حلقه، مقدار شمارنده یک گام کمتر یا بیشتر از مقدار نهایی است.

۴-۲ حلقه‌ی Do

این حلقه مانند If چند شکل مختلف دارد:

1) Do while condition

یک یا چند دستور

Loop

2) Do

یک یا چند دستور

Loop While condition

3) Do until condition

یک یا چند دستور

Loop

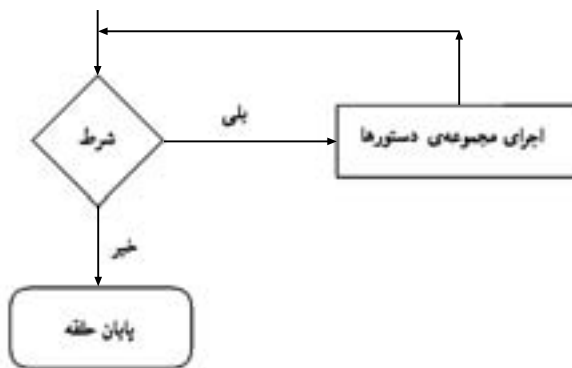
4) Do

یک یا چند دستور

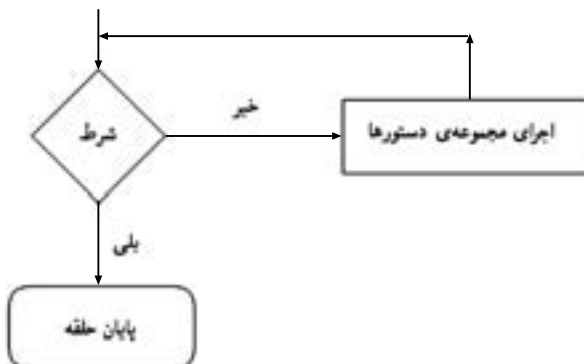
Loop Until condition

شرط condition در حلقه‌ی Do می‌تواند هر عبارت یا متغیر منطقی باشد که True یا False برمی‌گرداند. نوع حلقه‌ی Do به محل کاربرد آن بستگی دارد. تفاوت انواع حلقه‌های Do از این قرار است:

- **محل بررسی شرط حلقه:** اگر شرط حلقه در ابتدای آن بررسی شود، ممکن است که حلقه هرگز اجرا نشود. اما اگر بررسی حلقه در انتهای آن صورت گیرد، حلقه حداقل یک بار اجرا می‌شود، چون ویژوال بیسیک فقط بعد از اجرای حلقه است که شرط را بررسی خواهد کرد.
 - **خصلت بررسی حلقه:** حلقه‌هایی که While دارند، تا زمانی که شرط درست است اجرا می‌شوند، ولی در حلقه‌های Until، اگر شرط درست شود، اجرای حلقه پایان خواهد یافت.
- در شکل‌های ۴-۷ و ۴-۸، حلقه‌ی Do و طرز اجرای آن را مشاهده می‌کنید.



شکل ۴-۷ حلقه‌ی Do While/Loop



شکل ۴-۸ حلقه‌ی Do Until/Loop

تمرین

نمودار گردش کار حلقه‌های Do/Loop While و Do/Loop Until را رسم کنید.

بدنه‌ی حلقه‌ی برنامه‌ی ۱-۴، ده بار تکرار خواهد شد و هر بار متغیر intCtr یکی اضافه می‌شود. در این حلقه از Do ... Loop Until استفاده کرده‌ایم، بنابراین حلقه تا رسیدن intCtr به ۱۰ تکرار می‌شود. در قطعه برنامه‌ی ۱-۴ همین کار را با حلقه‌ی Do While ... Loop انجام داده‌ایم.

برنامه‌ی ۱-۴ از انواع حلقه‌های Do استفاده کنید.

1. **Do While** intCtr <= 10
2. **print** intCtr
3. intCtr = intCtr + 1 افزایش شمارنده به اندازه‌ی یک واحد
4. **Loop**

البته در این برنامه چون مقدار جدید intCtr درون برچسب قرار می‌گیرد، مقدار قبلی برچسب پاک شده و در نهایت فقط مقدار نهایی مشاهده می‌شود. یکی از نکات اساسی در حلقه‌ها این است که مقدار شرط حلقه باید در بدنه‌ی حلقه تغییر داده شود چون اگر چنین نشود، حلقه تا ابد ادامه خواهد یافت.

مثال ۷-۴

قطعه برنامه‌ی زیر، اعداد فرد ۱ تا ۲۰ را به صورت صعودی روی فرم چاپ می‌کند:

۱ → intI

۲- مقدار intI را نمایش بده

۳- intI + ۲ → intI

۴- اگر $intI \leq 20$ بود، برو به مرحله‌ی ۲

intI = 1

Do While intI <= 20

Print intI,

intI = intI + 2

Loop

مثال ۸-۴

قطعه برنامه‌ی زیر، مجموع اعداد ۱ تا ۱۰۰ را به دست آورده و روی فرم چاپ می‌کند:

۱- متغیرهای `intI` و `intS` را از نوع اعداد صحیح در نظر بگیر

۲- صفحه‌ی خروجی را پاک کن

۳- `intI → ۱`

۴- `intS → ۰`

۵- `intS + intI → intS`

۶- `intI + ۱ → intI`

۷- اگر `intI ≤ ۱۰۰` بود، برو به مرحله‌ی ۵

۸- `intS` را نمایش بده

```
Dim intI As Integer, intS As Integer
```

```
cls
```

```
intI = 1
```

```
intS = 0
```

```
Do While intI <= 100
```

```
    intS = intS + intI
```

```
    intI = intI + 1
```

```
Loop
```

```
Print intS
```

حلقه‌هایی که با گرفتن اطلاعات از کاربر سر و کار دارند، باید حداقل یک بار اجرا شوند، بنابراین بررسی این‌گونه حلقه‌ها باید در انتهای حلقه انجام شود. در برنامه‌ی ۲-۴ نمونه‌ای از این قبیل حلقه‌ها را مشاهده می‌کنید.

برنامه‌ی ۲-۴ وارد کردن داده‌های صحیح گاهی به چند بار تکرار نیاز دارد.

1. `Dim strAns As String`
2. `lblPrompt.Caption = "Do you want to continue (yes or no)?"`
3. `Do While (strAns <> "Yes" And strAns <> "No")`
4. `Beep` *'Warning'*

5. lblError.Caption = "You need to answer Yes or No"

6. Loop

7. lblError.Caption = ""

در مثال فوق، قسمت واقعی کد، یعنی گرفتن اطلاعات از کاربر نوشته نشده است، و فقط روی ساختار حلقه متمرکز شده است. حلقه‌ی Do در این مثال از خط ۹ شروع می‌شود. اگر کاربر کلمات Yes و No وارد نکرده باشد، بدنه‌ی حلقه این مطلب را به او تذکر خواهد داد و عملیات گرفتن پاسخ کاربر تکرار خواهد شد (خط ۱۳).

این حلقه یا هرگز تکرار نخواهد شد (اگر کاربر پاسخ Yes یا No داده باشد)، یا برای همیشه تکرار خواهد شد (تا زمانی که کاربر چنین پاسخی نداده باشد). توجه کنید که اگر کلید Caps Lock صفحه کلیدی که کاربر با آن کار می‌کند، روشن یا خاموش باشد، پاسخ وی متفاوت خواهد بود (yes یا YES ؛ no یا No). از آنجایی که ویژگی بیسیک این رشته‌ها را یکسان تلقی نمی‌کند، بررسی حلقه می‌تواند هرگز اعمال نشود، حتی اگر کاربر به خیال خود پاسخ صحیح داده باشد.

اگر تحت شرایطی بخواهید حلقه‌ی Do دیگر ادامه پیدا نکند، می‌توانید از دستور Exit Do استفاده کنید.

مثال ۹-۴

برنامه‌ای بنویسید که تعدادی عدد مثبت را از ورودی دریافت کرده و حاصل جمع و میانگین آنها را چاپ کند.

نکته

تعداد اعداد ورودی معلوم نیست و شرط توقف، ورود عدد صفر یا کوچک‌تر از صفر است.

مراحل کار:

۱. پروژه‌ی جدیدی را شروع کنید.
۲. فرمی به صورت شکل ۹-۴ طراحی کنید و کد مربوط به دکمه‌ی Close آن را نیز بنویسید.



شکل ۹-۴

۳. روی دکمه‌ی Run دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub cmdRun_Click()
```

و

```
End Sub
```

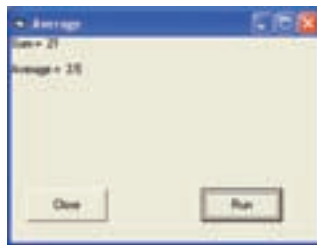
دستورهای زیر را وارد کنید:

- ۱- متغیرهای sngN و sngS را از نوع اعداد اعشاری در نظر بگیرید
- ۲- متغیر intCnt را از نوع اعداد صحیح در نظر بگیرید
- ۳- $sngS \rightarrow 0$ و $intCnt \rightarrow 0$
- ۴- مقداری را از طریق کادر ورودی دریافت کن و در sngN قرار بده
- ۵- $sngS + sngN \rightarrow sngS$
- ۶- $intCnt + 1 \rightarrow intCnt$
- ۷- مقداری را از طریق کادر ورودی دریافت کن و در sngN قرار بده
- ۸- اگر sngN بزرگ‌تر از صفر بود، برو به مرحله‌ی ۵
- ۹- صفحه‌ی خروجی را پاک کن
- ۱۰- پیغام $Sum =$ و مقدار sngS را نمایش بده
- ۱۱- یک خط خالی رد کن
- ۱۲- پیغام $Average =$ و مقدار $sngS/intCnt$ را نمایش بده

1. **Dim sngN As Single, sngS As Single**
2. **Dim intCnt As Integer**
3. $sngS = 0$
4. $intCnt = 0$
5. $sngN = \text{InputBox}(\text{"either Enter a Number or 0 to End:"}, \text{" "})$

6. **Do While** sngN > 0
7. sngS = sngS + sngN
8. intCnt = intCnt + 1
9. sngN = **InputBox**("either Enter a Number or 0 to End:",
"")
10. **Loop**
11. **cls**
12. **Print** "Sum = "; sngS
13. **Print**
14. **Print** "Average = "; sngS / intCnt

۴. برنامه را اجرا کنید و روی دکمه‌ی Run کلیک کنید و در پاسخ به کادر InputBox، تعدادی عدد را یک به یک وارد کنید (به‌عنوان مثال ۱، ۲، ۳، ۴، ۵، ۶). در پایان، عدد صفر را برای خاتمه‌ی حلقه وارد کنید. روی فرم، حاصل جمع و میانگین را به‌صورت شکل ۴-۱۰ مشاهده می‌کنید:



شکل ۴-۱۰

شرح کار برنامه:

در سطر ۱ و ۲، متغیر intCnt را به‌عنوان شمارنده‌ی تعداد اعداد ورودی، متغیر sngS را برای ذخیره‌ی نتیجه‌ی جمع اعداد و sngN را برای دریافت اعداد ورودی تعریف می‌کنیم.

در سطر ۳ و ۴، sngS و intCnt را مساوی صفر قرار می‌دهیم تا هیچ مقدار اولیه‌ای در شروع کار نداشته باشند.

در سطر ۵، به وسیله‌ی تابع InputBox اولین عدد را از ورودی دریافت کرده و در متغیر sngN ذخیره می‌کنیم.

در سطر ۶، به شرط آن‌که عدد ورودی از صفر بزرگ‌تر باشد، اجازه‌ی ورود به حلقه داده می‌شود.

در سطر ۷، به وسیله‌ی رابطه‌ی $sngS = sngS + sngN$ مقدار دریافت شده را با $sngS$ جمع می‌کنیم.

در سطر ۸، به متغیر `intCnt` یک واحد اضافه می‌کنیم تا تعداد اعداد ورودی را برای ما نگه دارد.

در سطر ۹، دوباره به وسیله‌ی تابع `InputBox` عدد بعدی را از ورودی گرفته و در متغیر `sngN` ذخیره می‌کنیم تا در برگشت مجدد به ابتدای حلقه، این مقدار جدید با صفر مقایسه شود و در صورت بزرگ‌تر بودن از صفر، دوباره وارد حلقه شده و مراحل قبل تکرار شود. در صورتی که عدد صفر یا کوچک‌تر وارد شود، حلقه خاتمه یافته و کنترل اجرا به سطر بعدی یعنی ۱۱ منتقل می‌شود.

در سطر ۱۱، ابتدا صفحه‌ی فرم پاک شده و در سطر ۱۲ و ۱۴، حاصل جمع و میانگین چاپ می‌شود.

سطر ۱۳ که دارای دستور `Print` است، در عمل فقط یک سطر خالی بین دو دستور چاپ مجموع و میانگین ایجاد می‌کند.

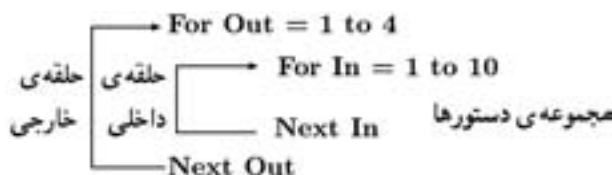
تمرین

۱. برنامه‌ی چاپ مقسوم‌علیه‌ها را با حلقه‌ی `Do` بازنویسی کنید.
۲. برنامه‌ای بنویسید که عددی صحیح و مثبت را از ورودی دریافت کرده و تعداد ارقام آن را چاپ کند.

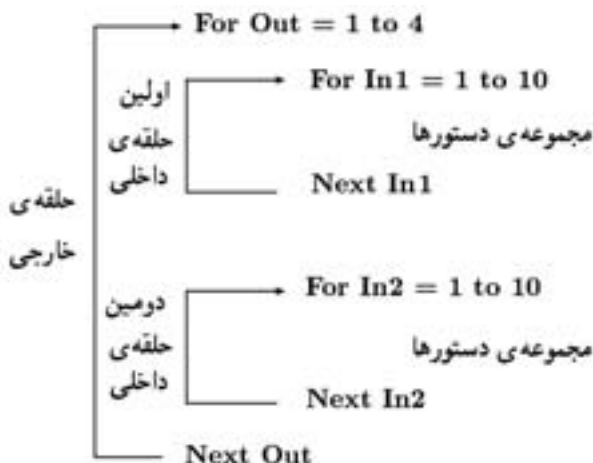
۴-۳ حلقه‌های متداخل

مانند دستور `If`، حلقه‌های `For` را هم می‌توان داخل یکدیگر قرار داد. شکل ۱۱-۴ یک حلقه‌ی `For` نودرتو را نشان می‌دهد. حلقه‌ای که داخلی‌تر است، معمولاً بیشتر اجرا می‌شود، زیرا با هر اجرای کامل حلقه‌ی داخلی (افزایش `In` از ۱ تا ۱۰) حلقه‌ی بیرونی فقط یک بار اجرا می‌شود. علت آن است که تا حلقه‌ی داخلی کامل نشود، نوبت به `Next` بیرونی‌تر نخواهد رسید.

در شکل ۴-۱۱، حلقه‌ی داخلی ۴۰ بار اجرا خواهد شد (۴ بار تکرار حلقه‌ی بیرونی، ضربدر ۱۰ بار تکرار حلقه‌ی داخلی). در شکل ۴-۱۲ دو حلقه را درون یک حلقه مشاهده می‌کنید. در هر تکرار حلقه‌ی بیرونی، هر یک از دو حلقه‌ی داخلی یک بار به طور کامل اجرا خواهند شد.



شکل ۴-۱۱ حلقه‌ی بیرونی تعداد اجراهای حلقه‌ی داخلی را تعیین می‌کند.



شکل ۴-۱۲ دو حلقه در یک حلقه

هنگام متداخل کردن حلقه‌ها به تطابق Nextها توجه کنید که در آن هر Next با نزدیکترین For قبل از خود متناظر است. اگر متغیر حلقه را بعد از Next ننویسید، خود ویزوال بیسیک وظیفه‌ی انطباق حلقه‌ها را برعهده خواهد گرفت. اما اگر تصمیم گرفتید خودتان این وظیفه را برعهده بگیرید و احياناً اشتباه کردید، ویزوال بیسیک بلافاصله هشدار خواهد داد!

مثال ۴-۱۰

برنامه‌ی زیر، جدول ضرب اعداد ۱ تا ۵ را روی فرم چاپ می‌کند.

مراحل کار:

۱. پروژه‌ی جدیدی را شروع کنید.

۲. فرمی به صورت شکل ۱۳-۴ طراحی کنید و کد مربوط به دکمه‌ی Close آن را نیز بنویسید.



شکل ۱۳-۴

۳. روی دکمه‌ی Run دوبار کلیک کرده و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub CmdRun_Click()
```

و

```
End Sub
```

دستورهای زیر را وارد کنید:

۱- متغیرهای $intI$ و $intJ$ را از نوع اعداد صحیح در نظر بگیر

۲- $intI \rightarrow 1$

۳- $intJ \rightarrow 1$

۴- حاصل $intI \times intJ$ را نمایش بده

۵- $intJ \rightarrow intJ + 1$

۶- اگر $intJ \leq 5$ بود برو به مرحله‌ی ۴

۷- یک خط خالی رد کن (نمایش بده)

۸- $intI \rightarrow intI + 1$

۹- اگر $intI \leq 5$ بود برو به مرحله‌ی ۳

```
Dim intI As Integer, intJ As Integer
```

```
For intI = 1 To 5
```

```
    For intJ = 1 To 5
```

```
        Print intI * intJ,
```

```
    Next
```

```
    Print 'print blank line
```

```
Next
```

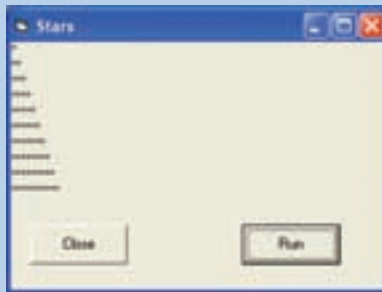
۴. برنامه را اجرا کنید و روی دکمه Run کلیک کنید. خروجی مانند شکل ۴-۱۴ خواهد بود.



شکل ۴-۱۴

تمرین

۱. برنامه‌ی قبل را به گونه‌ای تغییر دهید که جدول ضرب 10×10 را چاپ کند.
۲. برنامه‌ای بنویسید که شکل ۴-۱۵ را با کلیک کردن روی دکمه‌ی Run بر روی فرم نمایش دهد.
توجه: ستاره‌ها در 10 سطر چاپ شوند.



شکل ۴-۱۵

۴-۴ توابع

تاکنون با تعدادی از توابع داخلی ویژوال بیسیک آشنا شده‌اید. ویژوال بیسیک دارای ده‌ها تابع داخلی دیگر است که در ادامه‌ی این کتاب و درس‌های برنامه‌سازی ۲ و ۳ با آنها آشنا خواهید شد. توابعی که با آنها می‌توانید بر قدرت برنامه‌هایتان بیفزایید.

توابع انواع متعددی دارند که هر کدام کاربرد خاص خود را خواهند داشت:

۱. توابع ریاضی، ۲. توابع رشته‌ای، ۳. توابع تبدیلی، ۴. توابع تاریخ و زمان.

با توابع ریاضی در این فصل و توابع رشته‌ای و تبدیلی در فصل بعد آشنا خواهید شد.

بررسی توابع تاریخ و زمان در کتاب برنامه‌سازی ۲ مطرح خواهد شد.

۱-۴-۴ توابع ریاضی

ساده‌ترین توابع ریاضی، توابع تبدیل اعداد صحیح هستند. در زیر دو تا از رایج‌ترین توابع را مشاهده می‌کنید:

Int(numeric Value)

Fix(numeric Value)

که در آن numericValue، عدد، متغیر، عبارت یا تابعی است که عددی را برگرداند. هر دو تابع، همان نوعی که به آنها داده شده را برمی‌گردانند، ولی از این مقدار برگشتی می‌توان به‌عنوان یک عدد صحیح استفاده کرد. اگر آرگومان داده شده به این توابع عدد نباشد، ویژوال بیسیک خطا تولید خواهد کرد.

هر دو تابع فوق، **قسمت صحیح آرگومان مثبت خود را برمی‌گردانند**. تفاوت این دو تابع در

نحوه رفتار آنها با اعداد منفی است. در مثال‌های زیر تفاوت آنها را مشاهده می‌کنید:

intAns1 = Int (6.8) ' 6

intAns2 = Fix (6.8) ' 6

intAns3 = Int (-6.8) ' -7

intAns4 = Fix (-6.8) ' -6

دقت کنید که هیچ‌یک از دو تابع Int() یا Fix() اعداد مثبت را به بالا گرد نمی‌کنند.

تابع Int() آرگومان خود را به نزدیک‌ترین عدد صحیح کوچک‌تر گرد می‌کند (یعنی این تابع در مورد اعداد منفی، یک عدد کوچک‌تر برمی‌گرداند). تابع Fix() فقط قسمت صحیح را برمی‌گرداند و قسمت اعشاری را نادیده می‌گیرد.

تابع Abs(): یکی دیگر از توابع داخلی ویژوال بیسیک است که **قدرمطلق یک عدد** را برمی‌گرداند. تابع قدرمطلق برای محاسبه‌ی فاصله‌ی دو مقدار مناسب است؛ مثلاً فرض کنید

می‌خواهید تفاوت سن دو تن از دوستان خود را محاسبه کنید:

```
intAgeDiff = Abs (intEmpAge1 - intEmpAge2)
```

در این جا دیگر لازم نیست نگران باشید که کدام دوستان مسن تر است، زیرا این عبارت، همیشه یک عدد مثبت یا صفر برمی‌گرداند. اگر از Abs() استفاده نکنید، در مواردی ممکن است این عبارت منفی شود.

تابع Sqr(): جذر یک عدد مثبت را برمی‌گرداند. به مثال‌های زیر (و پاسخ آنها) توجه کنید:

```
intVal1 = Sqr (4) '2
```

```
intVal2 = Sqr (64) '8
```

```
intVal3 = Sqr (4096) '64
```

تابع COS(): این تابع، کسینوس زاویه‌ای برحسب رادیان را محاسبه می‌کند و برمی‌گرداند: (زاویه بر حسب رادیان) COS

مثال ۴-۱۱

برنامه‌ای بنویسید که کسینوس زوایای ۰، ۴۵، ۹۰، ۱۳۵، ۱۸۰، ۲۲۵، ۲۷۰، ۳۱۵ و ۳۶۰ درجه را محاسبه کند و نمایش دهد:

```
FOR I=0 TO 360 STEP 45
  RAD =I * 3.14/180
  PRINT COS(RAD),
NEXT I
```

راهنمایی

$$\text{زاویه برحسب درجه} \times \frac{\pi}{180} = \text{زاویه برحسب رادیان}$$

$$\pi = 3,141593$$

تابع SIN(): این تابع سینوس زاویه‌ای برحسب رادیان را محاسبه می‌کند و برمی‌گرداند:

SIN (زاویه برحسب رادیان)

مثال ۴-۱۲

برنامه‌ای بنویسید که زاویه‌هایی را پیدا کند که در آن، مقدار سینوس و کسینوس با هم برابرند و این زاویه‌ها را در صورت وجود، نمایش دهد:

۱- Angle \rightarrow °

۲- $\sin(\text{Angle} \times 3,1415/180) \rightarrow x$

۳- اگر $y = \cos(\text{Angle} \times 3,1415/180)$

۴- اگر $x = y$ بود، پیغام $x = y$ و مقدار Angle را نمایش بده

۵- Angle \rightarrow Angle + ۱

۶- اگر Angle ≤ 360 بود، برو به مرحله‌ی ۲

For Angle = 0 To 360

x = sin(Angle*3.1415/180)

y = cos(Angle*3.1415/180)

If x = y Then

Print "x=y=" ;Angle

End If

Next Angle

تابع TAN(): این تابع، تانژانت زاویه‌ای برحسب رادیان را تعیین کرده و برمی‌گرداند:

TAN (زاویه برحسب رادیان)

مثال ۴-۱۳

برنامه‌ای بنویسید که ارتفاع یک شیء را با در اختیار داشتن فاصله و زاویه‌ی دید، محاسبه کند و نمایش دهد:

$$\text{tag}(\text{زاویه}) = \frac{\text{ارتفاع}}{\text{فاصله}}$$

Dim Dis, Ang As Integer

Dim Height As Single



```

Dis = InputBox ("Distance=")
Ang = InputBox("Angle=")
Height = Dis * TAN(Ang* 3.1415/180)
PRINT "Height = "; Height

```

تابع $\text{SGN}()$: این تابع، علامت عددی را که به عنوان آرگومان می‌پذیرد، تعیین می‌کند: (عبارت عددی) SGN

اگر حاصل عبارت یا متغیر، عددی بزرگ‌تر از صفر باشد، تابع مقدار ۱ را برمی‌گرداند.
اگر حاصل عبارت یا متغیر، مساوی صفر باشد، تابع مقدار ۰ را برمی‌گرداند.
اگر حاصل عبارت یا متغیر، کوچک‌تر از صفر باشد، تابع مقدار -۱ را برمی‌گرداند.
ویژوال بیسیک چند تابع دیگر نیز دارد که عبارت‌اند از:

- $\text{Rnd}()$: یک عدد تصادفی تولید می‌کند. (بین صفر و یک)
- $\text{Log}()$: از آرگومان خود، لگاریتم طبیعی می‌گیرد.

نکته

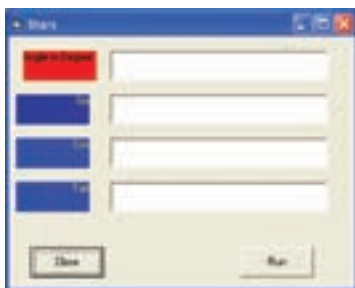
برای تولید عدد تصادفی بین صفر و n از رابطه $\text{Rnd}() * n$ استفاده می‌کنیم.

مثال ۴-۱۴

برنامه‌ی زیر، عددی را به عنوان مقدار یک زاویه برحسب درجه دریافت کرده و سینوس، کسینوس، تانژانت و کتانژانت آن زاویه را در کادرهای متن نمایش می‌دهد.

مراحل کار:

۱. برنامه‌ی ویژوال بیسیک را اجرا کرده و پروژه‌ی جدیدی را ایجاد کنید.
۲. فرمی به صورت شکل ۴-۱۶ طراحی کنید و کد مربوط به دکمه‌ی Close آن را نیز بنویسید.



شکل ۱۶-۴

۳. روی فرم دوبار کلیک کنید و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub Form_Load()
```

و

```
End Sub
```

دستورهای زیر را وارد کنید:

```
Text1 = ""
```

```
Text2 = ""
```

```
Text3 = ""
```

```
Text4 = ""
```

۴. روی دکمه‌ی Run دوبار کلیک کنید و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub CmdRun_Click()
```

و

```
End Sub
```

دستورهای زیر را وارد کنید:

1. **Dim** dblDegree **As Double**, dblRadian **As Double**

2. **Const** Pi **As Double** = 3.14159265358979

3. dblDegree = **Val**(Text1)

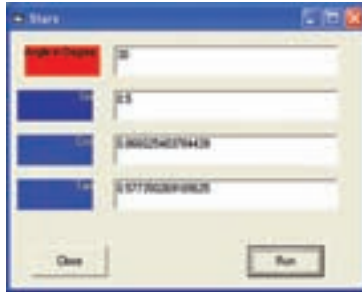
4. dblRadian = Pi * dblDegree / 180

5. Text2 = **Sin**(dblRadian)

6. Text3 = **Cos**(dblRadian)

7. Text4 = **Tan**(dblRadian)

۵. برنامه را اجرا کنید و بعد از وارد کردن اندازه‌ی زاویه برحسب درجه در اولین کادر متن (به‌عنوان مثال، عدد ۳۰)، روی دکمه‌ی Run کلیک کنید. خروجی به‌صورت شکل ۴-۱۷ خواهد بود.



شکل ۴-۱۷

خلاصه‌ی فصل

در زبان‌های برنامه‌نویسی برای تکرار تعدادی از دستورها از حلقه‌های تکرار استفاده می‌شود. حلقه‌ها به دو دسته تقسیم می‌شوند: ۱. حلقه‌های تکرار معین که تعداد دفعات تکرار در آنها مشخص است؛ ۲. حلقه‌های تکرار نامعین که تعداد دفعات تکرار در آنها مشخص نیست و بر اساس درستی یا نادرستی شرط، تکرار یا قطع می‌شوند.

حلقه‌ی For از نوع حلقه‌ی تکرار معین است.

حلقه‌ی Do از نوع حلقه‌های تکرار نامعین است.

حلقه‌ها را می‌توان به صورت متداخل به‌کار برد.

توابع داخلی و یژوال بیسیک، دارای انواع مختلفی هستند که می‌توان در برنامه‌ها به‌کار برد.

توابع Int و Fix: تبدیل اعداد به نوع صحیح

تابع Abs: قدرمطلق عدد

تابع Sqr: جذر یک عدد مثبت را برمی‌گرداند.

توابع مثلثاتی Sin، Cos و Tan: به ترتیب سینوس، کسینوس و تانژانت یک زاویه برحسب

رادیان را محاسبه و برمی‌گردانند.

تابع Sgn: علامت یک عدد را برمی‌گرداند.

تابع Rnd: یک عدد تصادفی تولید می‌کند.

تابع Log: لگاریتم یک عدد را محاسبه و برمی‌گرداند.

خودآزمایی

۱. برنامه‌ای بنویسید که مجموع مربعات اعداد ۱ تا ۱۰ را محاسبه و چاپ کند.
۲. برنامه‌ای بنویسید که تعدادی عدد صحیح از ورودی بگیرد و تعداد اعداد زوج و فرد را جداگانه چاپ کند (شرط توقف، ورود صفر است).
۳. برنامه‌ای بنویسید که عدد N را بگیرد و حاصل عبارت زیر را چاپ کند.

$$F = 1 * 2 * 3 * \dots * N$$
۴. برنامه‌ای بنویسید که عددی را از ورودی دریافت کرده و تعیین کند که آیا عدد اول است یا خیر؟ (راهنمایی: هر عدد اول دارای دو مقسوم‌علیه است یا مجموع مقسوم‌علیه‌های آن، یک واحد بیشتر از خود عدد است).
 توضیح: عدد اول، عددی است که به غیر از یک و خودش بر هیچ عدد دیگری بخش پذیر نباشد.
۵. برنامه‌ای بنویسید که اعداد اول ۱ تا ۱۰۰ را چاپ کند.
۶. برنامه‌ای بنویسید که عددی صحیح از ورودی دریافت کرده و مجموع ارقام آن را چاپ کند.
۷. برنامه‌ای بنویسید که عددی صحیح از ورودی دریافت کرده و مقلوب آن را چاپ کند.
 $(۲۳ \rightarrow ۳۲)$
۸. برنامه‌ای بنویسید که تمام اعداد صحیح سه رقمی را که رقم یکان آنها با صدگان برابر است، چاپ کند (۱۱۱ - ۱۲۱ - ۱۳۱ - ۲۴۲، ...).
۹. برنامه‌ای بنویسید که خروجی مقابل را تولید کند:

1

1 2

1 2 3

1 2 3 4

۱۰. برنامه‌ای بنویسید که خروجی مقابل را تولید کند:
- ```

1
2 2
3 3 3
4 4 4 4

```
۱۱. برنامه‌ای بنویسید که خروجی مقابل را تولید کند:
- ```

1 2 3 4 5 6 7
  1 2 3 4 5
    1 2 3
      1

```
- راهنمایی: از تابع (n) SPC در متد PRINT برای ایجاد فاصله‌ها استفاده کنید. n تعداد فضاهای خالی است.
۱۲. برنامه‌ای بنویسید که حاصل عبارت زیر را تا ۱۰ جمله حساب کند:
- $$1 + \frac{1}{4} + \frac{2}{9} + \frac{3}{16} + \dots$$
۱۳. برنامه‌ای بنویسید که دو عدد صحیح A و B را از ورودی بگیرد و بزرگ‌ترین مقسوم‌علیه مشترک و کوچک‌ترین مضرب مشترک بین آن دو عدد را محاسبه و چاپ کند.
۱۴. برنامه‌ای بنویسید که ۱۰۰ عدد را از ورودی دریافت کرده و بزرگ‌ترین و کوچک‌ترین مقدار بین آنها و میانگین اعداد را محاسبه کند و نمایش دهد.
۱۵. برنامه‌ای بنویسید که ۵۰ اسم را از ورودی دریافت کند و تعداد افرادی را که نامشان Ali است، نمایش دهد.
۱۶. برنامه‌ای بنویسید که نام و دمای ۱۰ شهر را از ورودی دریافت کند و سپس نام گرم‌ترین و سردترین شهر را نمایش دهد.
۱۷. می‌خواهیم مبلغ ۲۰ ریال را با سکه‌های ۱ ، ۲ ، ۵ و ۱۰ ریالی، خرد کنیم. برنامه‌ای بنویسید که حالت‌های ممکن را نمایش دهد.
۱۸. برنامه‌ای بنویسید که یکی از پاسخ‌های معادله‌ی $۳X^2 - ۶X + ۳ = ۰$ را به روش تقریبی محاسبه کند (برای انجام این کار، محدوده‌ای از اعداد را به‌عنوان ورودی در رابطه قرار دهید و اگر پاسخ صفر شود، این عدد پاسخ معادله است). محدوده از -۱ تا ۱ باشد.
۱۹. مثال $۱۴-۴$ را به نحوی تغییر دهید که به جای دریافت مقدار زاویه از کاربر، مقدار زاویه را به صورت تصادفی تولید کند.

فصل پنجم

رشته‌ها و توابع رشته‌ای

در این فصل، در مورد پردازش‌هایی که می‌توان در ویژوال بیسیک بر روی رشته و نویسه انجام داد، مطالبی را بیان خواهیم کرد. تکنیک‌هایی که در این فصل بیان می‌کنیم، تکنیک‌های مورد نیاز برای ایجاد و براس‌تارهای متن، واژه‌پردازها و سایر نرم‌افزارهایی است که به نحوی با متن سر و کار دارند. در این فصل به طور دقیق با نوع داده‌ی **String** در ویژوال بیسیک آشنا خواهیم شد. کار کردن با رشته‌ها یکی از خصوصیات اولیه‌ی زبان ویژوال بیسیک است و برنامه‌نویس می‌تواند برنامه‌هایی بنویسد که می‌توانند پردازش‌هایی بر روی حروف، کلمات، جملات، اسامی، آدرس‌ها، داده‌های توضیحی و نمادهای ویژه (مانند +، =، -، *، \، \$ و غیره) انجام دهند. عملیات روی رشته‌ها اغلب در **واژه‌پردازها** به کار گرفته می‌شود. با مفهوم و کاربرد واژه‌پردازها در درس بسته‌های نرم‌افزاری ۱ آشنا شده‌اید. در این فصل، در مورد انواع روش‌های موجود که می‌توان از آنها در سیستم‌های واژه‌پرداز استفاده کرد، مطالبی خواهیم آموخت.

پس از پایان این فصل، انتظار می‌رود که فراگیر بتواند:

- انواع توابع رشته‌ای را نام برده و طرز کار هر یک را شرح دهد.
- از توابع رشته‌ای در برنامه‌های خود استفاده کند.

۱-۵ نویسه‌ها و رشته‌ها

در درس مبانی کامپیوتر با مفهوم نویسه آشنا شده‌اید. رشته، دنباله‌ای از نویسه‌هایی است که کنار هم قرار گرفته‌اند و مانند یک واحد عمل می‌کنند. یک رشته ممکن است شامل حروف، ارقام و انواع نویسه‌های خاص مانند +، -، *، \، \$ و غیره باشد. رشته در ویژوال بیسیک از نوع داده‌ی **String** است. در ویژوال بیسیک، رشته به صورت نویسه‌های متوالی که در یک زوج علامت کوتیشن قرار گرفته‌اند، نوشته می‌شود.

ویژوال بیسیک همچنین دارای متغیرهایی از نوع رشته‌ای است که می‌توانند هنگام اجرای برنامه، رشته‌های مختلفی را در خود جای دهند. پسوند نوع اعلان رشته، نماد \$ (دلار) است.

۱-۱-۵ الحاق رشته با & و +

با ترکیب چند رشته‌ی کوچک‌تر، می‌توان رشته‌های بزرگ‌تر را ایجاد کرد. این عمل را می‌توان با استفاده از علامت جمع (+) یا امپرسند (&) انجام داد.

```
s1 = "Pro"
```

```
s2 = "gram"
```

```
s3 = s1 & s2
```

یا

```
s3 = s1 + s2
```

در عبارت‌های فوق، دو رشته‌ی s2 و s1 به یکدیگر الصاق شده و رشته‌ی جدید s3 را به وجود آورده‌اند که شامل Program است. اگر در هنگام الصاق رشته، دو عملوند موجود به صورت رشته باشند، استفاده از عملگر + و & فرقی با یکدیگر ندارند. با این وجود اگر عملگر + با عبارتی به کار رود که از نوع داده‌ی متفاوتی باشد، مسأله‌ساز خواهد بود. به عنوان مثال، در عبارت:

```
s1="hello"+22
```

ویژوال بیسیک ابتدا سعی می‌کند که رشته‌ی "hello" را به یک عدد تبدیل نماید و سپس با ۲۲ جمع کند؛ درحالی که رشته‌ی "hello" را نمی‌توان به عدد تبدیل کرد. بنابراین خطای عدم تطبیق (type mismatch) در اجرای برنامه رخ خواهد داد. بنابراین در الحاق رشته باید از عملگر & استفاده کرد.

نکته

همیشه از عملگر & برای الحاق رشته استفاده کنید.

۲-۱-۵ مقایسه‌ی رشته‌ها

در کار کردن با رشته‌ها، مقایسه‌ی دو رشته از اهمیت خاصی برخوردار است. ویژوال بیسیک، در مقایسه‌ی رشته‌ها از تمام عملگرهای مقایسه‌ای استفاده می‌کند. برای مقایسه‌ی دو رشته از تابع StrComp استفاده می‌شود و شکل کلی آن به صورت زیر است:

Variant یا صحیح = StrComp (String1, String2 [,Compare])

اگر String1 کوچک‌تر از String2 باشد، خروجی این تابع ۱- و اگر بزرگ‌تر باشد، عدد ۱ است و در صورتی که این دو رشته برابر باشند، خروجی صفر است. اگر هر یک از رشته‌ها Null باشند، خروجی تابع نیز Null خواهد بود.

هنگام استفاده از عملگرهای رابطه‌ای، ممکن است حالت خاصی پیش آید و آن Null بودن یکی از اجزای مقایسه است. در این حالت، ویژوال بیسیک مقدار Null برمی‌گرداند نه True یا False. اگر امکان Null بودن یکی از عناصر مقایسه وجود دارد، باید این حالت را هم در نظر بگیرید و برای آن چاره‌ای ببندیشید. ویژوال بیسیک برای مقابله با این وضعیت، توابعی دارد که در برنامه‌سازی ۲ با آنها آشنا خواهید شد.

برای دانستن این که یک رشته بزرگ‌تر یا کوچک‌تر از رشته‌ی دیگر است، باید در مورد پردازش‌هایی که بر روی مجموعه‌ی الفبا صورت می‌گیرد، کمی تأمل کرد. شما مطمئن هستید که "Jahrom" قبل از "Samirom" قرار می‌گیرد. زیرا در مجموعه‌ی الفبا، حرف اول "Jahrom" قبل از حرف اول "Samirom" قرار دارد. الفبا یک لیست ۲۶ حرفی است که به ترتیب در این لیست قرار گرفته‌اند. هر حرف، محل خاصی برای خود دارد. "Z" صرفاً یک حرف الفباست و به صورت دقیق‌تر، حرف بیست و ششم الفباست.

تمام نویسه‌هایی که در رایانه قرار دارند دارای یک کد اسکی (ASCII) هستند و زمانی که رایانه دو رشته را با هم مقایسه می‌کند، در واقع کدهای اسکی این دو را با هم مقایسه می‌نماید. مقادیر ممکن برای آرگومان Compare در جدول ۱-۵ نشان داده شده‌اند.

نوع vbUseCompareOption بر این نکته دلالت دارد که نوع مقایسه در سطح مدول به وسیله‌ی Option Compare تعیین می‌شود و به صورت Option Compare type است که type می‌تواند Text, Binary یا Database باشد. اگر از عبارت Option Compare استفاده شود، به طور پیش‌فرض، نوع مقایسه Binary انتخاب می‌شود.

جدول ۵-۱ انواع مقایسه

| مقایسه | توضیحات نوع |
|-----------------|---|
| VbBinarycompare | عمل مقایسه به صورت حساس به حروف کوچک و بزرگ (Case-Sensitive) است. به عنوان مثال A و a با استفاده از مقادیری که در جدول ASCII دارند، مقایسه می‌شوند و مشخص می‌شود که با هم برابر نیستند. |
| Vbtxtcompare | عمل مقایسه به صورت Case-insensitive (عدم حساسیت به بزرگی و کوچکی حروف) است (به عنوان مثال A و a برابرند). |

مثال ۵-۱

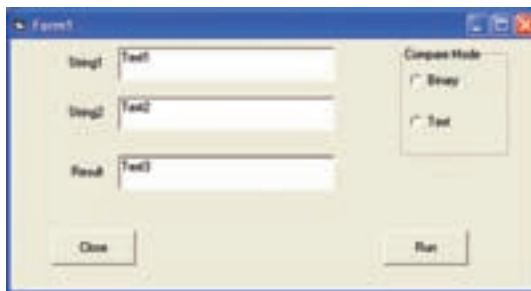
برنامه‌ی زیر، دو رشته را به وسیله‌ی دو کادر متن دریافت کرده و نتیجه‌ی مقایسه‌ی دو رشته را در کادر متن دیگری چاپ می‌کند.

توجه

برای انتخاب نوع مقایسه از OptionButton استفاده شده است.

مراحل کار:

۱. برنامه‌ی ویژوال بیسیک را اجرا کنید و پروژه‌ی جدیدی ایجاد کنید.
۲. فرمی به صورت شکل ۵-۱ طراحی کنید و کد مربوط به دکمه‌ی Close آن را نیز بنویسید.



شکل ۵-۱

۱. در سال سوم با مفهوم پایگاه داده و نرم‌افزار Access آشنا خواهید شد.

۳. روی فرم دوبار کلیک کنید و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub Form_Load()
```

و

```
End Sub
```

دستورهای زیر را وارد کنید:

```
Text1 = ""
```

```
Text2 = ""
```

```
Text3 = ""
```

```
OptBinary.Value = True
```

۴. روی دکمه‌ی Run دوبار کلیک کنید و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub CmdRun_Click()
```

و

```
End Sub
```

دستورهای زیر را وارد کنید:

```
Dim varResult As Variant
```

```
If OptBinary.Value = True Then
```

```
    varResult = StrComp(Text1, Text2, vbBinaryCompare)
```

```
Else
```

```
    varResult = StrComp(Text1, Text2, vbTextCompare)
```

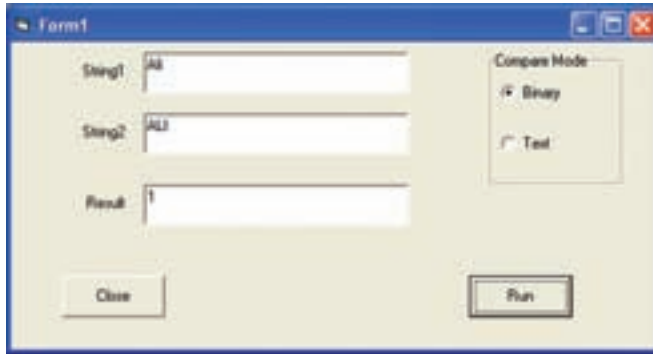
```
End If
```

```
Text3 = varResult
```

۵. برنامه را اجرا کنید و به‌عنوان مثال، کلمات Ali و ALI را وارد کنید و در وضعیت

Binary کلید Run را فشار دهید. در این صورت خروجی به‌صورت شکل ۲-۵

خواهد بود:

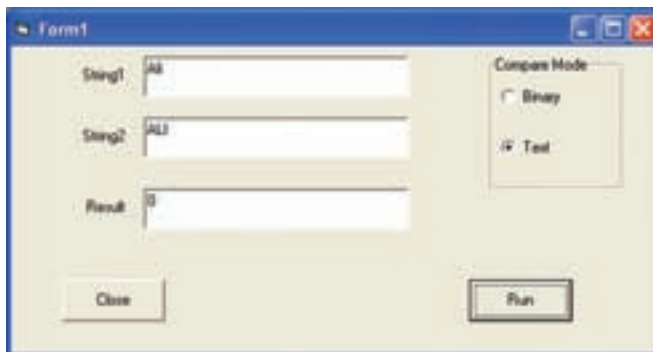


شکل ۵-۲

توجه

همان‌طور که بیان شد، در مقایسه‌ی دودویی، حروف بزرگ و کوچک با هم فرق دارند و کد اسکی حروف کوچک از بزرگ، بیشتر است. در نتیجه در مقایسه‌ی دو کلمه‌ی Ali و ALI ابتدا حروف اول مقایسه شده و چون برابر هستند، حروف دوم مقایسه می‌شوند که در این صورت، حرف l از نظر کد اسکی از L بزرگ‌تر است و جواب مقایسه، ۱ خواهد بود.

۶. برنامه را در وضعیت Text اجرا کنید. در این صورت خروجی به صورت شکل ۵-۳ خواهد بود:



شکل ۵-۳

توجه

در مقایسه به روش متنی، حروف بزرگ و کوچک با هم فرقی ندارند و در نتیجه در مقایسه‌ی دو کلمه‌ی Ali و ALI اختلافی وجود نداشته و چون برابر هستند، نتیجه‌ی مقایسه صفر خواهد بود.

۳-۱-۵ عملگر Like

این عملگر، روش دیگری برای **مقایسه‌ی دو رشته** است. **Like** این امکان را به برنامه‌نویس می‌دهد که الگویی از نویسه‌ها و رشته‌ها را با هم مورد مقایسه قرار دهد. به‌عنوان مثال،

```
"HBLT55DD" Like "HBLT55DD"
```

و

```
"HBLT55DD" Like "HBLT*"
```

هر دو مقایسه درست هستند. در مقایسه‌ی اول، دو رشته برای برابری مورد مقایسه قرار می‌گیرند. در مقایسه‌ی دوم، بررسی می‌کند تا ببیند HBLT55DD رشته‌ای است که با HBLT شروع شده است. علامت * (ستاره) یک نویسه‌ی الگوی تطبیق (Character Pattern) (matching) است که مشخص می‌کند هر تعداد نویسه می‌تواند پس از آن قرار گیرد.

البته نویسه‌های الگوی تطبیق دیگری به غیر از * هم وجود دارد. نویسه‌ی الگوی تطبیق ؟، که مشخص می‌کند یک نویسه از هر نوع می‌تواند در آن محل قرار گیرد (حروف، رقم و غیره) و نویسه‌ی الگوی تطبیق # که مشخص می‌کند یک نویسه از نوع رقم، می‌تواند در آن محل قرار گیرد. به‌عنوان مثال،

```
"HBLT55DD" Like "?#LT55DD"
```

نتیجه‌ی این مقایسه، False است. زیرا حرف دوم رشته‌ی HBLT55DD یک رقم نیست. در این مورد

```
"HBLT55DD" Like "?BLT5#DD"
```

نتیجه‌ی این مقایسه، True است. زیرا حرف اول HBLT55DD می‌تواند هر نویسه‌ای باشد و نویسه‌ی ششم هم، رقم ۵ است.

دنباله‌ای از نویسه‌ها را می‌توان با استفاده از نویسه‌ی الگوی تطبیق [] به کار گرفت. به‌عنوان مثال،

"HBLT55DD" Like "H[A-F]LT55DD"

نتیجه‌ی این مقایسه، True است. زیرا نویسه‌ی دوم در HBLT55DD در محدوده‌ی A تا F قرار دارد و در مورد "HBLT55DD" Like "H[A-F]LT[!4-7]55DD" نتیجه‌ی مقایسه، False است. زیرا نویسه‌ی پنجم در محدوده‌ی ۴ تا ۷ قرار دارد. علامت تعجب (!) هنگامی که در بین یک جفت براکت [] قرار گیرد، مانند عملگر Not عمل می‌کند.

مثال ۲-۵

برنامه‌ی زیر، رشته‌ای را دریافت کرده و به وسیله‌ی یک Pattern نتیجه‌ی مقایسه را نشان می‌دهد.

مراحل کار:

۱. برنامه‌ی ویژوال بیسیک را اجرا کرده و پروژه‌ی جدیدی ایجاد کنید.
۲. فرمی به‌صورت شکل ۴-۵ طراحی کنید و کد مربوط به دکمه‌ی Close آن را نیز بنویسید.



شکل ۴-۵

۳. روی فرم دوبار کلیک کنید و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub Form_Load()
```

و

```
End Sub
```

دستورهای زیر را وارد کنید:

```
Text1 = " "
```

```
Text2 = ""
Text3 = ""
LblPattern.BorderStyle = 1
LblPattern.AutoSize = True
LblPattern = "? Any one Character" + vbNewLine + _
             "* Multiple Character" + vbNewLine + _
             "[chars] Any One characters in Chars" + _
             vbNewLine + _
             "[!chars] Any one charachters not in chars"
```

نکته

هرگاه بخواهیم چند رشته را به‌گونه‌ای به هم بچسبانیم که به هنگام نمایش، زیر هم و در خطوط مجزا دیده شوند، لازم است از ثابت VbNewLine در بین رشته‌ها استفاده کنیم (ثابت VbCrLf نیز همین کار را انجام می‌دهد).

۴. روی دکمه‌ی Run دوبار کلیک کنید و در صفحه‌ای که ظاهر می‌شود، بین دو دستور:

```
Private Sub CmdRun_Click()
```

و

```
End Sub
```

دستورهای زیر را وارد کنید:

```
Dim blnResult As Boolean
```

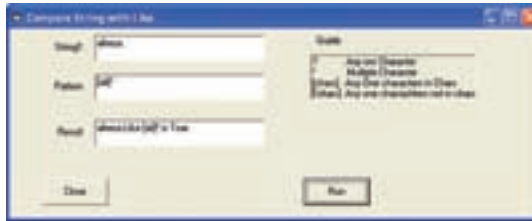
```
blnResult = Text1 Like Text2
```

```
Text3 = Text1 & "Like" & Text2 & "is" & blnResult
```

۵. برنامه را اجرا کنید و به عنوان مثال alireza را وارد کنید و الگوی مقایسه را با *ali]

پُر کنید و کلید Run را فشار دهید. در این صورت خروجی به صورت شکل ۵-۵

خواهد بود.



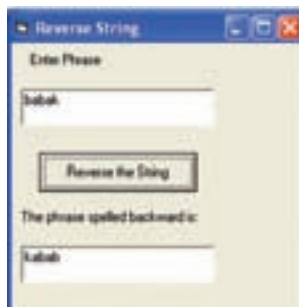
شکل ۵-۵

۵-۱-۴ تابع Mid() و تابع Len()

ویژوال بیسیک برای کار کردن با نویسه‌های موجود در یک رشته دارای چندین ابزار متفاوت است. مثال‌های بعدی در مورد این ابزارها و تکنیک‌ها خواهد بود.

برنامه‌ی زیر، نحوه‌ی استفاده از تابع Mid را که از آن برای **انتخاب یک زیر رشته از یک رشته** استفاده می‌شود، نشان می‌دهد. در این برنامه، کاربر رشته‌ای را به‌عنوان ورودی وارد کرده و معکوس شده‌ی آن رشته را ارایه می‌کند. برای این‌که بتوانیم یک کلمه را به صورت معکوس بنویسیم باید بتوانیم در هر بار، یک نویسه را از رشته استخراج کنیم. در واقع نیاز داریم که به آخرین نویسه دسترسی پیدا کنیم و سپس نویسه‌ی قبل از آن و به همین ترتیب به نویسه‌ی قبلی، و به همین ترتیب تا اولین نویسه. تابع Mid می‌تواند مقصود ما را برآورده سازد.

معمای این مسأله در حلقه‌ی For این برنامه است. حلقه، به‌صورت معکوس از بالاترین موقعیت رشته (انتهای رشته) تا ۱ (ابتدای رشته) تکرار می‌شود. تابع Len طول رشته‌ی دریافتی (phrase) را برمی‌گرداند. سپس با استفاده از تابع Mid در هر بار یک نویسه از رشته‌ی phrase استخراج می‌کند (شکل ۵-۶).



شکل ۵-۶ استفاده از توابع Mid و Len

1. *'Using Mid to reverse a String*
2. **Option Explicit**
3. **Private Sub** cmdReverse_Click()
4. **Dim** phrase **As String**, position **As Integer**
5. txtOutput.Text = ""
6. phrase = txtInput.Text
7. **For** position = **Len**(phrase) **To** 1 **Step** -1
8. txtOutput.Text = txtOutput.Text & _
9. **Mid**(phrase, position, 1)
10. **Next**
11. **End Sub**

رابطه کاربر برنامه شامل یک برچسب، دو کادر متن و یک دکمه‌ی فرمان است. کاربر، یک رشته در اولین کادر متن تایپ کرده و دکمه‌ی Reverse the String را کلیک می‌کند و رشته معکوس می‌شود. روال cmdReverse_Click که در خط ۳ قرار گرفته است، اجرا شده و نتیجه در کادر متن دوم به نمایش در می‌آید. در خط ۶ phrase = txtInput.Text رشته‌ی وارد شده به وسیله‌ی کاربر به phrase تخصیص می‌یابد. خطوط ۸ و ۹:

```
txtOutput.Text = txtOuput.Text &_
Mid(phrase, position, 1)
```

یک نویسه را از آخر رشته به کادر متن txtOutput الصاق می‌کند. عملگر الصاق رشته (&)، نویسه‌های سمت راست خود را به نویسه‌های سمت چپ خود الصاق می‌کند. نتیجه‌ی این عملیات یک رشته‌ی جدید است که می‌تواند در متغیر رشته‌ای ذخیره شود (در اینجا txtOutput.Text).

تابع Mid سه آرگومان می‌گیرد: یک رشته‌ی اصلی که زیر رشته از آن انتخاب خواهد شد، محل شروع نویسه در رشته و تعداد نویسه‌هایی که باید انتخاب شوند. اگر آرگومان آخر حذف شود یا اگر تعداد نویسه‌های باقیمانده کمتر از تعداد نویسه‌های انتخاب شده باشد، باقیمانده‌ی رشته از نویسه‌ی شروع برگردانده می‌شود. در این عبارت، یک نویسه از محل position در phrase انتخاب و برگردانده می‌شود.

از Mid می‌توان برای جایگزین کردن قسمتی از یک رشته در رشته‌ی دیگر هم استفاده کرد. به عنوان مثال، عبارت‌های

```
x = "Visual Basic 6!"
```

```
Mid(x, 2, 3) = "xxx"
```

محتویات x را به صورت "Vxxxal Basic 6!" تغییر می‌دهد.

۵-۱-۵ توابع Left() و Right()

تابع Left، یک زیر رشته با طول معین از سمت چپ رشته جدا کرده و برمی‌گرداند. به عنوان مثال، اگر s1 و s2 متغیرهای رشته‌ای باشند، عبارت‌های

```
s1 = "ABCDEF"
```

```
s2 = Left(s1, 4)
```

از سمت چپ رشته‌ی s1 چهار نویسه‌ی ABCD را انتخاب و به s2 اختصاص می‌دهد. تابع Right، یک زیر رشته با طول معین از سمت راست جدا کرده و برمی‌گرداند. به عنوان مثال، در عبارت‌های

```
s1 = "ABCDEF"
```

```
s2 = Right(s1, 4)
```

از سمت راست رشته‌ی s1 چهار نویسه‌ی CDEF انتخاب و به s2 اختصاص می‌یابد.

۵-۱-۶ توابع InStr و InStrRev

تابع InStr برای جستجو در یک رشته مفید است. رشته‌ای را که باید جستجو شود، رشته‌ی مبنا یا اصلی می‌نامند. اگر رشته‌ی دوم پیدا شود، مکان شروع آن نویسه در رشته برگردانده می‌شود. اگر رشته‌ی دوم دارای طول صفر باشد (یعنی هیچ نویسه‌ای در خود نداشته باشد) موقعیت شروع برگردانده می‌شود. به عنوان مثال، در عبارت‌های

```
s1 = "AEIOU"
```

```
s2 = "IOU"
```

```
result = InStr(1, s1, s2)
```

تعیین می‌کند که s2 (رشته‌ی مورد جستجو) در درون s1 (رشته‌ی اصلی) وجود دارد و در مکان ۳ قرار گرفته است. تابع InStr مکان شروع رشته‌ی s2 در رشته‌ی s1 را برمی‌گرداند که در این مورد ۳ است و این نتیجه (۳) در متغیر result قرار می‌گیرد. در این مثال، عدد ۱ که در InStr به‌کار رفته، بر این نکته دلالت دارد که جستجو برای یافتن رشته‌ی s2 از مکان ۱ در رشته‌ی s1 شروع شود.

می‌توان عمل جستجو را از هر مکانی در s1 آغاز کرد. اولین آرگومان را می‌توان در نظر نگرفت و آن را حذف کرد که در این صورت، عمل جستجو از ابتدای رشته‌ی اصلی آغاز می‌شود. اگر تابع InStr رشته‌ی موردنظر را در رشته‌ی اصلی پیدا نکند، تابع مقدار صفر را برگشت می‌دهد. به‌عنوان مثال در عبارتهای زیر، نتیجه‌ی صفر به وسیله‌ی تابع برگردانده شده و به result اختصاص می‌یابد:

```
result = InStr(1, "aeiou" , "aeb")
```

```
result = InStr(4, "aeiou" , "iou")
```

```
result = InStr(1, "aeiou" , "aeiouy")
```

اغلب واژه‌پردازها قابلیت جستجو برای یافتن رشته‌ای از ابتدا تا انتهای یک سند (document) را دارند. توابع InStrRev و InStr به ترتیب، جستجو برای یافتن یک زیر رشته در درون یک رشته را از ابتدا (نقطه‌ی شروع) تا انتهای رشته و از انتها تا ابتدای رشته انجام می‌دهند. همانطور که قبلاً نیز مشاهده کردید، تابع InStr می‌تواند از هر محلی در رشته، شروع به جستجو کند. **تابع InStrRev می‌تواند جستجو را از انتها یا هر مکانی از یک رشته شروع کند.**

آرگومان‌های تابع InStrRev شامل رشته‌ی اصلی، رشته‌ی مورد جستجو و محل نویسه‌ی آغازین در رشته‌ی اصلی است. اگر در جستجو رشته‌ی موردنظر یافت شود، محل شروع آن نویسه در رشته بازگشت داده می‌شود. اگر طول رشته‌ی جستجو صفر باشد (شامل هیچ نویسه‌ای نباشد)، طول رشته‌ی اصلی برگردانده خواهد شد. به‌عنوان مثال، فرض کنید s1 و s2 دو متغیر رشته‌ای باشند. عبارتهای

```
s1 = "Visual Basic 6.0"
```

```
s2 = "ic"
```

```
result = InStrRev(s1, s2, Len(s1))
```

تعیین می‌کند که s2 (رشته‌ی مورد جستجو) در درون s1 وجود داشته و در مکان ۱۱ قرار دارد. تابع InStrRev، محل شروع رشته‌ی s2 در s1 را که ۱۱ است، برمی‌گرداند و به result اختصاص می‌دهد.

در این مثال، Len(s1) بر این نکته دلالت دارد که جستجوی s2 از انتهای رشته‌ی s1 آغاز می‌شود. در ضمن می‌توان جستجو را از هر محلی در s1 آغاز کرد. اگر منظورتان جستجو از انتهای رشته‌ی اصلی باشد، می‌توانید آرگومان سوم را حذف کنید. دقت کنید که اگر تابع InStrRev رشته‌ی موردنظر را در رشته‌ی اصلی پیدا نکند، تابع، مقدار صفر را برگشت می‌دهد. به‌عنوان مثال، در عبارت‌های زیر، result مقدار صفر خواهد داشت:

```
result = InStrRev("aeiou" , "aeb")
```

```
result = InStrRev("aeiou" , "iou" , 2)
```

```
result = InStrRev("aeiou" , "aeiou")
```

تمرین

نتیجه‌ی تکه برنامه‌ی زیر را بررسی کنید.

```
s1 = "Visual Basic 6.0"
s2 = "a"
result1 = InStrRev(s1, s2, Len(s1))
result2 = InStr(1, s1, s2)
s2 = "ic"
result1 = InStr(4, s1, s2)
result2 = InStrRev(s1, s2, 4)
```

مثال ۳-۵

برنامه‌ی زیر، به کاربر اجازه‌ی وارد کردن دو رشته را در دو کادر متن می‌دهد. هنگامی که دکمه‌ی Search فشرده شود، برنامه جستجو برای یافتن رشته‌ی دوم در درون رشته‌ی اول را آغاز می‌کند و نتیجه، در دو برجسب که در پایین پنجره قرار گرفته‌اند به نمایش در می‌آید. در هنگام کلیک کردن روی دکمه‌ی Search، روال cmdSearch_Click که در خط ۵

قرار گرفته است، اجرا می‌شود. در خط ۵:

```
forwardResult = InStr(txtInput1.Text, txtInput2.Text)
```

از تابع InStr استفاده شده است تا اولین محلی را که txtInput2.text در txtInput1.text وجود دارد، از ابتدای جستجو کند. در خط ۱۳:

```
backwardResult = InStrRev(txtInput1.Text, txtInput2.Text)
```

از تابع InStrRev استفاده شده است تا جستجو از آخر رشته‌ی txtInput1.Text برای یافتن txtInput2.Text آغاز شود (شکل ۷-۵).



شکل ۷-۵ جستجو با استفاده از InStr و InStrRev

1. *'Using InStr and InStrRev to search a string for a Sub string*
2. **Option Explicit**
3. **Private Sub** cmdSearch_Click()
4. **Dim** forwardResult **As Integer**, backwardResult **As Integer**
5. forwardResult = **InStr**(txtInput1.Text, txtInput2.Text)
6. **If** forwardResult <> 0 **Then** *'txtInput2 was found*
7. lblOutput1.Caption = "Forward Search:" & _
8. "The text was found at position" & forwardresult
9. **Else** *'txtInput2 was not found*
10. lblOutput1.Caption = "Forward search:" & _

```

11. "The text was not found"
12. End If
13. backwardResult = InStrRev(txtInput1.Text, _
txtInput2.Text)
14. If backwardresult < > 0 Then
15. lblOutput2.Caption = "Backward search:" &_
16. "The text was found at Position" & backwardresult
17. Else
18. lblOutput2.Caption = "Backward Search:" &_
19. " The text was not found"
20. End If
21. End Sub

```

۷-۱-۵ توابع Trim() ، RTrim() و LTrim()

توابع LTrim ، RTrim و Trim به ترتیب فضاهای خالی در سمت چپ، راست و فضاهای خالی دوطرف یک رشته را حذف می‌کنند. این توابع برای حذف فضاهای خالی اضافی در رشته‌هایی که به صورت اندازه‌ی ثابت تعریف شده‌اند و فضا برای آنها مهم است، مفید هستند.

مثال ۴-۵

برنامه‌ی زیر، به کاربر امکان تایپ رشته‌ای با فاصله در کادر متن را می‌دهد. هنگامی که کاربر روی دکمه‌ی Trim space characters کلیک کند، روال cmdTrim_Click اجرا شده و سه مورد به برجسب اضافه می‌کند. برای هر مورد از *** استفاده شده است تا نتیجه به خوبی مشاهده شود. این برنامه، نتیجه‌ی استفاده از توابع Trim ، RTrim و LTrim است. در این برنامه، TxtInput کادر متنی است که رشته‌ی ورودی در آن قرار دارد و lbl2 برجسبی است که BorderStyle آن روی ۱ تنظیم شده است. در ضمن نام دکمه نیز cmdTrim است (شکل ۸-۵).



شکل ۵-۸

Option Explicit

```
Private Sub cmdTrim_Click()
```

```
lbl2.Caption = "***" & LTrim(TxtInput.Text) &_
"***"+vbNewLine
```

```
lbl2.Caption=lbl2.Caption + "***" &_
```

```
RTrim(TxtInput.Text) &"***" +vbNewLine_
```

```
lbl2.Caption = lbl2.Caption + "***" & Trim(TxtInput.Text)_
&"***"
```

```
End Sub
```

۵-۱-۸ توابع Space() و String()

تابع `String`، رشته‌ای با تعداد نویسه‌ی مشخص ایجاد می‌کند. تابع `Space`، یک رشته که از فضاهای خالی تشکیل شده است، ایجاد می‌کند.

مثال ۵-۵

برنامه‌ی زیر، از این توابع استفاده می‌کند. هنگامی که برنامه اجرا شود، سه مورد روی فرم چاپ می‌شود. در خط ۳:

```
String(10, "A")
```

رشته‌ای با ۱۰ حرف A ایجاد می‌شود.

نکته

در صورتی که رشته بیش از یک نویسه داشته باشد، فقط نویسه‌ی اول تکرار می‌شود.

در خط ۴:

```
String(5, 97)
```

رشته‌ای به طول ۵ نویسه با کد اسکمی ۹۷ (حرف a) ایجاد می‌شود. در خط ۵:

```
Space(5) & String(5, "a")
```

رشته‌ای با ۵ فضای خالی و الصاق آن به رشته‌ای با ۵ حرف a، به وجود می‌آید (شکل ۹-۵).



شکل ۹-۵ استفاده از توابع String و Space

1. **Option Explicit**
2. **Private Sub Form_Activate()**
3. **Print "10 A's: " & String(10, "A")**
4. **Print "5 a's: " & String(5, 97)**
5. **Print "5 a's preceded by 5 spaces:" & Space(5) &_**
String(5, "a")
6. **End Sub**

۹-۱-۵ جایگزینی زیررشته در یک رشته

واژه‌پردازها این توانایی را دارند که یک رشته را جستجو کرده و به جای آن، یک یا چند رشته جایگزین کنند. تابع Replace دارای سه آرگومان اولیه‌ی اختیاری است. اولین آرگومان، رشته‌ای است که زیررشته می‌خواهد در آن جایگزین شود. آرگومان دوم، زیررشته‌ای است که باید پیدا شده و جایگزین شود. آرگومان سوم، رشته‌ای است که باید جایگزین آرگومان دوم شود. آرگومان چهارم

محل نویسه‌ی شروع برای جستجو را تعیین می‌کند (پیش فرض، اولین نویسه است). از آرگومان پنجم برای تعیین تعداد نویسه‌های جایگزین استفاده می‌شود (پیش فرض، کل رشته است).

مثال ۵-۶

برنامه‌ی زیر، از تابع **Replace** استفاده کرده است. هنگامی که کاربر روی دکمه‌ی `cmdReplace_Click` اجرا می‌شود. در عبارت

```
LblOutput.Caption = Replace(txtInput.Text, " ", "..")
```

در این برنامه هر نویسه‌ی فضای خالی با دو نقطه جایگزین خواهد شد. نتیجه، در برجسبی به نام `lblOutput` به نمایش درمی‌آید (شکل ۵-۱۰).



شکل ۵-۱۰ استفاده از تابع **Replace**

1. **Option Explicit**
2. **Private Sub** cmdReplace_Click()
3. LblOutput.Caption = **Replace**(txtInput.Text, " ", "..")
4. **End Sub**

۵-۱-۱۰ معکوس کردن رشته

همانطور که در برنامه‌ی مربوط به تابع `Mid` مشاهده کردید، از تابع `Mid` برای **معکوس کردن رشته** استفاده کردیم. ویژوال بیسیک دارای تابعی به نام `StrReverse` است که این کار را به خوبی انجام می‌دهد.

مثال ۷-۵

برنامه‌ی زیر، از این تابع استفاده می‌کند. هنگامی که کاربر رشته‌ای در کادر متن وارد کند و بر روی دکمه‌ی Reverse the String کلیک کند، روال `cmdReverse_Click()` اجرا می‌شود و دستور

```
TxtOutput.Text = StrReverse(txtInput.Text)
```

رشته را معکوس کرده و نتیجه به `txtOutput.Text` اختصاص می‌یابد (شکل ۱۱-۵).



شکل ۱۱-۵ معکوس کردن رشته با تابع `StrReverse`

1. *'Using StrReverse to reverse the characters in a String*
2. **Option Explicit**
3. **Private Sub** `cmdReverse_Click()`
4. `TxtOutput.Text = StrReverse(txtInput.Text)`
5. **End Sub**

۱۱-۱-۵ سایر توابع رشته‌ای

تابع `Oct()`: این تابع، مقدار آرگومان خود را به مبنای ۸ تبدیل می‌کند. آرگومان از نوع عددی است.

```
Dim MyOct
```

```
MyOct = Oct(4) ' Returns 4.
```

```
MyOct = Oct(8) ' Returns 10.
```

```
MyOct = Oct(459) ' Returns 713.
```

تابع `Hex()`: این تابع، رشته‌ای را برمی‌گرداند که نشان‌دهنده‌ی مقدار مبنای ۱۶ آرگومان خود است. آرگومان از نوع عددی است.

`Dim MyHex`

`MyHex = Hex(5)` ' Returns 5.

`MyHex = Hex(10)` ' Returns A.

`MyHex = Hex(459)` ' Returns 1CB.

۵-۲ توابع تبدیلی

تابع `ASC()`: این تابع، کد اسکی اولین نویسه‌ی یک عبارت رشته‌ای را برمی‌گرداند.

(عبارت رشته‌ای) `ASC`

| | |
|-------------------------------|------------|
| <code>PRINT ASC("B")</code> | مثال ۵-۸ |
| <code>PRINT ASC("SKY")</code> | خروجی : ۶۶ |
| | خروجی : ۸۳ |

تابع `CHR()`: این تابع، برعکس تابع `ASC()` عمل می‌کند و کد اسکی نویسه‌ای را به عنوان آرگومان دریافت می‌کند و نویسه‌ی مربوط را نمایش می‌دهد. آرگومان، عددی بین صفر تا ۲۵۵ است.

(کد اسکی) `CHR`

| | |
|--|----------|
| <code>PRINT CHR(65) + CHR(76) + CHR(73)</code> | مثال ۵-۹ |
| <code>ALI</code> | خروجی |

تابع `STR()`: این تابع، مقدار یک عبارت عددی را که به عنوان آرگومان دریافت می‌کند، به یک مقدار رشته‌ای تبدیل می‌کند.

(عبارت عددی) `STR`

مثال ۱۰-۵

در برنامه‌ی زیر، عددی از ورودی دریافت شده و سپس، ارقام آن به تفکیک چاپ می‌شوند.

```
Dim intNO As Integer
Dim strNO As String
intNO = InputBox("Enter A number:")
strNO = STR(intNO)
For I = 1 TO LEN(strNO)
    PRINT MID(strNO,I,1),
Next I
```

تابع **CStr()**: آرگومان خود را به یک رشته تبدیل می‌کند.

تفاوت تابع **CStr()** با **Str()** (که هر دو آرگومان‌های خود را به رشته تبدیل می‌کنند) در این است که **Str()** در تبدیل اعداد مثبت، قبل از آنها یک نویسه‌ی فضای خالی اضافه می‌کند، در حالی که **CStr()** این نویسه را اضافه نمی‌کند.

مثال ۱۱-۵

با اجرای مثال زیر، تفاوت این دو تابع را بهتر مشاهده خواهید کرد.

```
1. Private Sub convStr ()
2.   Dim str1 As String, str2 As String
3.   Dim intMsg As Integer      'For button clicked
4.   str1 = CStr(12345)
5.   str2 = Str(12345)
6.   intMsg = MsgBox("****" & str1 & "****")
7.   intMsg = MsgBox("****" & str2 & "****")
8. End Sub
```

خروجی خط ۶ عبارت ***** 12345 ***** و خروجی خط ۷ عبارت ***** 12345 ***** خواهد بود.

تابع `VAL()`: این تابع برعکس `STR()` عمل می‌کند و یک رشته‌ی عددی را به عدد معادل آن تبدیل می‌کند.

(عبارت رشته‌ای) `VAL`

مثال ۱۲-۵

```
PRINT VAL ("-33/LP") → -33
PRINT VAL ("25/76") → 25
```

نکته

اگر درون رشته فضای خالی وجود داشته باشد، از آن صرف‌نظر می‌شود.

```
PRINT VAL("12 45") → 1245
```

ولی اگر نویسه‌ی دیگری به کار رود، این تابع فقط تا ابتدای این نویسه، عمل تبدیل را انجام خواهد داد.

۱-۲-۵ تبدیل نوع نویسه‌های رشته

تابع `UCase()`^۱ تمام نویسه‌های آرگومان خود را به نویسه‌های بزرگ و تابع `LCase()`^۲ نویسه‌های آرگومان خود را به نویسه‌های کوچک تبدیل می‌کند. دستور زیر سبب نمایش عبارت `VISUAL BASIC` خواهد شد:

```
Print UCase("Visual Basic")
```

به‌همین ترتیب، دستور زیر سبب نمایش `visual basic` می‌شود:

```
Print LCase("VISUAL BASIC")
```

۳-۵ مشخصه‌های تکمیلی کنترل `Text Box`

می‌توان هنگام تایپ، از رویداد `KeyPress` برای محدود کردن یا انتقال نویسه‌ها استفاده کرد. رویداد `KeyPress` از آرگومان `keyascii` استفاده می‌کند. این آرگومان، عدد صحیحی است که کد اسکی نویسه‌ی تایپ شده در کادر متن را نمایش می‌دهد.

۱. U مخفف Upper است.

۲. L مخفف Lower است.

مثال ۵-۱۳

مثال زیر، چگونگی نادیده گرفتن فشار کلیدها در هنگام تایپ را نشان می‌دهد.

```
Private sub txtEnterNums_ keyPress (keyAscii As Integer)
    If keyAscii < Asc("0") or keyAscii > Asc("9") Then
        keyAscii = 0 'cancel the character.
        Beep          'sound error signal.
    End If
End Sub
```

اگر نویسه‌ی تایپ شده در محدوده‌ی موردنظر نباشد، روال keyAscii با صفر مقداردهی شده و از آن نویسه صرف‌نظر می‌کند. نام کادر متن این مثال، txtEnterNums است و روال از دریافت نویسه‌های غیر رقمی جلوگیری می‌کند.

۵-۳-۱ ایجاد کادر متن فقط خواندنی

می‌توان از مشخصه‌ی Locked برای جلوگیری از ویرایش محتوای کادر متن به وسیله‌ی کاربر، استفاده کرد. مشخصه‌ی Locked را با True مقداردهی کنید تا برای کاربر، امکان اسکرول^۳ و انتخاب متن را بدون امکان تغییر فراهم کنید. هنگامی که این مشخصه با True مقداردهی شود، فرمان Copy در داخل کادر متن کار خواهد کرد ولی فرمان‌های Cut و Paste کار نمی‌کنند. این مشخصه فقط روی عملکرد کاربر در زمان اجرا تأثیر خواهد گذاشت. با وجود تنظیم مشخصه‌ی Locked با True، هنوز هم می‌توان محتوای کادر متن را با تغییر مشخصه‌ی Text در داخل برنامه تغییر داد.

۵-۳-۲ نمایش علامت کوتیشن در یک رشته

چون رشته‌های متنی داخل زوج کوتیشن قرار می‌گیرند، برای چاپ علامت کوتیشن باید از کوتیشن‌های اضافی استفاده کنید. به‌عنوان مثال، برای نمایش عبارت زیر:

She said, "you deserve a treat!"

در یک کادر متن، از کد زیر استفاده کنید:

```
Text1.Text = "she said, ""you deserve a treat! """
```

مثال ۵-۱۴

بررسی گذرواژه

هدف از این پروژه، درخواست گذرواژه از کاربر و بررسی درستی آن است. در صورت درست بودن یا نبودن گذرواژه، کادر پیامی ظاهر شده و آن را به کاربر اعلام می‌کند. دو دکمه‌ی فرمان، یک برجسب و یک کادر متن روی فرم قرار داده و ظاهر آن را مطابق شکل ۵-۱۲، تنظیم کنید.



شکل ۵-۱۲

مشخصه‌ها:

Form1:

| | |
|-------------|---------------------|
| BorderStyle | 1-Fixed Single |
| Caption | Password Validation |
| Name | frmPassword |

Label1:

| | |
|-------------|-----------------------------|
| Alignment | 2-Center |
| BorderStyle | 1-Fixed Single |
| Caption | Please Enter Your Password: |
| FontSize | 10 |
| FontStyle | Bold |

Text1:

| | |
|----------|----|
| FontSize | 14 |
|----------|----|

| | |
|--------------|---|
| FontStyle | Regular |
| Name | txtPassword |
| PasswordChar | * |
| Tag | [گذرواژه‌ی موردنظر را در اینجا تایپ کنید] |
| Text | [خالی] |

Command1:

| | |
|---------|------------|
| Caption | & Validate |
| Default | True |
| Name | cmdValid |

Command2:

| | |
|---------|---------|
| Cancel | True |
| Caption | E&xit |
| Name | cmdExit |

کد برنامه به صورت زیر خواهد بود:

```
Private Sub cmdValid_Click()
    این روال، گذرواژه‌ی ورودی را بررسی می‌کند؛
    Dim Response As Integer
    If txtPassword.Text = txtPassword.Tag Then
        اگر گذرواژه معتبر بود، کادر پیغام نمایش داده می‌شود؛
        MsgBox "You've passed security!", vbOKOnly+ _
        vbExclamation, "Access Granted"
    Else
        اگر گذرواژه معتبر نباشد، امکان سعی مجدد را فراهم می‌کند؛
        Response=MsgBox("Incorrect password", vbRetryCancel+ _
        vbCritical,"Access Denied")
        If Response = vbRetry Then
            txtPassword.SelStart = 0
```

```
txtPassword.SelLength = Len(txtPassword.Text)
Else
End
End If
End If
txtPassword.SetFocus
End Sub
Private Sub Form_Activate()
    txtPassword.SetFocus
End Sub
Private Sub cmdExit_Click()
    End
End Sub
```


خلاصه‌ی فصل

رشته، دنباله‌ای از نویسه‌هاست که در کنار هم قرار گرفته‌اند. برای ترکیب رشته‌ها از عملگر + یا & استفاده می‌شود. توابع رشته‌ای متعددی وجود دارد که برخی از آنها به صورت زیر است:

تابع StrComp: برای مقایسه‌ی دو رشته به کار می‌رود. عملگر Like نیز به همین منظور استفاده می‌شود.

تابع Mid: برای جدا کردن یک زیررشته از یک رشته به کار می‌رود.

تابع Left: یک زیررشته از سمت چپ رشته جدا کرده و برمی‌گرداند.

تابع Right: یک زیررشته از سمت راست رشته جدا کرده و برمی‌گرداند.

توابع Instr و InstrRev: برای جستجو در یک رشته مورد استفاده قرار می‌گیرند.

توابع LTrim, RTrim و Trim: به ترتیب فضاهای خالی سمت چپ، راست و فضاهای خالی دوطرف یک رشته را حذف می‌کنند.

تابع String: رشته‌ای با تعداد نویسه‌ی مشخص ایجاد می‌کند.

تابع Space: یک رشته را که از فضای خالی تشکیل شده است، ایجاد می‌کند.

تابع Replace: یک رشته را جستجو کرده و با رشته‌ی دیگر جایگزین می‌کند.

تابع StrReverse: معکوس یک رشته را برمی‌گرداند.

تابع Oct: این تابع، مقدار آرگومان خود را به مبنای ۸ تبدیل می‌کند.

تابع Hex: این تابع، رشته‌ای را برمی‌گرداند که نشان‌دهنده‌ی مقدار مبنای ۱۶ آرگومان خود است.

تابع ASC: این تابع کد اسکی اولین نویسه‌ی یک عبارت رشته‌ای را برمی‌گرداند.

تابع CHR: این تابع، عکس تابع ASC عمل می‌کند و کد اسکی نویسه‌ای را به عنوان

آرگومان دریافت می‌کند و نویسه‌ی مربوطه را نمایش می‌دهد.

تابع STR: این تابع، مقدار یک عبارت عددی را که به عنوان آرگومان دریافت می‌کند، به

یک مقدار رشته‌ای تبدیل می‌کند.

تابع CStr: آرگومان خود را به یک رشته تبدیل می‌کند.

تابع VAL: این تابع، عکس STR کار می‌کند و یک رشته‌ی عددی را به عدد معادل آن

تبدیل می‌کند.

تابع UCase تمام نویسه‌های آرگومان خود را به نویسه‌های بزرگ، و تابع LCase نویسه‌های آرگومان خود را به نویسه‌های کوچک تبدیل می‌کند.

به کمک رویداد KeyPress می‌توان هنگام تایپ نویسه‌ها در کادر متن، آنها را محدود کرد. مقداردهی مشخصه‌ی Locked با True سبب می‌شود که کاربر نتواند محتوای کادر متن را ویرایش کند.

خودآزمایی

۱. برنامه‌ای بنویسید که نام سه نفر را دریافت کرده و به ترتیب چاپ کند.
۲. برنامه‌ای بنویسید که نام و نام خانوادگی شخصی را از ورودی دریافت کرده و در یک متغیر قرار دهد و سپس آنها را به صورت مجزا نمایش دهد.
۳. برنامه‌ای بنویسید که رشته‌ای را از ورودی دریافت کرده و تعداد حروف بزرگ و کوچک آن را شمارش کند و با پیام مناسبی نمایش دهد.
۴. برنامه‌ای بنویسید که دو عدد در مبنای دودویی را از ورودی دریافت کرده و حاصل جمع آنها را محاسبه کند و نمایش دهد.
۵. برنامه‌ای بنویسید که رشته‌ای را از ورودی دریافت کرده و تعیین کند که آیا رشته از هر دو طرف که در نظر گرفته شود، یکسان است یا خیر؟ به عنوان مثال، رشته‌ی "beeb" چنین خاصیتی دارد.
۶. برنامه‌ای بنویسید که رشته‌ای را از ورودی دریافت کند و مجموع ارقام نویسه‌ای موجود در رشته را محاسبه نماید و نمایش دهد (کد اسکی ارقام ۰ تا ۹ برابر است با ۴۸ تا ۵۷).
۷. برنامه‌ای بنویسید که بدون استفاده از تابع Replace، کار این تابع را شبیه‌سازی کند.
۸. برنامه‌ای بنویسید که عددی را در مبنای ۱۶ دریافت کرده و معادل آن را در مبنای ۱۰ نمایش دهد.

فصل ششم

پروژه‌های برنامه‌نویسی

۶-۱ ماشین حساب

پروژه‌ای که به شرح گام‌به‌گام آن می‌پردازیم یک **ماشین حساب** است که دارای چهار عمل اصلی، توابع مثلثاتی، لگاریتمی و توان به شکل‌های مختلف است. ولی لازم است قبل از انجام کار به نکات زیر توجه کنید:

- برای نمایش اعداد به جای کادر متن، از یک برچسب استفاده می‌کنیم زیرا نباید اجازه بدهیم کاربر بتواند درون کادر متن چیزی مثل رشته بنویسد.
- عملگرهای محاسباتی از نقطه نظر ما به دو دسته تقسیم می‌شوند: دسته‌ی اول، آنهایی هستند که برای محاسبه به **دو عملوند** یا عدد نیاز دارند، مثل $+$ ، $-$ ، \times ، $/$ و X^Y و دسته‌ی دوم آنهایی هستند که برای محاسبه فقط به **یک عملوند** نیاز دارند، مثل توابع مثلثاتی، لگاریتم و توان ۲ و ...
- در حالت عادی یعنی هنگامی که هیچ عددی نوشته نشده است، لازم است عدد صفر به صورت "0." نمایش یابد.

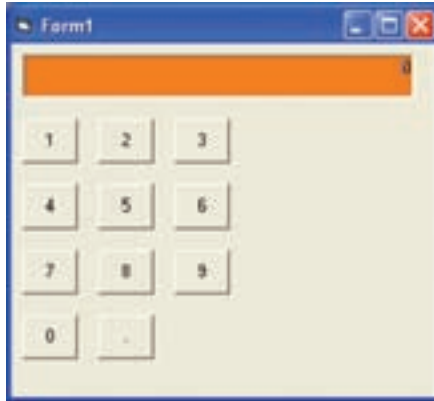
۶-۱-۱ مرحله‌ی اول طراحی

ابتدا فرمی مانند شکل ۶-۱ طراحی کرده و در طرح خود به نکات زیر توجه کنید:
برچسبی به نام lblOutput را برای نمایش اعداد در فرم قرار دهید و مشخصه‌های زیر را از طریق پنجره‌ی مشخصه‌ها، برای آن تنظیم کنید:

BorderStyle = 1

BorderColor = یک رنگ دلخواه

Caption = 0.



شکل ۱-۶

برای فرم نیز مشخصه‌های زیر را تنظیم کنید:

Caption = Calculator

BorderStyle = 1-Fixed single

کلیدهای اعداد (۰ تا ۹) را روی فرم قرار داده و Caption و Name آنها را متناسب با آن عدد تنظیم کنید (از طریق پنجره‌ی مشخصه‌ها). به عنوان مثال، برای کلید ۱، Caption=1 و Name = cmd1 و برای کلید ۲، Caption = 2 و Name = cmd2 و الی آخر. برای کلید نقطه ممیز نیز، Caption = . و Name = cmdpoint را تنظیم کنید.

بر روی فرم دوبار کلیک کرده و در بخش General متغیری به صورت Dim blnpoint Boolean AS تعریف کنید. وظیفه‌ی این متغیر آن است که تعیین کند که آیا عدد وارد شده یک نقطه ممیز دارد یا نه؟ زیرا نباید اجازه دهیم بیش از یک ممیز در عدد وجود داشته باشد. روی عدد ۱ دوبار کلیک کرده و در رویداد مربوطه دستوره‌ای زیر را بنویسید:

1. **Private Sub** cmd1_Click()
2. **If** lblOutPut = "0." **And** blnPoint = **False** **Then**
3. lblOutPut = "1"
4. **Else**
5. lblOutPut = lblOutPut + "1"
6. **End If**
7. **End Sub**

شرح کار برنامه:

در سطر ۲، ابتدا lblOutput با "0." و blnPoint با False مقایسه می‌شوند. اگر blnPoint = False بود یعنی نقطه ممیز هنوز فشار داده نشده است و اگر lblOutput برابر "0." بود، یعنی هیچ عددی وارد نشده است. بنابراین، عدد ۱ اولین عدد ورودی است و باید به‌طور مستقیم در برجسب نوشته شود و در سطر ۳ دستور lblOutput="1" این کار را انجام می‌دهد. اما اگر یکی از این دو شرط یا هر دو نادرست باشند، بدین معناست که نقطه ممیز قبلاً زده شده است، پس عدد ۱ باید بعد از آن نوشته شود، مثل (1). یا قبلاً عددی وارد شده که باید عدد ۱ به دنبال آن برجسبند. بنابراین، از دستور lblOutput=lblOutput+"1" در سطر ۵ استفاده شده است.

برای کلیدهای ۲ تا ۹ و صفر نیز همین عمل را انجام دهید. متن برنامه‌ی ۱-۶ را ببینید.

برنامه‌ی ۱-۶

```
Private Sub cmd1_Click()
If lblOutput = "0." And blnpoint = False Then
    lblOutput = "1"
Else
    lblOutput = lblOutput + "1"
End If
End Sub
```

```
Private Sub cmd2_Click()
If lblOutput = "0." And blnpoint = False Then
    lblOutput = "2"
Else
    lblOutput = lblOutput + "2"
End If
End Sub
```

```
Private Sub cmd3_Click()
```

```
If lblOutPut = "0." And blnpoint = False Then
```

```
    lblOutPut = "3"
```

```
Else
```

```
    lblOutPut = lblOutPut + "3"
```

```
End If
```

```
End Sub
```

```
Private Sub cmd4_Click()
```

```
If lblOutPut = "0." And blnpoint = False Then
```

```
    lblOutPut = "4"
```

```
Else
```

```
    lblOutPut = lblOutPut + "4"
```

```
End If
```

```
End Sub
```

```
Private Sub cmd5_Click()
```

```
If lblOutPut = "0." And blnpoint = False Then
```

```
    lblOutPut = "5"
```

```
Else
```

```
    lblOutPut = lblOutPut + "5"
```

```
End If
```

```
End Sub
```

```
Private Sub cmd6_Click()
```

```
If lblOutPut = "0." And blnpoint = False Then
```

```
    lblOutPut = "6"
```

```
Else
```

```
    lblOutPut = lblOutPut + "6"
```

```
End If
```

```
End Sub
```

```
Private Sub cmd7_Click()  
If lblOutPut = "0." And blnpoint = False Then  
    lblOutPut = "7"  
Else  
    lblOutPut = lblOutPut + "7"  
End If  
End Sub
```

```
Private Sub cmd8_Click()  
If lblOutPut = "0." And blnpoint = False Then  
    lblOutPut = "8"  
Else  
    lblOutPut = lblOutPut + "8"  
End If  
End Sub
```

```
Private Sub cmd9_Click()  
If lblOutPut = "0." And blnpoint = False Then  
    lblOutPut = "9"  
Else  
    lblOutPut = lblOutPut + "9"  
End If  
End Sub
```

```
Private Sub cmd0_Click()  
If lblOutPut = "0." And blnpoint = False Then  
    lblOutPut = "0."  
Else  
    lblOutPut = lblOutPut + "0"  
End If  
End Sub
```


بر روی cmdPoint دوبار کلیک کرده و دستورهای زیر را بنویسید:

1. **Private Sub** cmdpoint_Click()
2. blnpoint = **True**
3. **If InStr**(lblOutPut, ".") = 0 **Then**
4. lblOutPut = lblOutPut + "."
5. **End If**
6. **End Sub**

شرح کار رویداد:

در سطر ۲ متغیر blnPoint با مقدار True پر می‌شود. از این متغیر برای تعیین اینکه دکمه‌ی cmdPoint فشار داده شده استفاده شده است تا در دکمه‌های عددی دیگر بدانیم که باید عدد جدید را بعد از ممیز اضافه کنیم.

در سطر ۳ به وسیله‌ی تابع InStr وجود نقطه در برجسب خروجی را بررسی می‌کنیم تا در یک عدد بیش از یک ممیز وارد نکنیم و اگر وجود نداشت، یعنی نتیجه‌ی تابع صفر بود، یک نقطه ممیز به آن اضافه کنیم.

برنامه را اجرا کنید و اعدادی را وارد کنید. مشاهده خواهید کرد که مشکلی وجود ندارد و فقط می‌توان یک ممیز در عدد اضافه کرد.

۶-۱-۲ مرحله‌ی دوم طراحی

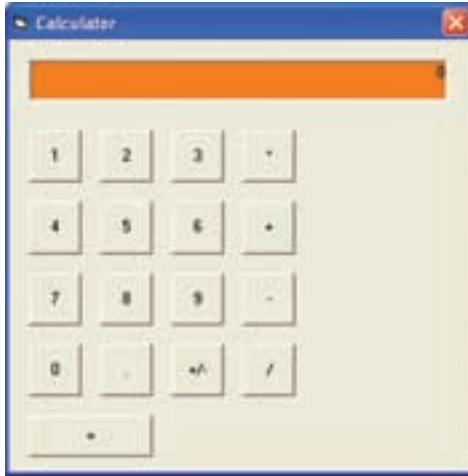
مطابق شکل ۶-۲، به برنامه‌ی خود چهار کلید برای چهار عمل اصلی، یک کلید برای تغییر علامت و یک کلید برای مساوی اضافه کنید.

نام کلید * را به cmdMul، نام کلید + را به cmdPlus، نام کلید - را به cmdMinus، نام کلید / را به cmdDiv، نام کلید علامت (+/-) را به cmdSign و نام کلید مساوی را به CmdEqual تغییر دهید.

در بخش General متغیرهایی به صورت زیر تعریف کنید:

Dim op As String * 1, strLastinput As String

Dim n1 As Double



شکل ۲-۶

متغیر op از نوع رشته‌ای به طول ۱ بوده و از آن برای نگهداری عملگر فشار داده شده استفاده می‌شود. زیرا در یک ماشین حساب، ابتدا عدد اول، بعد عملگر و سپس عدد دوم و در پایان مساوی وارد می‌شود. پس لازم است عملگر زده شده را ذخیره کنیم تا پس از زدن مساوی از روی آن عملگر، محاسبه را انجام دهیم.

متغیر strLastInput از نوع رشته‌ای بوده و برای تعیین آخرین کلید فشار داده شده استفاده می‌شود. اگر آخرین کلید فشار داده شده عملگر بود، مقدار "op" و اگر عدد یا ممیز بود، مقدار "nums" در آن ذخیره می‌شود.

متغیر n1 از نوع Double (اعشاری با دقت مضاعف) تعریف شده است تا بتواند عدد اول را در خود نگه دارد.

در این کتاب نحوه‌ی ساخت تابع آموزش داده نشده است ولی برای جلوگیری از افزایش حجم برنامه، تابعی به صورت زیر در بخش General تعریف می‌کنیم:

```
Private Function Calc(a As Double, b As Double, op
As String) As Double
```

```
Select Case op
```

```
Case "*"
```

```
Calc = a * b
```

```
Case "-"
```

Calc = a - b

Case "+"

Calc = a + b

Case "/"

Calc = a / b

Case " "

Calc = b

End Select

End Function

این تابع دارای سه پارامتر یا آرگومان ورودی است: پارامتر اول و دوم (a,b) برای انتقال دو مقدار عددی و پارامتر سوم برای تعیین نوع عملگر. سپس با استفاده از فرمان Select Case، متغیر op بررسی شده و عملیات مربوطه انجام می‌شود. باید توجه داشت که در یک تابع برای بازگرداندن مقداری به خروجی، باید مقدار موردنظر را در اسم تابع ذخیره کنیم. به عنوان مثال، برای محاسبه حاصل ضرب دو عدد می‌نویسیم $Calc=a*b$ و ...

اگر عملگر خالی باشد یعنی هیچ عملگری زده نشود، باید عدد دوم را برگرداند. در صورتی که نخواهیم از این تابع استفاده کنیم، باید به جای تابع در برنامه، کل دستورهای درون تابع را در رویدادهایی که از تابع استفاده کرده‌اند، به صورت زیر مورد استفاده قرار دهیم:

Select Case op

Case "*"

n1 = n1 * Val(lblOutPut)

Case "-"

n1 = n1 - Val(lblOutPut)

Case "+"

n1 = n1 + Val(lblOutPut)

Case "/"

n1 = n1 / Val(lblOutPut)

Case " "

n1 = Val(lblOutPut)

End Select

توجه

از این تابع در روال‌های دکمه‌های cmdDiv, cmdPlus, cmdMinus
cmdEqual, cmdMul استفاده شده است.

بر روی فرم دوبار کلیک کرده و در رویداد Load دستوراتی را به صورت زیر وارد کنید:

```
Private Sub Form_Load()
```

```
n1 = 1
```

```
op = " "
```

```
strLastinput = " "
```

```
blnpoint = False
```

```
End Sub
```

در این رویداد $n1 = 1$ شده است، زیرا هیچ عددی در لحظه شروع وارد نشده است تا $n1$ پر شود. $op = " "$ با یک فاصله پر می‌شود تا بیانگر خالی بودن عملگر در لحظه شروع باشد و $StrLastInput = " "$ زیرا هنوز هیچ عملگر یا عددی وارد نشده است. بر روی cmdMul دوبار کلیک کرده و دستورهای زیر را بنویسید:

1. **Private Sub** cmdMul_Click()
2. **If** strLastinput = "nums" **Then**
3. $n1 = \text{Calc}(n1, \text{Val}(\text{lblOutPut}), \text{op})$
4. $\text{lblOutPut} = n1$
5. **End If**
6. $n1 = \text{Val}(\text{lblOutPut})$
7. $\text{op} = "*"$
8. $\text{strLastinput} = \text{"op"}$
9. $\text{blnpoint} = \text{False}$
10. **End Sub**

شرح کار رویداد:

در سطر ۲، متغیر strLastInput با رشته‌ی "nums" مقایسه می‌شود. اگر آخرین کلید فشار داده شده از نوع عددی باشد، بدین معناست که اولین یا دومین عدد به طور کامل وارد شده

است. اگر عدد دوم وارد شده باشد، یعنی عدد اول در n1 و عملگر قبل در op ذخیره شده است. بنابراین، باید ابتدا حاصل ورودی قبلی محاسبه شود. برای درک بهتر به مثال زیر توجه کنید: فرض کنید ابتدا عدد ۲ و بعد عملگر * و سپس عدد ۳ و در نهایت عملگر + وارد شده است. ورودی معادل است با $2 * 3 + \dots$ یعنی بعد از عملگر + باید عدد دیگری وارد شود ولی قبل از عملگر + یعنی عدد دوم وارد شده، پس باید حاصل ۲ ضرب در ۳ ابتدا محاسبه شود، سپس نتیجه با عدد بعد از + جمع شود. برای انجام این کار، از تابع Calc به صورت $n1 = \text{Calc}(n1, \text{Val}(lblOutput), op)$ استفاده کرده‌ایم. باید توجه داشت که عدد اول در n1 و عدد دوم در برچسب خروجی وجود دارد. حاصل، پس از محاسبه در n1 قرار می‌گیرد و در سطر ۳ نیز در برچسب خروجی قرار می‌گیرد.

اما اگر قبل از زدن * فقط یک عدد وارد شده باشد، لازم است عملگر ذخیره و عدد وارد شده نیز در n1 ذخیره شود تا بعداً با عدد دوم محاسبه شود. این عمل در سطرهای ۶ و ۷ انجام می‌گیرد. در سطر ۸، strLastInput با رشته‌ی "op" پر می‌شود تا بعداً معلوم شود که آخرین کلید فشار داده شده، عملگر بوده است.

در سطر ۹، متغیر blnPoint برابر False قرار می‌گیرد زیرا هرگاه عملگری زده شود به این معناست که ورود عدد قبلی خاتمه یافته است و برای عدد بعدی ممکن است ممیز نیاز داشته باشیم. پس این متغیر را False می‌کنیم تا در مراحل بعدی از آن استفاده کنیم. برای سه عملگر بعدی نیز به ترتیب قبل عمل کنید، متن برنامه‌ی ۲-۶ را ببینید.

برنامه‌ی ۲-۶

```
Private Sub cmdMul_Click()
    If strLastinput = "nums" Then
        n1 = Calc(n1, Val(lblOutput), op)
        lblOutput = n1
    End If
    n1 = Val(lblOutput)
    op = "*"
    strLastinput = "op"
    blnpoint = False
End Sub
```

```
Private Sub cmdDiv_Click()  
    If strLastinput = "nums" Then  
        n1 = Calc(n1, Val(lblOutPut), op)  
        lblOutPut = n1  
    End If  
    n1 = Val(lblOutPut)  
    op = "/"  
    strLastinput = "op"  
    blnpoint = False
```

```
End Sub
```

```
Private Sub cmdMinus_Click()  
    If strLastinput = "nums" Then  
        n1 = Calc(n1, Val(lblOutPut), op)  
        lblOutPut = n1  
    End If  
    n1 = Val(lblOutPut)  
    op = "-"  
    strLastinput = "op"  
    blnpoint = False
```

```
End Sub
```

```
Private Sub cmdPlus_Click()  
    If strLastinput = "nums" Then  
        n1 = Calc(n1, Val(lblOutPut), op)  
        lblOutPut = n1  
    End If  
    n1 = Val(lblOutPut)  
    op = "+"  
    strLastinput = "op"  
    blnpoint = False
```

```
End Sub
```

بر روی cmdSign دوبار کلیک کرده و دستورهای زیر را بنویسید:

1. **Private Sub** cmdSign_Click()
2. **If** lblOutPut = "0." **And** blnpoint = **False** **Then**
3. **Exit Sub**
4. **End If**
5. **If** lblOutPut = "0." **Then**
6. lblOutPut = "-0."
7. **Else**
8. lblOutPut = **-Val**(lblOutPut)
9. **End If**
10. **End Sub**

شرح کار رویداد:

در سطر ۲، ابتدا برچسب خروجی با رشته‌ی "0." و blnpoint نیز با False مقایسه می‌شود. اگر هر دو شرط درست بود، یعنی اولاً عددی وارد نشده است زیرا رشته‌ی "0." دیده می‌شود، ثانیاً دکمه‌ی ممیز زده نشده است. بنابراین، لازم نیست عدد صفر را با علامت منفی نشان دهیم، در نتیجه با فرمان Exit Sub در سطر ۳، از رویداد خارج می‌شویم. اما اگر قبلاً ممیز زده شده باشد، یعنی به دنبال آن، عدد وارد می‌شود یا وارد شده است. پس در سطر ۵ بررسی می‌کنیم که آیا بعد از ممیز، عدد زده شده یا نه؟ اگر وارد نشده است، رشته‌ی "0-" نمایش می‌یابد ولی اگر عدد وارد شده است، در سطر ۸ ابتدا برچسب خروجی به عدد تبدیل و سپس منفی می‌شود و در برچسب خروجی نمایش می‌یابد. این عمل سبب می‌شود که عدد مثبت به منفی و عدد منفی به مثبت تبدیل شود.

بر روی دکمه‌ی مساوی دوبار کلیک کرده و دستورهای زیر را بنویسید:

1. **Private Sub** cmdEqual_Click()
2. n1 = **Calc**(n1, **Val**(lblOutPut), op)
3. **If** n1 = 0 **Then**
4. lblOutPut = "0."
5. **Else**

6. lblOutPut = n1
7. End If
8. op = " "
9. strLastinput = "op"
10. blnpoint = False
11. End Sub

در سطر ۲، به وسیله‌ی تابع Calc عدد اول n1 با عدد دوم که در برچسب قرار دارد با عملگری که در op ذخیره شده است، محاسبه و نتیجه در n1 ذخیره می‌شود. در سطر ۳، n1 با صفر مقایسه می‌شود. اگر حاصل محاسبه صفر باشد، در خروجی به صورت "0" و در غیر این صورت عدد n1 نمایش می‌یابد.

در سطر ۸، متغیر "op=" می‌شود زیرا پس از زدن مساوی نباید هیچ عملگری از قبل باقی بماند. در سطر ۹ نیز strLastinput="op" می‌شود زیرا مساوی یک عملگر محسوب می‌شود و در سطر ۱۰ نیز متغیر blnpoint با False مقداردهی می‌شود، زیرا محاسبه تمام شده و آماده‌ی دریافت عدد جدید هستیم. برنامه را اجرا کنید. ماشین حساب به خوبی کار می‌کند.

۳-۱-۶ مرحله‌ی سوم طراحی

مطابق شکل ۳-۶، به برنامه‌ی خود چهار کلید برای چهار عمل حافظه‌ای (MC, MS, MR, M+) و دو کلید برای عمل حذف یا پاک کردن (C, CE) و یک دکمه برای حذف یک به یک ارقام عدد (BKS) و یک برچسب برای نمایش وضعیت حافظه اضافه کنید. کار کلیدهای حافظه به شرح زیر است:

- MS (Memory Set): یعنی مقدار فعلی در حال نمایش را در حافظه ذخیره کن
- MC (Memory Clear): یعنی متغیر حافظه را پاک کن
- MR (Memory Request): یعنی بازگرداندن مقدار درون حافظه به برچسب خروجی
- M+(Add To Memory): یعنی افزودن مقدار فعلی lblOutPut به مقدار قبلی

حافظه



شکل ۳-۶

برای ذخیره‌ی عدد در حافظه در بخش General متغیری به نام dbLM به صورت زیر تعریف کنید:

Dim dbLM As Double

نام کلیدهای حافظه را به ترتیب cmdMP برای M+ و cmdMC برای MC و cmdMS و cmdMR برای MS و cmdMR برای MR قرار دهید و برای برجسب حافظه نیز نام lblM را انتخاب کنید. برای دکمه‌ی CE نام cmdCE و برای دکمه‌ی C نام cmdC و برای دکمه‌ی BKS یا همان BackSpace نام cmdBks را انتخاب کنید.

بر روی دکمه‌ی MS دوبار کلیک کرده و دستورهای زیر را بنویسید:

Private Sub cmdMS_Click()

```
dbLM = Val(lblOutput)
```

```
lblM = "M"
```

```
strLastinput = "op"
```

```
blnpoint = False
```

End Sub

در این رویداد، ابتدا مقدار lblOutput خوانده شده و پس از تبدیل به عدد، در متغیر dbLM ذخیره می‌شود. سپس برجسب حافظه با حرف M مقداردهی می‌شود و چون کلید

فشار داده شده از نوع عددی نیست، لازم است strLastInput با "op" و blnpoint با False مقداردهی شود تا عدد بعدی یک عدد جدید باشد.
بر روی دکمه‌ی MC دوبار کلیک کرده و دستورهای زیر را بنویسید:

```
Private Sub cmdMC_Click()
```

```
    dblM = 0
```

```
    lblM = " "
```

```
    strLastinput = " "
```

```
End Sub
```

در این رویداد متغیر dblM صفر یا پاک می‌شود. برجسب نمایش پر بودن حافظه یا همان lblM خالی می‌شود و StrLastInput نیز خالی می‌شود زیرا این دکمه جزو هیچ دسته‌ای از عملیات، مثل ورود عدد یا عملگر نیست.
بر روی دکمه‌ی MR دوبار کلیک کرده و دستورهای زیر را بنویسید:

```
Private Sub cmdMR_Click()
```

```
    lblOutPut = IIf(dblM = 0, "0.", dblM)
```

```
    strLastinput = "nums"
```

```
    If InStr(lblOutPut, ".") < > 0 Then blnpoint = True
```

```
End Sub
```

در این رویداد ابتدا مقدار متغیر حافظه با صفر مقایسه می‌شود و اگر صفر بود، رشته‌ی "0." و در غیر این صورت، خود همان مقدار در lblOutPut قرار می‌گیرد. سپس متغیر strLastinput با رشته‌ی "nums" مقداردهی می‌شود زیرا عدد موجود در حافظه مثل یک عدد جدید در خروجی نمایش یافته و این متغیر باید نشان‌دهنده‌ی وجود عدد جدید در خروجی باشد. در سطر آخر در برجسب خروجی به دنبال ممیز می‌گردیم. اگر ممیز وجود داشت، متغیر lblpoint را True می‌کنیم تا معلوم شود در عدد فعلی یک ممیز وجود دارد و نیازی به ممیز دیگری نیست.

بر روی دکمه‌ی M+ دوبار کلیک کرده و دستورهای زیر را بنویسید:

```
Private Sub cmdMP_Click()
```

```
    dblM = dblM + Val(lblOutPut)
```

```
lblM = "M"
strLastinput = "op"
```

End Sub

فشار دادن کلید $M+$ به معنی اضافه کردن مقدار فعلی درون خروجی، به مقدار قبلی حافظه است. در نتیجه در این رویداد، ارزش عدد برچسب خروجی به وسیله فرمان $dblM = dblM + Val(lblOutput)$ به متغیر حافظه اضافه می شود و برچسب حافظه نیز با حرف M پر می شود تا نشان دهنده وجود عدد در حافظه باشد. در خاتمه $strLastInput$ با رشته op مقداردهی می شود تا نشان دهد که آخرین کلید فشار داده شده عملگر بوده است. بر روی دکمه C دوبار کلیک کرده و دستورهای زیر را بنویسید:

Private Sub cmdC_Click()

```
n1 = 1
op = " "
lblOutput = "0."
strLastinput = " "
blnpoint = False
```

End Sub

هرگاه کلید C فشار داده شود، یعنی بجز حافظه همه چیز باید پاک یا Reset شوند. مثل این که برنامه از ابتدا اجرا شده است. در رویداد فوق، همین عمل اتفاق افتاده است. بر روی دکمه CE دوبار کلیک کرده و دستورهای زیر را بنویسید:

Private Sub cmdCE_Click()

```
lblOutput = "0."
strLastinput = "op"
blnpoint = False
```

End Sub

فشار دادن کلید CE به این معنی است که فقط آخرین عدد وارد شده حذف شود. در این رویداد برچسب خروجی، صفر شده و متغیر $strLastInput$ نیز بر روی op تنظیم می شود زیرا هنگامی که آخرین عدد پاک شود، کلید قبلی حتماً عملگر بوده است. در ضمن

چون عدد پاک شده است، لازم است blnpoint نیز False شود، زیرا عددی وجود ندارد که ممیز داشته باشد.

بر روی دکمه‌ی Bks دوبار کلیک کرده و دستورهای زیر را بنویسید:

```
Private Sub cmdBks_Click()
Dim L As Byte
L = Len(lblOutPut)
If L > 0 Then
    lblOutPut = Left(lblOutPut, L - 1)
End If
If Val(lblOutPut) = 0 Then
    lblOutPut = "0."
    blnpoint = False
End If
strLastinput = " "
End Sub
```

فشار دادن دکمه‌ی BackSpace به معنی حذف آخرین رقم وارد شده در خروجی است. در این رویداد ابتدا متغیر تعریف شده و سپس طول برجسب خروجی محاسبه می‌شود. اگر این طول از صفر بیشتر بود، اجازه‌ی حذف رقم داده می‌شود و در غیر این صورت، خروجی "0." را نشان داده و blnpoint با False مقداردهی شده و strLastInput نیز خالی می‌شود. ولی اگر طول بزرگ‌تر از صفر بود، تابع Left از سمت چپ برجسب خروجی، به اندازه‌ی یکی کمتر از طول آن را برداشته و در خروجی نمایش می‌دهد. در این صورت آخرین عدد وارد شده حذف می‌شود. این عمل به وسیله‌ی عبارت lblOutPut=Left(lblOutPut,L-1) انجام شده است.

برنامه را اجرا کنید و کار خود را بررسی کنید.

۶-۱-۴ مرحله‌ی چهارم طراحی

مطابق شکل ۴-۶، به برنامه‌ی خود چهار کلید برای توابع مثلثاتی Sin، Cos، Tan، Cot و شش کلید برای عملیات دیگر اضافه کنید.



شکل ۶-۴

برای کلیدهای اضافه شده مطابق جدول ۶-۱ نام تعیین کنید.

جدول ۶-۱

| کلید | نام کلید |
|------|------------|
| Sin | cmdSin |
| Cos | cmdCos |
| Tan | cmdTan |
| Cot | cmdCot |
| PI | cmdPI |
| Log | cmdLog |
| Sqrt | cmdSqrt |
| X^2 | CmdPower2 |
| X^Y | cmdPowerxy |
| 1/X | CmdReverse |

در بخش General متغیری به نام dbIPi تعریف کنید تا بتوانید عدد پی را با دقت بالا در آن ذخیره کنید:

Dim dbIPi As Double

در رویداد Load، فرمان زیر را در سطر آخر اضافه کنید:

$$\text{dbIPi} = 4 * \text{Atn}(1)$$

تابع $\text{Atn}(1)$ معادل ریاضی تانژانت معکوس عدد ۱ است و بدین معناست که زاویه‌ی معادل تانژانت معکوس ۱ را بازگرداند. این زاویه برحسب رادیان و معادل $\frac{\pi}{4}$ است که اگر در عدد ۴ ضرب شود، عدد پی را با دقت بالا باز می‌گرداند (توضیحات بیشتر را به هنرآموز محترم واگذار می‌کنیم).

این فرمان عدد پی را در متغیر dblPi ذخیره می‌کند تا در محاسبات بعدی استفاده شود. بر روی دکمه‌ی Sin دوبار کلیک کرده و دستورهای زیر را بنویسید:

1. **Private Sub** cmdSin_Click()
2. **Dim** dblR **As Double**
3. $\text{dblR} = \text{Val}(\text{lblOutput}) / 180 * \text{dblPi}$
4. $\text{lblOutput} = \text{Sin}(\text{dblR})$
5. **If** $\text{Val}(\text{lblOutput}) < 1\text{E}-16$ **Then** $\text{lblOutput} = "0."$
6. $\text{strLastinput} = \text{"nums"}$
7. **End Sub**

همان‌طور که می‌دانید توابع مثلثاتی با زوایای برحسب رادیان کار می‌کنند. در این رویداد ابتدا متغیری برای ذخیره‌ی زاویه برحسب رادیان معرفی شده و سپس به وسیله‌ی فرمول $R = D/180 * 3.14$ زاویه از درجه به رادیان تبدیل می‌شود. در سطر ۳ برنامه این عمل انجام می‌شود. سپس در سطر ۴ حاصل Sin زاویه محاسبه و در برحسب خروجی ذخیره می‌شود. در سطر ۵ این مقدار با عدد $1\text{E}-16$ مقایسه شده و اگر کوچک‌تر بود، صفر فرض می‌شود. زیرا متغیر نوع Double در تعداد ارقام بعد از ممیز محدودیت دارد. در سطر ۶ نیز متغیر strLastInput با رشته‌ی "nums" مقداردهی می‌شود زیرا نتیجه‌ی محاسبه بلافاصله در خروجی دیده می‌شود.

نکته

همان‌طور که قبلاً نیز بیان شد بعضی از عملگرها برای محاسبه فقط به یک عملوند نیاز دارند، مثل Sin، Cos و ...

برای توابع مثلثاتی دیگر نیز به همین صورت عمل کنید. متن برنامه‌ی ۳-۶ را ببینید.

برنامه‌ی ۳-۶

```

Private Sub cmdCos_Click()
Dim dblR As Double
dblR = Val(lblOutPut) / 180 * dblPi
lblOutPut = Cos(dblR)
If Val(lblOutPut) < 1E-16 Then lblOutPut = "0."
strLastinput = "nums"
End Sub
Private Sub cmdCot_Click()
Dim dblR As Double
dblR = Val(lblOutPut) / 180 * dblPi
If dblR = 0 Then
    MsgBox "Cannot Division by zero", vbCritical, "Error"
    Exit Sub
End If
lblOutPut = 1 / Tan(dblR)
If Val(lblOutPut) < 1E-16 Then lblOutPut = "0."
If Val(lblOutPut) > 1E+16 Then MsgBox "Invalid input
function", vbCritical, "Error"
strLastinput = "nums"
End Sub
Private Sub cmdtan_Click()
Dim dblR As Double
dblR = Val(lblOutPut) / 180 * dblPi
lblOutPut = Tan(dblR)
If Val(lblOutPut) < 1E-16 Then lblOutPut = "0."
If Val(lblOutPut)>1E+16 Then MsgBox "Invalid input function",
vbCritical, "Error"
    strLastinput = "nums"
End Sub

```

توجه

در رویدادهای مربوط به توابع تانژانت و کتانژانت به دلیل محدود بودن مقادیر عددی متغیر Double پس از محاسبه، مقدار به دست آمده را با عدد $1e+16$ نیز مقایسه می‌کنیم و اگر پاسخ بزرگ‌تر بود پیام خطا صادر می‌شود، زیرا در ریاضیات بی‌نهایت وجود دارد ولی در متغیرهای عددی بی‌نهایت معنی ندارد.

برای دکمه‌های دیگر نیز به صورت زیر عمل کنید:

```
Private Sub cmdPi_Click()
```

```
    lblOutput = dblPi
```

```
    strLastinput = "nums"
```

```
End Sub
```

```
Private Sub cmdLog_Click()
```

```
Dim dblN As Double
```

```
dblN = Val(lblOutput)
```

```
If dblN <= 0 Then
```

```
    MsgBox "Invalid input function", vbCritical, "Error"
```

```
    Exit Sub
```

```
End If
```

```
lblOutput = Log(dblN) / Log(10)
```

```
If Val(lblOutput) < 1E-16 Then lblOutput = "0."
```

```
If Val(lblOutput) > 1E+16 Then MsgBox "Invalid input  
function", vbCritical, "Error"
```

```
strLastinput = "op"
```

```
End Sub
```

```
Private Sub cmdSqrt_Click()
```

```
Dim dblN As Double
```

```
dblN = Val(lblOutput)
```



```

If dblN < 0 Then
    MsgBox "Invalid input function", vbCritical, "Error"
    Exit Sub
End If
lblOutPut = Sqr(dblN)
If Val(lblOutPut) < 1E-16 Then lblOutPut = "0."
If Val(lblOutPut) > 1E+16 Then MsgBox "Invalid input
function", vbCritical, "Error"
strLastinput = "op"
End Sub

Private Sub cmdPower2_Click()
Dim dblN As Double
dblN = Val(lblOutPut)
lblOutPut = dblN ^ 2
If Val(lblOutPut) < 1E-16 Then lblOutPut = "0."
If Val(lblOutPut) > 1E+16 Then MsgBox "Invalid input
function", vbCritical, "Error"
strLastinput = "op"
End Sub

Private Sub cmdPowerxy_Click()
If strLastinput = "nums" Then
    n1 = Calc(n1, Val(lblOutPut), op)
    lblOutPut = n1
End If
n1 = Val(lblOutPut)
op = "^"
strLastinput = "op"

```

```

blnpoint = False
End Sub

Private Sub cmdReverse_Click()
Dim dblN As Double
dblN = Val(lblOutPut)
If dblN = 0 Then
    MsgBox "Cannot Division by zero", vbCritical, "Error"
    Exit Sub
End If
lblOutPut = 1 / dblN
If Val(lblOutPut) < 0.000000000000001 Then lblOutPut = "0."
strLastinput = "op"
End Sub

```

توجه

از بین این عملگرها فقط عملگر x^y برای محاسبه به دو عملوند نیاز دارد یعنی مثل عملگر $*$ ، در نتیجه ساختار رویداد آن مثل عملگر $*$ است و لازم است در تابع Calc علامت $^$ برای توان اضافه شود.

لیست اصلاح شده‌ی این تابع به صورت زیر است:

1. **Private Function Calc(a As Double, b As Double, op As String) As Double**
2. **Select Case op**
3. **Case "*"**
4. Calc = a * b
5. **Case "-"**
6. Calc = a - b
7. **Case "+"**

```

8.      Calc = a + b
9.      Case "^"
10.     If a < 0 And b - Fix(b) <> 0 Then
11.         MsgBox "Invalid input function", vbCritical, "Error"
12.     Else
13.         Calc = a ^ b
14.     End If
15. Case "/"
16.     If b = 0 Then
17.         MsgBox "Cannot Division by zero", vbCritical,
                "Error"
18.     Else
19.         Calc = a / b
20.     End If
21. Case " "
22.     Calc = b
23. End Select
24. End Function

```

در این تابع در سطر ۱۰ اولین عدد بررسی می‌شود. اگر این عدد منفی و عدد دوم اعشاری باشد، پیام خطا صادر می‌شود زیرا یک عدد منفی را نمی‌توان به توان اعشاری رساند. برای تشخیص اعشاری بودن یک عدد لازم است که عدد اعشاری از بخش صحیح همان عدد کم شود و اگر نتیجه مخالف صفر بود به این معنی است که عدد اعشاری است.

$3.51 - \text{Fix}(3.51) = 3.51 - 3 = 0.51 \Rightarrow$ عدد اعشاری است

$3.0 - \text{Fix}(3.0) = 3.0 - 3.0 = 0 \Rightarrow$ عدد صحیح است

در سطر ۱۶ نیز عدد دوم با صفر مقایسه می‌شود زیرا در تقسیم دو عدد بر هم، عدد دوم یا همان مخارج کسر نباید صفر باشد. در نتیجه اگر عدد دوم صفر بود، پیام خطا صادر می‌شود و تقسیم انجام نمی‌شود.

توجه

در عملگر $\frac{1}{X}$ نیز نباید مخرج صفر شود که در رویداد مربوطه به این نکته توجه شده است.

برنامه را اجرا کنید. مشاهده می‌کنید که یک ماشین حساب تقریباً کامل را طراحی کرده‌اید و می‌توانید از آن برای کاربردهای عمومی خود استفاده کنید.

متن کامل برنامه‌ی ۴-۶ را ببینید:

برنامه‌ی ۴-۶

Option Explicit

Dim blnpoint As Boolean

Dim op As String * 1, strLastinput As String

Dim n1 As Double, dblM As Double

Dim dblPi As Double

Private Function Calc(a As Double, b As Double, op As String) As Double
Select Case op

Case "*"

Calc = a * b

Case "-"

Calc = a - b

Case "+"

Calc = a + b

Case "^"

If a < 0 And b - Fix(b) <> 0 Then

MsgBox "Invalid input function", vbCritical, "Error"

Else

Calc = a ^ b

End If

```
Case "/"
    If b = 0 Then
        MsgBox "Cannot Division by zero", vbCritical, "Error"
    Else
        Calc = a / b
    End If
Case " "
    Calc = b
End Select
End Function

Private Sub cmd1_Click()
If Len(lblOutPut) = 20 Then Exit Sub
If strLastinput <> "nums" Then
    lblOutPut = IIf(blnpoint, lblOutPut + "1", "1")
Else
    lblOutPut = lblOutPut + "1"
End If
strLastinput = "nums"
End Sub

Private Sub cmd2_Click()
If Len(lblOutPut) = 20 Then Exit Sub
If strLastinput <> "nums" Then
    lblOutPut = IIf(blnpoint, lblOutPut + "2", "2")
Else
    lblOutPut = lblOutPut + "2"
End If
strLastinput = "nums"
End Sub
```

```
Private Sub cmd3_Click()  
If Len(lblOutPut) = 20 Then Exit Sub  
If strLastinput <> "nums" Then  
    lblOutPut = IIf(blnpoint, lblOutPut + "3", "3")  
Else  
    lblOutPut = lblOutPut + "3"  
End If  
strLastinput = "nums"  
End Sub  
  
Private Sub cmd4_Click()  
If Len(lblOutPut) = 20 Then Exit Sub  
If strLastinput <> "nums" Then  
    lblOutPut = IIf(blnpoint, lblOutPut + "4", "4")  
Else  
    lblOutPut = lblOutPut + "4"  
End If  
strLastinput = "nums"  
End Sub  
  
Private Sub cmd5_Click()  
If Len(lblOutPut) = 20 Then Exit Sub  
If strLastinput <> "nums" Then  
    lblOutPut = IIf(blnpoint, lblOutPut + "5", "5")  
Else  
    lblOutPut = lblOutPut + "5"  
End If  
strLastinput = "nums"  
End Sub
```

```
Private Sub cmd6_Click()  
If Len(lblOutPut) = 20 Then Exit Sub  
If strLastinput <> "nums" Then  
    lblOutPut = IIf(blnpoint, lblOutPut + "6", "6")  
Else  
    lblOutPut = lblOutPut + "6"  
End If  
strLastinput = "nums"  
End Sub  
  
Private Sub cmd7_Click()  
If Len(lblOutPut) = 20 Then Exit Sub  
If strLastinput <> "nums" Then  
    lblOutPut = IIf(blnpoint, lblOutPut + "7", "7")  
Else  
    lblOutPut = lblOutPut + "7"  
End If  
strLastinput = "nums"  
End Sub  
  
Private Sub cmd8_Click()  
If Len(lblOutPut) = 20 Then Exit Sub  
If strLastinput <> "nums" Then  
    lblOutPut = IIf(blnpoint, lblOutPut + "8", "8")  
Else  
    lblOutPut = lblOutPut + "8"  
End If  
strLastinput = "nums"  
End Sub
```

```
Private Sub cmd9_Click()  
If Len(lblOutPut) = 20 Then Exit Sub  
If strLastinput <> "nums" Then  
    lblOutPut = IIf(blnpoint, lblOutPut + "9", "9")  
Else  
    lblOutPut = lblOutPut + "9"  
End If  
strLastinput = "nums"  
End Sub  
  
Private Sub cmd0_Click()  
If Len(lblOutPut) = 20 Then Exit Sub  
If strLastinput <> "nums" Then  
    lblOutPut = IIf(blnpoint, lblOutPut + "0", "0.")  
Else  
    lblOutPut = lblOutPut + "0"  
End If  
strLastinput = "nums"  
End Sub  
  
Private Sub cmdC_Click()  
    n1 = 1  
    op = " "  
    lblOutPut = "0."  
    strLastinput = " "  
    blnpoint = False  
End Sub  
  
Private Sub cmdCE_Click()  
    lblOutPut = "0."
```



```
strLastinput = "op"
```

```
blnpoint = False
```

```
End Sub
```

```
Private Sub cmdDiv_Click()
```

```
If strLastinput = "nums" Then
```

```
    n1 = Calc(n1, Val(lblOutPut), op)
```

```
    lblOutPut = n1
```

```
End If
```

```
n1 = Val(lblOutPut)
```

```
op = "/"
```

```
strLastinput = "op"
```

```
blnpoint = False
```

```
End Sub
```

```
Private Sub cmdEqual_Click()
```

```
    n1 = Calc(n1, Val(lblOutPut), op)
```

```
    If n1 = 0 Then
```

```
        lblOutPut = "0."
```

```
    Else
```

```
        lblOutPut = n1
```

```
    End If
```

```
    op = " "
```

```
    strLastinput = "op"
```

```
    blnpoint = False
```

```
End Sub
```

```
Private Sub cmdMC_Click()
```

```
    dblM = 0
```

```
    lblM = " "
```

```
strLastinput = " "
```

End Sub

Private Sub cmdMinus_Click()

If strLastinput = "nums" **Then**

```
n1 = Calc(n1, Val(lblOutPut), op)
```

```
lblOutPut = n1
```

End If

```
n1 = Val(lblOutPut)
```

```
op = "-"
```

```
strLastinput = "op"
```

```
blnpoint = False
```

End Sub

Private Sub cmdMP_Click()

```
dblM = dblM + Val(lblOutPut)
```

```
lblM = "M"
```

```
strLastinput = "op"
```

End Sub

Private Sub cmdMR_Click()

```
lblOutPut = IIf(dblM = 0, "0.", dblM)
```

```
strLastinput = "nums"
```

```
If InStr(lblOutPut, ".") <> 0 Then blnpoint = True
```

End Sub

Private Sub cmdMS_Click()

```
dblM = Val(lblOutPut)
```

```
lblM = "M"
```

```
strLastinput = "op"
```

```
    blnpoint = False
```

```
End Sub
```

```
Private Sub cmdMul_Click()
```

```
If strLastinput = "nums" Then
```

```
    n1 = Calc(n1, Val(lblOutPut), op)
```

```
    lblOutPut = n1
```

```
End If
```

```
n1 = Val(lblOutPut)
```

```
op = "*" 
```

```
strLastinput = "op"
```

```
blnpoint = False
```

```
End Sub
```

```
Private Sub cmdPlus_Click()
```

```
If strLastinput = "nums" Then
```

```
    n1 = Calc(n1, Val(lblOutPut), op)
```

```
    lblOutPut = n1
```

```
End If
```

```
n1 = Val(lblOutPut)
```

```
op = "+" 
```

```
strLastinput = "op"
```

```
blnpoint = False
```

```
End Sub
```

```
Private Sub cmdpoint_Click()
```

```
    blnpoint = True
```

```
    If strLastinput = "op" Then
```

```
        lblOutPut = "0."
```

```
    Exit Sub
```

```

End If
If InStr(lblOutPut, ".") = 0 Then
    lblOutPut = lblOutPut + "."
End If
    strLastinput = "nums"
End Sub

Private Sub cmdPi_Click()
    lblOutPut = dblPi
    strLastinput = "nums"
End Sub

Private Sub cmdLog_Click()
Dim dblN As Double
dblN = Val(lblOutPut)
If dblN <= 0 Then
    MsgBox "Invalid input function", vbCritical, "Error"
    Exit Sub
End If
    lblOutPut = Log(dblN) / Log(10)
If Val(lblOutPut) < 1E-16 Then lblOutPut = "0."
If Val(lblOutPut) > 1E+16 Then MsgBox "Invalid input
function", vbCritical, "Error"
    strLastinput = "op"
End Sub

Private Sub cmdSqrt_Click()
Dim dblN As Double
dblN = Val(lblOutPut)
If dblN < 0 Then

```

```

    MsgBox "Invalid input function", vbCritical, "Error"
Exit Sub
End If
lblOutPut = Sqr(dblN)
If Val(lblOutPut) < 1E-16 Then lblOutPut = "0."
If Val(lblOutPut) > 1E+16 Then MsgBox "Invalid input
function", vbCritical, "Error"
strLastinput = "op"
End Sub

Private Sub cmdPower2_Click()
Dim dblN As Double
dblN = Val(lblOutPut)
lblOutPut = dblN ^ 2
If Val(lblOutPut) < 1E-16 Then lblOutPut = "0."
If Val(lblOutPut) > 1E+16 Then MsgBox "Invalid input
function", vbCritical, "Error"
strLastinput = "op"
End Sub

Private Sub cmdPowerxy_Click()
If strLastinput = "nums" Then
    n1 = Calc(n1, Val(lblOutPut), op)
    lblOutPut = n1
End If
n1 = Val(lblOutPut)
op = "^"
strLastinput = "op"
blnpoint = False
End Sub

```

```
Private Sub cmdReverse_Click()  
Dim dblN As Double  
dblN = Val(lblOutPut)  
If dblN = 0 Then  
    MsgBox "Cannot Division by zero", vbCritical, "Error"  
    Exit Sub  
End If  
lblOutPut = 1 / dblN  
If Val(lblOutPut) < 0.000000000000001 Then lblOutPut = "0."  
strLastinput = "op"  
End Sub  
  
Private Sub cmdSign_Click()  
If lblOutPut = "0." And blnpoint = False Then  
    Exit Sub  
End If  
If lblOutPut = "0." Then  
    lblOutPut = "-0."  
Else  
    lblOutPut = -Val(lblOutPut)  
End If  
End Sub  
  
Private Sub cmdBks_Click()  
Dim L As Byte  
L = Len(lblOutPut)  
If L > 0 Then  
    lblOutPut = Left(lblOutPut, L - 1)  
End If
```

If Val(lblOutPut) = 0 Then

 lblOutPut = "0."

 blnpoint = **False**

End If

 strLastinput = " "

End Sub

Private Sub cmdSin_Click()

Dim dblR As Double

dblR = Val(lblOutPut) / 180 * dblPi

lblOutPut = **Sin**(dblR)

If Val(lblOutPut) < 1E-16 Then lblOutPut = "0."

strLastinput = "nums"

End Sub

Private Sub cmdCos_Click()

Dim dblR As Double

dblR = Val(lblOutPut) / 180 * dblPi

lblOutPut = **Cos**(dblR)

If Val(lblOutPut) < 1E-16 Then lblOutPut = "0."

strLastinput = "nums"

End Sub

Private Sub cmdCot_Click()

Dim dblR As Double

dblR = Val(lblOutPut) / 180 * dblPi

If dblR = 0 Then

MsgBox "Cannot Division by zero", **vbCritical**, "Error"

Exit Sub

End If

lblOutPut = 1 / **Tan**(dblR)

```

If Val(lblOutPut) < 1E-16 Then lblOutPut = "0."
If Val(lblOutPut) > 1E+16 Then MsgBox "Invalid input
function", vbCritical, "Error"
strLastinput = "nums"
End Sub

Private Sub cmdtan_Click()
Dim dblR As Double
dblR = Val(lblOutPut) / 180 * dblPi
lblOutPut = Tan(dblR)
If Val(lblOutPut) < 1E-16 Then lblOutPut = "0."
If Val(lblOutPut) > 1E+16 Then MsgBox "Invalid input
function", vbCritical, "Error"
strLastinput = "nums"
End Sub

Private Sub Form_Load()
n1 = 1
op = " "
strLastinput = " "
lblM = " "
dblPi = 4 * Atn(1)
End Sub

```

۶-۲ نمایش تصویر

پروژه‌ای که به شرح گام به گام آن می‌پردازیم، یک برنامه‌ی نمایش تصویر است که قابلیت‌های زیر را داراست:

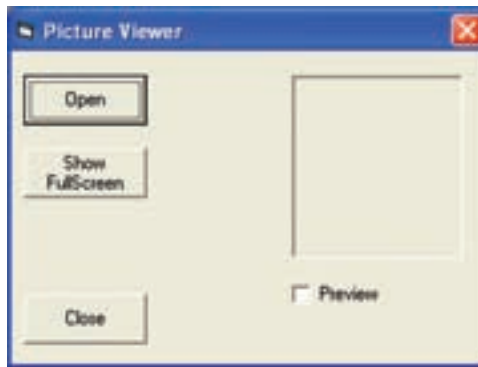
- استفاده از پنجره‌ی استاندارد برای باز کردن پرونده‌های تصویری
- قابلیت انتخاب و تعیین نوع پرونده گرافیکی
- نمایش عکس در حالت پیش‌نمایش و حذف این حالت

- نمایش عکس در اندازه‌ی واقعی در یک فرم دیگر

در این پروژه از یک شیء به نام **CommonDialog** برای نمایش پنجره‌ی استاندارد **Open** استفاده شده است. این شیء در حالت عادی در جعبه‌ابزار خود ویژگی‌ها و بیسیک وجود ندارد و لازم است از پنجره‌ی **Components** انتخاب و به جعبه‌ابزار اضافه شود که نحوه‌ی انجام این کار شرح داده خواهد شد. لازم به ذکر است که در ویژگی‌ها بیسیک بسیاری از شیء‌ها یا کنترل‌های اضافه را می‌توان مورد استفاده قرار داد به شرط آنکه ابتدا از پنجره‌ی **Components** انتخاب شده و به جعبه‌ابزار اضافه شوند. در درس‌های برنامه‌سازی ۲ و ۳ با شیء‌های دیگر و نحوه‌ی استفاده از آنها آشنا خواهید شد.

مراحل کار:

پروژه‌ی جدیدی ایجاد کرده و فرمی مانند شکل ۵-۶ در آن طراحی کنید.



شکل ۵-۶

روی فرم، کلیدهایی به اسمی `cmdOpen`، `cmdShowFull` و `cmdClose` قرار دهید. یک کنترل `Image` بر روی فرم قرار داده و مشخصه‌های زیر را برای آن تنظیم کنید:

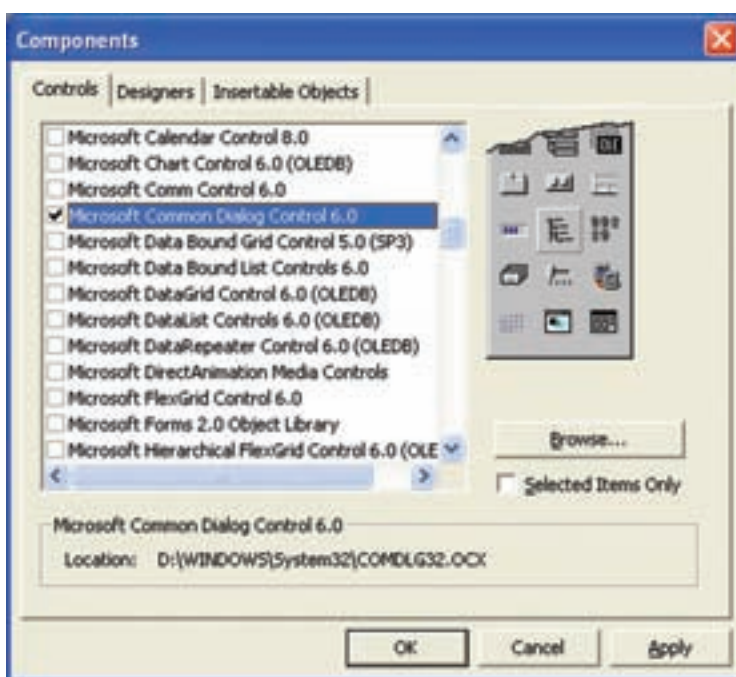
`BorderStyle=1-Fixed single`

`Stretch=True`

- یک کادر علامت به نام `chkPreview` در فرم قرار دهید.

- کد مربوط به دکمه‌ی `cmdClose` را بنویسید.

- روی فرم دوبار کلیک کرده و فرمان `chkPreview.Value=1` را در رویداد `Load` مربوط به فرم بنویسید.
- از منوی `Project` گزینه‌ی `Components` را انتخاب کرده یا کلید ترکیبی `Ctrl+T` را فشار دهید. در این صورت کادری مانند شکل ۶-۶ نمایش می‌یابد. با حرکت نوار لغزان به سمت پایین، گزینه‌ی `Microsoft Common Dialog Control 6.0` را یافته و انتخاب کنید.



شکل ۶-۶

روی `OK` کلیک کنید. در این صورت کنترل مربوطه مانند شکل ۶-۷ به انتهای جعبه‌ایزار اضافه خواهد شد.

روی کنترل `CommonDialog` دوبار کلیک کنید تا یک کپی از آن به وسط فرم منتقل شود. باید توجه داشت که این کنترل در هنگام اجرا دیده نمی‌شود، بلکه فقط دارای یک سری متد می‌باشد که لازم است آنها را فراخوانی کنیم تا اجرا شوند. در آینده با متدهای مهم این شیء آشنا خواهیم شد.



شکل ۶-۷

روی دکمه‌ی Open دوبار کلیک کنید و در رویداد مربوطه دستورهای زیر را بنویسید:

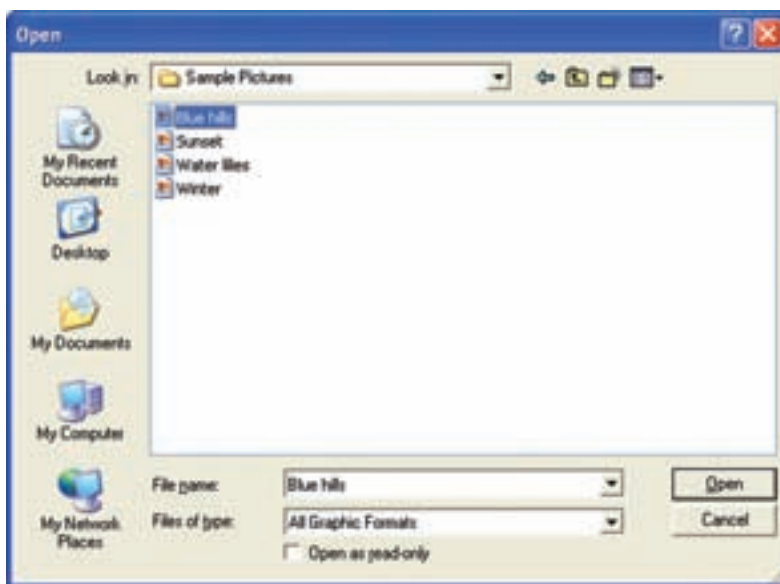
1. **Private Sub** cmdOpen_Click()
2. CommonDialog1.Filter = "All Graphic
3. Formats|*.jpg;*.gif;*.bmp;*.ico;*.wmf| Jpeg Format |*.jpg| Gif
4. Format |*.gif| Bitmap Format |*.bmp| _con Format |*.ico|
- Windows
5. Meta File Jpeg Format |*.wmf "
6. CommonDialog1.ShowOpen
7. **If** CommonDialog1.FileName <>" " **Then**
8. Image1.Picture = **LoadPicture**(CommonDialog1.FileName)
9. **End If**
10. **End Sub**

در این رویداد، در سطر ۲، ابتدا نوع پرونده‌های قابل نمایش به وسیله‌ی مشخصه‌ی Filter برای شیء CommonDialog تنظیم می‌شود.
شکل کلی مشخصه‌ی Filter به صورت زیر است:

CommonDialog1.Filter[=*description1|filter1|description2|filter2...]*

Description بیانگر یک عبارت رشته‌ای به عنوان پیام نوع پرونده و **Filter** نیز یک عبارت رشته‌ای نشان‌دهنده‌ی یک یا چند نوع پرونده است که باید به صورت *Type.** تعریف شود و **Type** همان نوع پرونده، مثل Jpg یا Gif و... است. در صورتی که بخواهیم چند نوع پرونده را با هم معرفی کنیم، باید بین آنها از علامت ؛ استفاده کنیم. در ضمن بین توضیحات و نوع پرونده از علامت Pipe یا | استفاده می‌کنیم.

در سطر ۶، متد ShowOpen از این شیء فراخوانی شده است تا پنجره‌ی استاندارد Open مانند شکل ۸-۶ نمایش یابد و کاربر می‌تواند با انتخاب نوع پرونده، لیست پرونده‌ها را مشاهده کرده و یکی از آنها را انتخاب کند.



شکل ۸-۶

اگر کاربر پرونده‌ای انتخاب کند، مشخصه‌ی FileName از این شیء، نام پرونده انتخابی را برمی‌گرداند. در سطر ۸، بررسی شده است که اگر پرونده‌ای انتخاب شده باشد، به وسیله‌ی تابع

LoadPicture تصویر موردنظر از حافظه خوانده شده و در کنترل Image قرار گیرد. به دلیل اینکه در این شیء، مشخصه `Stretch=True` شده است، تصویر به صورت پیش‌نمایش مشاهده می‌شود.

روی کادر علامت، دوبار کلیک کرده و در رویداد مربوطه، دستورهای زیر را بنویسید:

```
Private Sub chkPreview_Click()
    If chkPreview.Value = 1 Then
        Image1.Visible = True
    Else
        Image1.Visible = False
    End If
End Sub
```

از این کادر علامت، برای نمایش یا پنهان کردن کنترل Image استفاده می‌کنیم. منوی Project را باز کرده و گزینه Add Form را انتخاب و یک فرم جدید استاندارد را به پروژه‌ی خود اضافه کنید. نام این فرم را به `frmShow` تغییر دهید. مشخصه `BorderStyle` مربوط به فرم را روی عدد `1-Fixed single` تنظیم کنید و مشخصه `Caption` آن را نیز روی `Show Picture` تنظیم کنید. از پنجره‌ی پروژه، `frmShow` را انتخاب کنید و روی این فرم یک کنترل تصویر به نام `pic1` قرار دهید.

در رویداد Load مربوط به فرم `frmShow`، دستورهای زیر را بنویسید:

```
Private Sub Form_Load()
    Pic1.AutoSize = True
    Pic1.Left = 0
    Pic1.Top = 0
End Sub
```

فرمان `Pic1.AutoSize=True` سبب می‌شود که تصویر در اندازه‌ی واقعی خود نمایش یابد. در رویداد Activate مربوط به فرم `frmShow`، دستورهای زیر را بنویسید:

Private Sub Form_Activate()

```
frmShow.Width = Pic1.Width + 30
```

```
frmShow.Height = Pic1.Height + 405
```

End Sub

هرگاه این فرم فعال شود، این رویداد به‌طور خودکار اجرا می‌شود و این دو فرمان سبب می‌شوند که اندازه‌ی فرم با اندازه‌ی کنترل تصویر برابر شده و کمی هم بزرگ‌تر شود تا کل تصویر قابل مشاهده باشد. عدد ۴۰۵ به‌طور تقریب، برابر اندازه‌ی حاشیه‌ی بالای صفحه و ۳۰، اندازه‌ی ضخامت لبه‌های صفحه است که باید به‌اندازه‌ی اصلی اضافه شوند. مجدداً به فرم قبلی بازگشته و روی کلید cmdShowFull دوبار کلیک کنید و دستورهای زیر را بنویسید:

1. **Private Sub** cmdShowFull_Click()
2. frmShow.Pic1.Picture = Image1.Picture
3. frmShow.Show 1
4. **End Sub**

در سطر ۲، فرمان `frmShow.Pic1.Picture = Image1.Picture`، تصویر موجود در Image را به درون مشخصه‌ی Picture از کنترل تصویر در فرم frmShow منتقل می‌کند.

نکته

اگر بخواهیم به مشخصه‌ی یک کنترل که در یک فرم دیگر وجود دارد، دسترسی پیدا کنیم، لازم است قبل از نام کنترل، نام فرم مربوطه را بنویسیم؛ ولی اگر کنترل در فرم جاری باشد، نوشتن نام فرم ضروری نیست. در سطر ۲، نام فرم frmShow قبل از نام کنترل Pic1 نوشته شده است، زیرا این کنترل متعلق به آن فرم است.

در سطر ۳ متد Show از frmShow را با پارامتر ۱ فراخوانی می‌کنیم. این متد سبب نمایش فرم می‌شود. در نتیجه، تصویر درون آن هم مشاهده می‌شود.

متد Show دارای پارامتری به نام Style است و این پارامتر می‌تواند دارای دو مقدار ° یا ۱ باشد. در صورتی که این متد با پارامتر صفر فراخوانی شود، بدین معناست که پس از نمایش فرم، دستورهای بعد از این فرمان تا انتهای رویداد اجرا شوند؛ ولی اگر با عدد ۱ فراخوانی شود، بدین معناست که پس از نمایش فرم تا زمانی که فرم بسته نشده است، کنترل اجرا به فرم قبلی باز نمی‌گردد و دستورهای بعد از این فرمان اجرا نخواهند شد، مگر اینکه فرم نمایش یافته بسته شود. در نتیجه در این مثال، تا وقتی که frmShow در حال نمایش است، به فرم قبلی بازنگشته و کاربر نمی‌تواند تصویر دیگری را انتخاب کند.

تمرینات تکمیلی

۱. برنامه‌ای بنویسید که خروجی مقابل را تولید کند:

```
1
1 2 3
1 2 3 4 5
1 2 3 4 5 6 7
1 2 3 4 5
1 2 3
1
```

۲. برنامه‌ای بنویسید که X را بگیرد و حاصل عبارت زیر را تا 10° جمله حساب کند:

$$X - \frac{X^2}{2} + \frac{X^3}{3} - \frac{X^4}{4} + \dots$$

۳. برنامه‌ای بنویسید که یک عدد صحیح سه‌رقمی مثبت را از ورودی دریافت کرده و به صورت حروف چاپ کند. به عنوان مثال، عدد 100 را با پیام One Hundred و عدد 127 را با پیام One Hundred Twenty Seven نمایش دهد (فرض کنید که رقم دوم عدد، 1 نباشد).

۴. برنامه‌ای بنویسید که رشته‌ای شامل یک عبارت ریاضی را از ورودی دریافت کند، سپس حاصل آن را چاپ کند. به عنوان مثال، اگر کاربر رشته‌ی $"3*12="$ را وارد کرد، عدد 36 در خروجی چاپ شود.

توجه: رشته فقط دارای 4 عمل اصلی باشد.

پیوست

مرجع سریع دستورها و توابع ویژوال بیسیک

ویژوال بیسیک دارای بیش از ۲۰۰ دستور و تابع است. به خاطر سپردن همه‌ی آنها غیرممکن است. حتی برنامه‌نویسان حرفه‌ای نیز ممکن است شکل کلی بعضی از دستورها و توابع را ندانند. از این ضمیمه به عنوان راهنمای مرجع سریع برای دستورها و توابع بیان شده در این کتاب استفاده کنید. در این ضمیمه، شکل کلی هر دستور یا تابع، شرح مختصر آن و چگونگی استفاده از آرگومان‌ها (در صورت وجود) بیان خواهد شد.

توابع و دستورها

تابع Abs

Abs (*number*)

قدرمطلق مقدار *number* را برمی‌گرداند. نوع داده‌ی برگردانده‌شده، همان نوع داده‌ی آرگومان *number* است.

تابع Asc

Asc (*string*)

یک عدد صحیح را برمی‌گرداند. کد اسکی اولین کاراکتر *string* را برمی‌گرداند.

تابع Atn

Atn (*number*)

یک مقدار Double را برمی‌گرداند که آرک‌تانژانت *number* است.

دستور Beep

Beep

صدای بوق را از بلندگوی کامپیوتر پخش می‌کند. فرکانس و طول بوق، بستگی به نوع سیستم کامپیوتری دارد.

تابع Choose

Choose (*index*, *choice-1* [, *choice-2*, ... [, *choice-n*]])

بر اساس مقدار آرگومان *index* مقداری را از بین لیست گزینه‌ها (که با آرگومان‌های *choice-1* تا *choice-n* مشخص می‌شوند) برمی‌گرداند. اگر مقدار *index* برابر ۱ باشد، مقدار ارایه شده در *choice-1* و اگر برابر ۲ باشد، مقدار *choice-2* برگردانده می‌شود.

تابع Chr

Chr (*charcode*)

کاراکتر اسکپی معادل عدد آرگومان *charcode* را برمی‌گرداند.

دستور Const

[Public | Private] Const *constname* [As *type*] = *expression*

ثابتی به نام *constname* را تعریف کرده و مقدار آن را برابر با *expression* قرار می‌دهد. کلید واژه‌های Public و Private حوزه‌ی دید ثابت را تعیین می‌کنند که در برنامه‌سازی ۲ با آنها آشنا خواهید شد. *type* نوع داده‌ی ثابت را تعیین می‌کند. اگر این پارامتر اختیاری ذکر نشود، نوع داده‌ی ثابت با نوع *expression* متناسب خواهد بود.

تابع Cos

Cos (*number*)

مقدار Double که کسینوس زاویه‌ی تعیین‌شده در آرگومان *number* را برمی‌گرداند.

تابع CStr

CStr (*expression*)

مقدار *expression* را به نوع داده‌ی String تبدیل می‌کند.

دستور Dim

Dim [WithEvents] *varname* [([*subscripts*])] [As [New] *type*]

[, [WithEvents] *varname* [([*subscripts*])] [As [New] *type*]] ...

به کمک این دستور، می‌توان یک یا چند متغیر را تعریف کرد.

دستور Do...Loop

Do [{While | Until} condition]

[statements]

[Exit Do]

[statements]

Loop

یا

Do

[statements]

[Exit Do]

[statements]

Loop [{While | Until} condition]

یک یا چند *statements* را تا زمانی که *condition* برابر True باشد یا تا وقتی که True

شود، تکرار می‌کند. کلیدواژه‌ی اختیاری Exit Do کنترل برنامه را به خط بعد از ساختار Do

Loop منتقل می‌کند.

دستور End

End

End Function

End If

End Property

End Select

End Sub

End Type

End With

اجرای برنامه یا ساختاری را خاتمه می‌دهد.

دستور Exit

Exit Do

Exit For

Exit Function

Exit Property

Exit Sub

سبب خروج از یک روال یا ساختار حلقه می‌شود.

Exp تابع

Exp (*number*)

یک مقدار Double را برمی‌گرداند که e به توان *number* است.

Fix تابع

Fix (*number*)

قسمت صحیح عدد مشخص شده به وسیله‌ی آرگومان *number* را برمی‌گرداند. اگر عدد منفی باشد، این تابع اولین عدد صحیح منفی بزرگ‌تر یا مساوی با آرگومان را برمی‌گرداند.

For...Next دستور

For *counter* = start To end [Step *step*]

[*statements*]

[Exit For]

[*statements*]

Next [*counter*]

یک یا چند عبارت را به تعداد دفعات معین تکرار می‌کند. آرگومان *counter*، متغیر مورد استفاده برای افزایش از *start* به *end* است. به‌طور پیش‌فرض، *counter* هر بار که حلقه اجرا شود، یک واحد افزایش می‌یابد. ولی از آرگومان اختیاری *step* می‌توان برای تعیین مقدار افزایش یا کاهش استفاده کرد. از Exit For برای خروج از ساختار حلقه استفاده کنید.

GoTo دستور

GoTo line

کنترل برنامه را به خط خاصی از کد منتقل می‌کند.

Hex تابع

Hex (*number*)

یک مقدار String را که نشان دهنده‌ی مقدار شانزده‌تایی آرگومان *number* است، برمی‌گرداند.

دستور **If...Then...Else**

If condition Then [statements] [Else elsestatements]

یا

If *condition* Then

[*statements*]

[ElseIf *condition-n* Then

[*elseifstatements*] ...

[Else

[*elsestatements*]]

End If

اگر مقدار تعیین شده به وسیله‌ی *condition* برابر True باشد، یک یا چند *statements* اجرا می‌شود. در صورت نادرست بودن شرط، عبارت‌های بعد از Else اجرا می‌شوند. با استفاده از عبارت ElseIf می‌توان شرط‌های مختلفی را به صورت تودرتو بررسی کرد.

تابع **IIf**

IIf (*expression*, *truepart*, *falsepart*)

اگر مقدار *expression* برابر True باشد، تابع مقدار *truepart* و در صورتی که False باشد، *falsepart* را برمی‌گرداند.

تابع **InputBox**

InputBox(prompt[, title][, default][, xpos][, ypos][, ypos][,helpfile, context])

یک کادر محاوره‌ای را نمایش داده و منتظر دریافت متن از کاربر و کلیک روی دکمه‌ای می‌شود. آرگومان *prompt*، پیامی را که باید نمایش داده شود، تعیین می‌کند و *title* یک عنوان اختیاری برای نوار عنوان کادر محاوره‌ای مشخص می‌کند. آرگومان اختیاری *default* مقدار پیش‌فرضی را مشخص می‌کند و در صورتی که کاربر مقداری وارد نکند، تابع این مقدار پیش‌فرض را برمی‌گرداند.

آرگومان‌های اختیاری *xpos* و *ypos* محل افقی و عمودی کادر محاوره‌ای روی صفحه را تعیین می‌کنند.

آرگومان‌های اختیاری *helpfile* و *context* راهنمایی را برای کادر محاوره‌ای ارائه می‌کنند.

تابع InStr

InStr ([start,] string1, string2 [, compare])

موقعیت اولین مورد از یک رشته را در درون رشته‌ی دیگر برمی‌گرداند. آرگومان *start* محل شروع را مشخص می‌کند (پیش‌فرض، ۱ است). آرگومان اختیاری *compare* نوع مقایسه‌ی دو رشته را تعیین می‌کند (° برای دودویی و ۱ برای متنی، بدون در نظر گرفتن بزرگی و کوچکی حروف).

تابع Int

Int (number)

قسمت صحیح عدد *number* را برمی‌گرداند. اگر آرگومان منفی باشد، اولین عدد صحیح منفی کوچک‌تر یا مساوی *number* برگردانده می‌شود.

تابع LCase

LCase (string)

تمام نویسه‌های آرگومان رشته‌ای را به حروف کوچک تبدیل کرده و برمی‌گرداند.

تابع Left

Left (string, length)

به تعداد، *length* از سمت چپ *string* نویسه جدا کرده و برمی‌گرداند.

تابع Len

Len (string | varname)

یک مقدار Long برمی‌گرداند که تعداد *string* نویسه‌های یا تعداد بایت‌های موردنیاز برای ذخیره‌ی یک متغیر خاص (*varname*) را مشخص می‌کند.

دستور Let

[Let] varname = expression

مقدار *expression* را به یک متغیر (*varname*) انتساب می‌دهد. معمولاً از نوشتن این کلیدواژه صرف نظر می‌شود و ویژوال بیسیک آن را در نظر می‌گیرد.

تابع LoadPicture

LoadPicture (*[stringexpression]*)

تصویر تعیین شده در آرگومان را فراخوانی کرده و برمی‌گرداند. برای بارگذاری و انتساب تصاویر به مشخصه‌ی Picture فرم‌ها، کنترل PictureBox یا Image از این تابع استفاده می‌شود. اگر این تابع بدون آرگومان به کار رود، یک تصویر خالی را برمی‌گرداند.

تابع Log

Log (*number*)

یک مقدار Double برمی‌گرداند که مشخص‌کننده‌ی لگاریتم آرگومان است.

تابع LTrim

LTrim (*string*)

فضاهای خالی سمت چپ رشته را حذف کرده و رشته را برمی‌گرداند.

تابع Mid

Mid (*string*, *start* [, *length*])

یک زیررشته از رشته‌ی اصلی جدا کرده و برمی‌گرداند.

آرگومان *string* رشته‌ی اصلی است.

آرگومان *start* محل شروع زیررشته است.

آرگومان *length* طول زیررشته است. اگر این آرگومان ذکر نشود، تا انتهای رشته‌ی اصلی

برگردانده می‌شود.

دستور Mid

Mid(*stringvar*, *start* [, *length*]) = *string*

در یک متغیر رشته‌ای (*stringvar*) یک یا چند نویسه را با رشته‌ی دیگری (*string*) جایگزین می‌کند. آرگومان *start* محل شروع جایگزینی و *length* طول رشته‌ی جایگزین را تعیین می‌کنند.

تابع MsgBox

MsgBox(prompt[, buttons][, title][, helpfile, context])

پیامی را در یک کادر محاوره‌ای که دارای یک یا چند دکمه است، نمایش می‌دهد و منتظر پاسخ کاربر می‌ماند. سپس این تابع، یک مقدار Integer را برمی‌گرداند که مشخص‌کننده‌ی دکمه‌ی کلیک‌شده است.

آرگومان *prompt* پیام موردنظر است.

آرگومان *title* عنوان کادر محاوره‌ای را تعیین می‌کند.

آرگومان *buttons* دکمه‌هایی را که نمایش داده خواهند شد، تعیین می‌کنند.

آرگومان‌های *helpfile* و *context* راهنمایی را برای کادر محاوره‌ای ارایه می‌کنند.

تابع Oct

Oct (*number*)

یک مقدار رشته‌ای را برمی‌گرداند که معادل مبنای ۸ آرگومان است.

دستور Option Compare

Option Compare [Binary | Text | Database]

روش پیش‌فرض برای مقایسه‌ی رشته‌ها را تعیین می‌کند. این دستور فقط می‌تواند در سطح مدول مورد استفاده قرار گیرد.

دستور Option Explicit

Option Explicit

با اجرای این دستور، برنامه‌نویس باید تمام متغیرهای خود را به‌طور صریح اعلان کند. اگر این دستور اجرا نشود، متغیرهای اعلان‌نشده از نوع Variants در نظر گرفته خواهند شد.

دستور Rem

Rem *comments*

به کمک این دستور، می‌توان توضیحاتی را به برنامه اضافه کرد. هر چیزی که بعد از این دستور نوشته شود، به‌وسیله‌ی ویژوال بیسیک صرف‌نظر می‌شود. به‌جای این دستور می‌توان از علامت آپستروف (') نیز استفاده کرد.

تابع Right

Right (*string*, *length*)

یک زیررشته را از سمت راست رشته‌ی *string* به طول *length* جدا کرده و برمی‌گرداند.

تابع Rnd

Rnd[(*number*)]

یک مقدار Single برمی‌گرداند که عدد تصادفی بین صفر و یک است.

تابع RTrim

RTrim (*string*)

فضاهای خالی سمت راست رشته را حذف کرده و رشته را برمی‌گرداند.

دستور SavePicture

SavePicture *picture*, *stringexpression*

یک تصویر گرافیکی را از مشخصه‌ی Picture یا Image شیء، در یک فایل ذخیره می‌کند.

دستور Select Case

Select Case *testexpression*[*Case expressionlist-n*[*statements-n*]] ...

[Case Else

[*elsestatements*]]

End Select

این دستور یکی از شرط‌های متعدد را انتخاب می‌کند. تعداد دستورهای Case، به تعداد شرط‌های مسأله بستگی دارد. اگر هیچ‌یک از شرط‌های مسأله واقع نشوند، دستورهای قسمت Case Else اجرا خواهند شد. توجه داشته باشید که نوع داده‌ی عبارت موردنظر (در قسمت Select Case) باید با مقادیر (در قسمت‌های Case) یکسان باشد.

دستور SendKeys

یک یا چند ضربه‌کلید را تولید می‌کند.

تابع Sgn

Sgn(*number*)علامت *number* را برمی‌گرداند.

تابع Sin

Sin (*number*)سینوس زاویه‌ی *number* را برمی‌گرداند.

تابع Space

Space (*number*)رشته‌ای را برمی‌گرداند که شامل فضاهای خالی است و طول آن به تعداد *number* است.

تابع Spc

Spc(*n*)هنگامی که از متد Print استفاده می‌کنید، به کمک این تابع می‌توانید تعدادی فضای خالی (*n*) درج کنید.

تابع Sqr

Sqr (*number*)یک مقدار Double برمی‌گرداند که ریشه‌ی دوم (جذر) *number* است.

تابع Str

Str (*number*)

آرگومان را به رشته‌ی معادل تبدیل می‌کند.

تابع StrComp

StrComp (*string1*, *string2*[, *compare*])نتیجه‌ی مقایسه‌ی دو رشته را برمی‌گرداند. آرگومان *compare* چگونگی مقایسه‌ی دو رشته را به صورت دودویی و متنی تعیین می‌کند.

تابع StrConv

StrConv (*string*, *conversion*)

رشته‌ای را بر اساس آرگومان *conversion* تبدیل کرده و برمی‌گرداند. آرگومان *conversion* می‌تواند یکی از مقادیر ثابت مثل *vbProperCase* و *vbUpperCase*، *vbLowerCase* باشد.

تابع String

String (*number*, *character*)

رشته‌ای به طول *number* و با *character* تعیین شده را برمی‌گرداند.

تابع Switch

Switch (*nexpr-1*, *value-1* [, *expr-2*, *value-2* ... [, *expr-n*, *value-n*])

لیستی از عبارتها را ارزیابی کرده و یک مقدار *variant* برمی‌گرداند یا مقدار مربوط به اولین عبارتی را که درست است، برمی‌گرداند.

تابع Tab

Tab(*n*)

هنگام استفاده از متد *Print*، خروجی را به ستون موردنظر (*n*) منتقل می‌کند.

تابع Tan

Tan (*number*)

یک مقدار *Double* برمی‌گرداند که تانژانت زاویه‌ی آرگومان است.

تابع Trim

Trim (*string*)

تمام فضاهای خالی رشته را حذف کرده و رشته را برمی‌گرداند.

تابع UCase

UCase (*string*)

تمام نویسه‌های رشته را به حروف بزرگ تبدیل می‌کند.

دستور Unload

Unload object

یک شیء مثل فرم یا کنترل را از حافظه خارج کرده و منابع مورد استفاده به وسیله‌ی شیء را آزاد می‌کند.

تابع Val

Val (*string*)

مقدار عددی رشته‌ی آرگومان را برمی‌گرداند. نوع داده‌ای که برگردانده می‌شود، بستگی به نوع مقدار عددی رشته دارد. اگر رشته شامل هیچ مقدار عددی نباشد، تابع مقدار صفر را برمی‌گرداند.

دستور While...Wend

While *condition*

[*statements*]

Wend

یک یا چند عبارت (*statements*) را تا زمانی که *condition* برابر True باشد، تکرار می‌کند. هنگامی که شرط نادرست شود، کنترل برنامه به خط بعد از ساختار حلقه منتقل می‌شود.

منابع

- [1] Sams Teach yourself Visual Basic 6 in 21 Days, Greg Perry.
- [2] Visual Basic 6 How To Program, Deitel & Deitel.
- [3] Learn Visual Basic 6, Lou Tylee.
- [4] Using Visual Basic 6, Bob Reselman ,....
- [5] MCSD Visual Basic 6, Microsoft.



