فصل دوم

«انواع اصلی»

1 – 2 انواع دادهٔ عددی

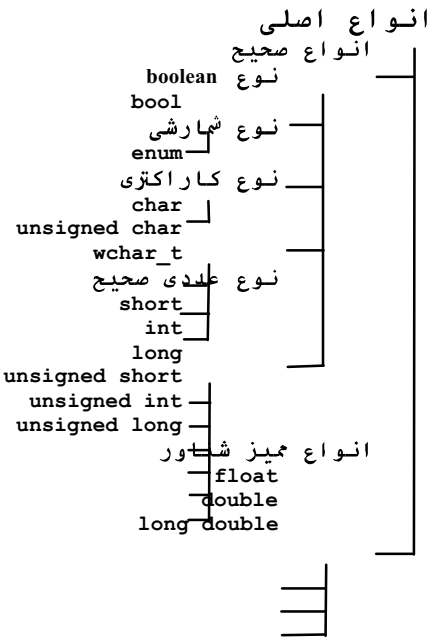
ما در زندگی روزمره از داده‌های مختلفی استفاده می‌کنیم: اعداد ، تصاویر، نوشته‌ها یا حروف الفبا، صداها، بوها و ... . با پردازش این داده‌ها می‌توانیم تصمیماتی اتخاذ کنیم، عکس‌العمل‌هایی نشان دهیم و مساله‌ای را حل کنیم. رایانه‌ها نیز قرار است همین کار را انجام دهند. یعنی داده‌هایی را بگیرند، آن‌ها را به شکلی که ما تعیین می‌کنیم پردازش کنند و در نتیجه اطلاعات مورد نیازمان را استخراج کنند. اما رایانه‌ها یک محدودیت مهم دارند: فقط اعداد را می‌توانند پردازش کنند. پس هر داده‌ای برای این که قابل پردازش باشد باید تبدیل به عدد شود. ممکن است عجیب به نظر برسد که مثلا صدا یا تصویر را چطور می‌توان به اعداد تبدیل کرد اما این کار واقعا در رایانه‌ها انجام می‌گیرد و هر نوع داده‌ای به ترکیبی از صفرها و یک‌ها که «اعداد دودویی[1]» خوانده می‌شوند، تبدیل می‌گردد. سر و کار داشتن با اعدادی که فقط از صفرها و یک‌های طولانی تشکیل شده‌اند بسیار گیج‌کننده و وقت‌گیر است. علاوه بر این مایلیم که با داده‌های واقعی در برنامه‌ها کار کنیم. بنابراین در زبان‌های برنامه‌نویسی، این تبدیل

---

1 – Binary

داده‌ها به کامپایلر واگذار شده است و برنامه‌نویس با خیال راحت می‌تواند انواع واقعی را که آن زبان در اختیار می‌گذارد به کار برد. وقتی برنامه کامپایل شد، این داده‌ها خود به خود به اعداد دودویی تبدیل می‌شوند.

در C++ دو نوع اصلی داده وجود دارد: «نوع صحیح[1]» و «نوع ممیز شناور[2]». همهٔ انواع دیگر از روی این دو ساخته می‌شوند (به شکل زیر دقت کنید).



نوع صحیح برای نگهداری اعداد صحیح (اعداد 0 و 1 و 2 و ...) استفاده می‌شود. این اعداد بیشتر برای شمارش به کار می‌روند و دامنه محدودی دارند.

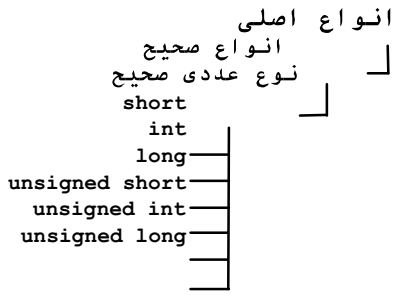
نوع ممیز شناور برای نگهداری اعداد اعشاری استفاده می‌شود. اعداد اعشاری بیشتر برای اندازه‌گیری دقیق به کار می‌روند و دامنهٔ بزرگ‌تری دارند. یک عدد اعشاری مثل 187/352 را می‌توان به شکل $18/7352\times10$ یا $1/87352\times10^{2}$ یا $1873/52\times10^{-1}$ یا $18735/2\times10^{-2}$ و یا ... نوشت. به این ترتیب با کم و زیاد کردن توان عدد 10 ممیز عدد نیز جابه‌جا می‌شود. به همین دلیل است که به اعداد اعشاری «اعداد ممیز شناور» می‌گویند.

2 – 2 متغیر عدد صحیح

C++ شش نوع متغیر عدد صحیح دارد که در شکل آمده است. تفاوت این شش نوع مربوط به میزان حافظهٔ مورد استفاده و محدودهٔ مقادیری است که هر کدام

1 – Integer
2 – Floating point

می‌توانند داشته باشند. این میزان حافظهٔ مورد استفاده و محدودهٔ مقادیر، بستگی زیادی به سخت‌افزار و همچنین سیستم عامل دارد. یعنی ممکن است روی یک رایانه، نوع int دو بایت از حافظه را اشغال کند در حالی که روی رایانه‌ای از نوع دیگر نوع int به چهار بایت حافظه نیاز داشته باشد. با استفاده از سرفایل <limits> می‌توان مشخص نمود که هر نوع عدد صحیح روی رایانه‌تان چه محدوده‌ای دارد.



× مثال 1 – 2 محدوده‌های نوع عدد صحیح

این برنامه محدوده‌های شش نوع عدد صحیح در C++ را چاپ می‌کند:

```
#include <iostream>
#include <limits>     //defines the constants SHRT_MIN, etc.
using namespace std;
int main()
{  //prints some of the constants stored in the <limits>
header:
   cout << "minimum short = " << SHRT_MIN << endl;
   cout << "maximum short = " << SHRT_MAX << endl;
   cout << "minimum unsigned short = 0" << endl;
   cout << "maximum unsigned short = " << USHRT_MAX << endl;
   cout << "minimum int = " << INT_MIN << endl;
   cout << "maximum int = " << INT_MAX << endl;
   cout << "minimum unsigned int = 0" << endl;
   cout << "maximum unsigned int = " << UINT_MAX << endl;
   cout << "minimum long = " << LONG_MIN << endl;
   cout << "maximum long = " << LONG_MAX << endl;
   cout << "minimum unsigned long = 0" << endl;
   cout << "maximum unsigned long = " << ULONG_MAX << endl;
   return 0;
}
```

```
minimum short = -32768
maximum short = 32767
```

```
minimum unsigned short = 0
maximum unsigned short = 65535
minimum int = -2147483648
maximum int = 2147483647
minimum unsigned int = 0
maximum unsigned int = 4294967295
minimum long = -2147483648
maximum long = 2147483647
minimum unsigned long = 0
maximum unsigned long = 4294967295
```

سرفایل <limits> حاوی تعریف شناسه‌های SHRT_MIN ، SHRT_MAX ، USHRT_MAX و سایر شناسه‌هایی است که در برنامهٔ بالا استفاده شده است. این شناسه‌ها گستره‌ای که نوع عدد صحیح مربوطه می‌تواند داشته باشد را نشان می‌دهند. مثلا شناسه SHRT_MIN نشان می‌دهد که متغیری از نوع short حداقل چه مقداری می‌تواند داشته باشد و شناسه SHRT_MAX بیان می‌کند که متغیری از نوع short حداکثر چه مقداری می‌تواند داشته باشد. مثال بالا روی یک رایانه با پردازنده Pentium II با سیستم عامل ویندوز 98 اجرا شده است. خروجی این مثال نشان می‌دهد که شش نوع عدد صحیح در این رایانه محدوده‌های زیر را دارند:

short:	-32,786 تا 32,767	($2^8 \Rightarrow$ 1 byte)
int:	-2,147,483,648 تا 2,147,483,647	($2^{32} \Rightarrow$ 4 bytes)
long:	-2,147,483,648 تا 2,147,483,647	($2^{32} \Rightarrow$ 4 bytes)
unsigned short:	0 تا 65,535	($2^8 \Rightarrow$ 1 byte)

با دقت در این جدول مشخص می‌شود که در رایانهٔ مذکور، نوع long مانند نوع int است و نوع unsigned long نیز مانند unsigned int است. گرچه ممکن است این انواع روی رایانه‌ای از نوع دیگر متفاوت باشد.

وقتی برنامه‌ای می‌نویسید، توجه داشته باشید که از نوع صحیح مناسب استفاده کنید تا هم برنامه دچار خطا نشود و هم حافظهٔ سیستم را هدر ندهید.

3 – 2 محاسبات اعداد صحیح

اکنون که با انواع متغیرهای عدد صحیح آشنا شدیم، می‌خواهیم از این متغیرها در محاسبات ریاضی استفاده کنیم. C++ مانند اغلب زبان‌های برنامه‌نویسی برای محاسبات

از عملگرهای جمع (+) ، تفریق (-) ، ضرب (*) ، تقسیم (/) و باقیمانده (%) استفاده می‌کند.

× مثال 2 – 2 محاسبات اعداد صحیح

برنامهٔ زیر نحوهٔ استفاده و عملکرد عملگرهای حسابی را نشان می‌دهد:

```
int main()
{  //tests operators +, -, *, /, and %:
   int m=54;
   int n=20;
   cout << "m = " << m << " and n = " << n << endl;
   cout << "m+n = " << m+n << endl;   // 54+20 = 74
   cout << "m-n = " << m-n << endl;   // 54-20 = 34
   cout << "m*n = " << m*n << endl;   // 54*20 = 1080
   cout << "m/n = " << m/n << endl;   // 54/20 = 2
   cout << "m%n = " << m%n << endl;   // 54%20 = 14
   return 0;
}
```

```
m = 54 and n = 20
m+n = 74
m-n = 34
m*n = 1080
m/n = 2
m%n = 14
```

نتیجهٔ تقسیم m/n جالب توجه است. حاصل این تقسیم برابر با 2 است نه 2.7 توجه به این مطلب بسیار مهم است. این امر نشان می‌دهد که حاصل تقسیم یک عدد صحیح بر عدد صحیح دیگر، همواره یک عدد صحیح است نه عدد اعشاری. همچنین به حاصل m%n نیز دقت کنید. عملگر % باقیماندهٔ تقسیم را به دست می‌دهد. یعنی حاصل عبارت 54%20 برابر با 14 است که این مقدار، باقیماندهٔ تقسیم 54 بر 20 است.

4 – 2 عملگرهای افزایشی و کاهشی

C++ برای دستکاری مقدار متغیرهای صحیح، دو عملگر جالب دیگر دارد: عملگر ++ مقدار یک متغیر را یک واحد افزایش می‌دهد و عملگر -- مقدار یک متغیر

را یک واحد کاهش می‌دهد. اما هر کدام از این عملگرها دو شکل متفاوت دارند: شکل «پیشوندی» و شکل «پسوندی».

در شکل پیشوندی، عملگر قبل از نام متغیر می‌آید مثل ++m یا --n . در شکل پسوندی، عملگر بعد از نام متغیر می‌آید مثل m++ یا n-- . تفاوت شکل پیشوندی با شکل پسوندی در این است که در شکل پیشوندی ابتدا متغیر، متناسب با عملگر، افزایش یا کاهش می‌یابد و پس از آن مقدار متغیر برای محاسبات دیگر استفاده می‌شود ولی در شکل پسوندی ابتدا مقدار متغیر در محاسبات به کار می‌رود و پس از آن مقدار متغیر یک واحد افزایش یا کاهش می‌یابد. برای درک بهتر این موضوع به مثال بعدی توجه کنید.

× مثال 3 – 2 استفاده از عملگرهای پیش‌افزایشی و پس‌افزایشی

```
int main()
{  //shows the difference between m++ and ++m:
   int m, n;
   m = 75;
   n = ++m;    // the pre-increment operator is applied to m
   cout << "m = " << m << ", n = " << n << endl;
   m = 75;
   n = m++;    // the post-increment operator is apllied to m
   cout << "m = " << m << ", n = " << n << endl;
   return 0;
}
```

```
m = 45, n = 45
m = 45, n = 44
```

در خط پنجم برنامه یعنی در عبارت

```
n = ++m;
```

از عملگر پیش‌افزایشی استفاده شده است. پس ابتدا مقدار m به 76 افزایش می‌یابد و سپس این مقدار به n داده می‌شود. بنابراین وقتی در خط ششم مقدار این دو متغیر چاپ می‌شود، m = 76 و n = 76 خواهد بود.

در خط هشتم برنامه یعنی در عبارت `n = m++;` از عملگر پس‌افزایشی استفاده شده است. بنا بر این ابتدا مقدار m که 75 است به n تخصیص می‌یابد و پس از آن مقدار m به 76 افزایش داده می‌شود. پس وقتی در خط نهم برنامه مقدار این دو متغیر چاپ می‌شود، `m = 76` است ولی `n = 75` خواهد بود.

عملگرهای افزایشی و کاهشی در برنامه‌های C++ فراوان به کار می‌روند. گاهی به شکل پیشوندی و گاهی به شکل پسوندی؛ این بستگی به منطق برنامه دارد که کجا از کدام نوع استفاده شود.

5 – 2 عملگرهای مقدارگذاری مرکب

قبلا از عملگر = برای مقدارگذاری در متغیرها استفاده کردیم. مثلا دستور `m=75;` مقدار 75 را درون m قرار می‌دهد و همچنین دستور `m = m+8;` مقدار m را هشت واحد افزایش می‌دهد. C++ عملگرهای دیگری دارد که مقدارگذاری در متغیرها را تسهیل می‌نمایند. مثلا با استفاده از عملگر += می‌توانیم هشت واحد به m اضافه کنیم اما با دستور کوتاه‌تر: **`m += 8;`**

دستور بالا معادل دستور `m = m + 8;` است با این تفاوت که کوتاه‌تر است. به عملگر **+=** «عملگر مرکب» می‌گویند زیرا ترکیبی از عملگرهای + و = می‌باشد. پنج عملگر مرکب در C++ عبارتند از: **+=** و **-=** و ***=** و **/=** و **%=**

نحوهٔ عمل این عملگرها به شکل زیر است:

```
m += 8;      ⟶    m = m + 8;
m -= 8;      ⟶    m = m - 8;
m *= 8;      ⟶    m = m * 8;
m /= 8;      ⟶    m = m / 8;
m %= 8;      ⟶    m = m % 8;
```

مثال زیر، کار این عملگرها را نشان می‌دهد.

× مثال 4 – 2 کاربرد عملگرهای مرکب

```
int main()
{  //tests arithmetic assignment operators:
```

```
    int n=22;
    cout << " n = " << n << endl;
    n += 9;  // adds 9 to n
    cout << "After n += 9, n = " << n << endl;
    n -= 5;  //substracts 5 from n
    cout << "After n -= 5, n = " << n << endl;
    n *= 2;  //multiplies n by 2
    cout << "After n *= 2, n = " << n << endl;
    n /= 3;  //divides n by 3
    cout << "After n /= 3, n = " << n << endl;
    n %= 7;  //reduces n to the remainder from dividing by 4
    cout << "After n %= 7, n = " << n << endl;
    return 0;
}
```

```
n = 22
After n += 9, n = 31
After n -= 5, n = 26
After n *= 2, n = 52
After n /= 3, n = 17
After n %= 7, n = 3
```

6 – 2 انواع ممیز شناور

عدد ممیز شناور به بیان ساده همان عدد اعشاری است. عددی مثل 123.45 یک عدد اعشاری است. برای این که مقدار این عدد در رایانه ذخیره شود، ابتدا باید به شکل دودویی تبدیل شود:

$$123.45 = 1111011.0111001_2$$

اکنون برای مشخص نمودن محل اعشار در عدد، تمام رقم‌ها را به سمت راست ممیز منتقل می‌کنیم. البته با هر جابجایی ممیز، عدد حاصل باید در توانی از 2 ضرب شود:

$$123.45 = 0.11110110111001 \times 2^7$$

به مقدار 11110110111001 «مانتیس عدد» و به 7 که توان روی دو است، «نمای عدد» گفته می‌شود. از آن‌جا که ممیز می‌تواند به شکل شناور جابجا شود، به اعداد اعشاری اعداد ممیز شناور می‌گویند. حال برای ذخیره‌سازی عدد مفروض کافی است

که مانتیس و نما را ذخیره کنیم. هنگامی که بخواهیم این مقدار ذخیره شده را بازیابی کنیم، سیستم‌عامل نما و مانتیس را در مسیری عکس مسیر بالا به کار می‌گیرد تا عدد 123.45 را از روی آن دوباره بسازد. در مورد عددی مثل عدد مذکور ممکن است این روش ذخیره‌سازی، طولانی و بی‌مورد به نظر برسد. اما اعداد ممیز شناور شامل اعداد خیلی کوچک مثل 0.000000001 یا اعداد خیلی بزرگ مثل 100000000.00000002 هستند که ذخیره‌سازی و انجام محاسبات ریاضی روی آن‌ها با استفاده از مانتیس و نما بسیار آسان‌تر است.

در C++ سه نوع ممیز شناور وجود دارد: نوع float و نوع double و نوع long double.

معمولا نوع float از چهار بایت برای نگهداری عدد استفاده می‌کند، نوع double از هشت بایت و نوع long double از هشت یا ده یا دوازده یا شانزده بایت. در یک float 32 بیتی (چهار بایتی) از 23 بیت برای ذخیره‌سازی مانتیس استفاده می‌شود و 8 بیت نیز برای ذخیره‌سازی نما به کار می‌رود و یک بیت نیز علامت عدد را نگهداری می‌کند.

در یک double 64 بیتی (هشت بایتی) از 52 بیت برای ذخیره‌سازی مانتیس استفاده می‌شود و 11 بیت برای نگهداری نما به کار می‌رود و یک بیت نیز علامت عدد را نشان می‌دهد.

7 – 2 تعریف متغیر ممیز شناور

تعریف متغیر ممیز شناور مانند تعریف متغیر صحیح است. با این تفاوت که از کلمهٔ کلیدی float یا double برای مشخص نمودن نوع متغیر استفاده می‌کنیم. مثلا دستور `float x;` متغیر x را از نوع ممیز شناور تعریف می‌کند. دستور `float x=12.3;` متغیر x را از نوع ممیز شناور تعریف کرده و مقدار اولیهٔ 12.3 را درون آن قرار می‌دهد. دستور `double x,y=0;` دو متغیر x و y را از نوع double تعریف می‌کند که مقدار x هنوز مشخص نیست ولی y مقدار 0.0 دارد.

× مثال 5 – 2 حساب ممیز شناور

اعداد ممیز شناور را نیز مثل اعداد صحیح می‌توانیم در محاسبات به کار ببریم. مثال زیر این موضوع را نشان می‌دهد. این مثال مانند مثال 2 – 2 است با این تفاوت که متغیرها از نوع ممیز شناور float هستند:

```
int main()
{  //tests operators +, -, *, /, and %:
   float x=54.0;
   float y=20.0;
   cout << "x = " << x << " and y = " << y << endl;
   cout << "x+y = " << x+y << endl;   // 54.0+20.0 = 74.0
   cout << "x-y = " << x-y << endl;   // 54.0-20.0 = 34.0
   cout << "x*y = " << x*y << endl;   // 54.0*20.0 = 1080.0
   cout << "x/y = " << x/y << endl;   // 54.0/20.0 = 2.7
   return 0;
}
```

```
x = 54 and y = 20
x+y = 74
x-y = 34
x*y = 1080
x/y = 2.7
```

به پاسخ‌های بالا دقت کنید: بر خلاف تقسیم اعداد صحیح، تقسیم اعداد ممیز شناور به صورت بریده‌شده نیست: 54.0 / 20.0 = 2.7

تفاوت نوع float با نوع double در این است که نوع double دو برابر float از حافظه استفاده می‌کند. پس نوع double دقتی بسیار بیشتر از float دارد. به همین دلیل محاسبات double وقت‌گیرتر از محاسبات float است. بنابراین اگر در برنامه‌هایتان به محاسبات و پاسخ‌های بسیار دقیق نیاز دارید، از نوع double استفاده کنید. ولی اگر سرعت اجرا برایتان اهمیت بیشتری دارد، نوع float را به کار بگیرید.

8 – 2 شکل علمی مقادیر ممیز شناور

اعداد ممیز شناور به دو صورت در ورودی و خروجی نشان داده می‌شوند: به شکل «ساده» و به شکل «علمی».

مقدار 12345.67 شکل ساده عدد است و مقدار 1.234567×10^{4} شکل علمی همان عدد است. مشخص است که شکل علمی برای نشان دادن اعداد خیلی کوچک و همچنین اعداد خیلی بزرگ، کارآیی بیشتری دارد:

$-0.000000000123 = -1.23\times10^{-10}$
$123000000000 = 1.23\times10^{11}$

در C++ برای نشان دادن حالت علمی اعداد ممیز شناور از حرف انگلیسی e یا E استفاده می‌کنیم:

-1.23×10^{-10} = -1.23e-10
1.23×10^{11} = 1.23e11

هنگام وارد کردن مقادیر ممیز شناور، می‌توانیم از شکل ساده یا شکل علمی استفاده کنیم. هنگام چاپ مقادیر ممیز شناور، معمولا مقادیر بین 0.1 تا 999.999 به شکل ساده چاپ می‌شوند و سایر مقادیر به شکل علمی نشان داده می‌شوند.

× مثال 6 – 2 شکل علمی اعداد ممیز شناور

برنامهٔ زیر یک عدد ممیز شناور (x) را از ورودی گرفته و معکوس آن (1/x) را چاپ می‌کند:

```
int main()
{  // prints reciprocal value of x:
   double x;
   cout << "Enter float: "; cin >> x;
   cout << "Its reciprocal is: " << 1/x << endl;
   return 0;
}
```

```
Enter float: 234.567e89
Its reciprocal is: 4.26317e-92
```

تا این‌جا انواع عددی را در C++ دیدیم. این انواع برای محاسبات استفاده می‌شوند و تقریبا در هر برنامه‌ای که می‌نویسید به کار می‌روند. اما C++ انواع دیگری نیز دارد که کاربردهای دیگری دارند. نوع بولین که برای عملیات منطقی استفاده می‌شود و نوع کاراکتری که برای به کار گرفتن کاراکترها تدارک دیده شده است و نوع شمارشی که بیشتر برای مجموعه‌هایی که برنامه‌نویس تعریف می‌کند به کار می‌رود. این انواع جدید گرچه کاربردشان با اعداد تفاوت دارد اما در حقیقت به شکل اعداد صحیح در رایانه ذخیره و شناسایی می‌شوند. به همین دلیل این نوع‌ها را نیز زیرمجموعه‌ای از انواع صحیح در C++ می‌شمارند. در ادامهٔ این فصل به بررسی این انواع می‌پردازیم.

9 – 2 نوع بولین[1] bool

نوع bool یک نوع صحیح است که متغیرهای این نوع فقط می‌توانند مقدار **true** یا **false** داشته باشند. true به معنی درست و false به معنی نادرست است. گرچه درون برنامه مجبوریم از عبارات true یا false برای مقداردهی به این نوع متغیر استفاده کنیم، اما این مقادیر در اصل به صورت 1 و 0 درون رایانه ذخیره می‌شوند: 1 برای true و 0 برای false. مثال زیر این مطلب را نشان می‌دهد.

× مثال 7 – 2 استفاده از متغیرهای نوع bool

```
int main()
{  //prints the vlaue of a boolean variable:
   bool flag=false;
   cout << "flag = " << flag << endl;
   flag = true;
   cout << "flag = " << flag << endl;
   return 0;
}
```

```
flag = 0
flag = 1
```

1 – Boolean

در خط سوم از برنامهٔ بالا متغیری به نام falg از نوع bool تعریف شده و با مقدار false مقداردهی اولیه شده است. در خط بعدی مقدار این متغیر در خروجی چاپ شده و در خط پنجم مقدار آن به true تغییر یافته است و دوباره مقدار متغیر چاپ شده است. گرچه به متغیر flag مقدار false و true داده‌ایم اما در خروجی به جای آن‌ها مقادیر 0 و 1 چاپ شده است.

10 -2 نوع کاراکتری char

یک ***کاراکتر*** یک حرف، رقم یا نشانه است که یک شمارهٔ منحصر به فرد دارد. به عبارت عامیانه، هر کلیدی که روی صفحه‌کلید خود می‌بینید یک کاراکتر را نشان می‌دهد (البته به غیر از کلیدهای مالتی‌مدیا یا کلیدهای اینترنتی که اخیرا در صفحه‌کلیدها مرسوم شده‌اند). مثلا هر یک از حروف 'A' تا 'Z' و 'a' تا 'z' و هر یک از اعداد '0' تا '9' و یا نشانه‌های '~' تا '+' روی صفحه‌کلید را یک کاراکتر می‌نامند. رایانه‌ها برای شناسایی کاراکترهای استاندارد از جدول اسکی استفاده می‌کنند. با دقت در این جدول خواهید دید که هر کاراکتر یک شمارهٔ منحصر به فرد دارد. مثلا کاراکتر 'A' کد 65 دارد. کاراکترها در رایانه به شکل عددی‌شان ذخیره می‌شوند اما به شکل کاراکتری‌شان نشان داده می‌شوند. مثلا کاراکتر 'A' به شکل عدد 65 ذخیره می‌شود اما اگر سعی کنیم متغیری که کاراکتر 'A' در آن ذخیره شده را چاپ کنیم، شکل A را در خروجی می بینیم نه عدد 65 را.

برای تعریف متغیری از نوع کاراکتر از کلمه کلیدی **char** استفاده می‌کنیم. یک کاراکتر باید درون دو علامت آپستروف (') محصور شده باشد. پس 'A' یک کاراکتر است؛ همچنین '8' یک کاراکتر است اما 8 یک کاراکتر نیست بلکه یک عدد صحیح است.

مثال بعدی نحوهٔ به کارگیری متغیرهای کاراکتری را نشان می‌دهد.

× مثال 8 – 2 استفاده از متغیرهای نوع char

```
int main()
{  //prints the character and its internally stored integer value:
   char c ='A';
```

```
    cout << "c = " << c << ", int(c) = " << int(c) << endl;
    c ='t';
    cout << "c = " << c << ", int(c) = " << int(c) << endl;
    c ='\t';   // the tab character
    cout << "c = " << c << ", int(c) = " << int(c) << endl;
    c ='!';
    cout << "c = " << c << ", int(c) = " << int(c) << endl;
    return 0;
}
```

```
c = A, int(c) = 65
c = t, int(c) = 116
c =      , int(c) = 9
c = !, int(c) = 33
```

در خط سوم از برنامهٔ بالا متغیری به نام c از نوع char تعریف شده و با مقدار 'A' مقدارگذاری اولیه شده است. سپس در خط بعدی ابتدا مقدار c چاپ شده که در خروجی همان A دیده می‌شود نه مقدار عددی آن. در ادامهٔ خط چهارم، با استفاده از دستور int(c) مقدار عددی c یعنی 65 در خروجی چاپ خواهد شد. در خطوط بعدی کاراکترهای دیگری به c اختصاص یافته و به همین ترتیب مقدار c و معادل عددی آن چاپ شده است.

به خط هفتم برنامه نگاه کنید: `c = '\t';`

کاراکتر '\t' یک کاراکتر خاص است. اگر سعی کنیم کاراکتر '\t' را روی صفحه‌نمایش نشان دهیم، هفت جای خالی روی صفحه دیده می‌شود(به خروجی دقت کنید). غیر از این کاراکتر، کاراکترهای خاص دیگری نیز هستند که کارهایی مشابه این انجام می‌دهند. مثل کاراکتر '\n' که در فصل قبل دیدیم و مکان‌نما را به سطر بعدی منتقل می‌کند. این کاراکترها برای شکل‌دهی صفحه‌نمایش و کنترل آن استفاده می‌شوند. کاراکترهای خاص در جدول اسکی بین شماره‌های 0 تا 32 قرار گرفته‌اند. سعی کنید مثل برنامهٔ بالا یک برنامهٔ آزمایشی بنویسید و با استفاده از آن مقادیر کاراکترهای خاص را چاپ کنید تا ببینید چه اتفاقی می‌افتد. لابه‌لای برنامه‌های بعدی از این کاراکترهای خاص استفاده خواهیم کرد تا با آن‌ها بهتر آشنا شوید.

11 – 2 نوع شمارشی enum

علاوه بر انواعی که تا کنون بررسی کردیم، می توان در C++ انواع جدیدی که کاربر نیاز دارد نیز ایجاد نمود. برای این کار راه‌های مختلفی وجود دارد که بهترین و قوی‌ترین راه، استفاده از کلاس‌ها است(فصل ده)، اما راه ساده‌تری نیز وجود دارد و آن استفاده از نوع شمارشی enum است.

یک نوع ***شمارشی*** یک نوع صحیح است که توسط کاربر مشخص می‌شود. نحو تعریف یک نوع شمارشی به شکل زیر است:

```
enum typename{enumerator-list}
```

که **enum** کلمه‌ای کلیدی است، typename نام نوع جدید است که کاربر مشخص می‌کند و *enumerator-list* مجموعه مقادیری است که این نوع جدید می‌تواند داشته باشد. به عنوان مثال به تعریف زیر دقت کنید:

```
enum Day{SAT,SUN,MON,TUE,WED,THU,FRI}
```

حالا Day یک نوع جدید است و متغیرهایی که از این نوع تعریف می‌شوند می‌توانند یکی از مقادیر SAT و SUN و MON و TUE و WED و THU و FRI را داشته باشند:

```
Day day1,day2;
day1 = MON;
day2 = THU;
```

وقتی نوع جدید Day و محدودهٔ مقادیرش را تعیین کردیم، می‌توانیم متغیرهایی از این نوع جدید بسازیم. در کد بالا متغیرهای day1 و day2 از نوع Day تعریف شده‌اند. آنگاه day1 با مقدار MON و day2 با مقدار THU مقداردهی شده است.

مقادیر SAT و SUN و ... هر چند که به همین شکل به کار می‌روند اما در رایانه به شکل اعداد صحیح 0 و 1 و 2 و ... ذخیره می‌شوند. به همین دلیل است که به هر یک از مقادیر SAT و SUN و ... یک ***شمارشگر***[1] می‌گویند. وقتی فهرست شمارشگرهای یک نوع تعریف شد، به طور خودکار مقادیر 0 و 1 و 2 و ... به ترتیب

1 – Enumerator

به آن‌ها اختصاص می‌یابد. هرچند که می‌توان این ترتیب را شکست و مقادیر صحیح دلخواهی را به شمارشگرها نسبت داد:

```
enum Day{SAT=1,SUN=2,MON=4,TUE=8,WED=16,THU=32,FRI=64}
```

اگر فقط بعضی از شمارشگرها مقداردهی شوند، آنگاه سایر شمارشگرها که مقداردهی نشده‌اند مقادیر متوالی بعدی را خواهند گرفت:

```
enum Day{SAT=1,SUN,MON,TUE,WED,THU,FRI}
```

دستور بالا مقادیر 1 تا 7 را به ترتیب به روزهای هفته تخصیص خواهد داد.

همچنین دو یا چند شمارشگر در یک فهرست می‌توانند مقادیر یکسانی داشته باشند:

```
enum Answer{NO=0,FALSE=0,YES=1,TRUE=1,OK=1}
```

در کد بالا دو شمارشگر NO و FALSE دارای مقدار یکسان 0 و شمارشگرهای YES و TRUE و OK نیز دارای مقدار یکسان 1 هستند. پس کد زیر معتبر است و به درستی کار می‌کند:

```
Answer answer;
cin >> answer;
if (answer==TRUE) cout << "you said OK.";
```

به اولین خط کد فوق نگاه کنید. این خط ممکن است کمی عجیب به نظر برسد:

```
Answer answer;
```

این خط متغیری به نام answer از نوع Answer تعریف می‌کند. اولین قانون در برنامه‌های C++ را به خاطر بیاورید: «C++ بین حروف کوچک و بزرگ تفاوت قایل است». پس Answer با answer متفاوت است. Answer را در خط‌های قبلی یک نوع شمارشی تعریف کردیم و answer را متغیری که از نوع Answer است. یعنی answer متغیری است که می‌تواند یکی از مقادیر YES یا TRUE یا OK یا FALSE یا NO را داشته باشد. نحوهٔ انتخاب نام‌ها آزاد است اما بیشتر برنامه‌نویسان از توافق زیر در برنامه‌هایشان استفاده می‌کنند:

1 – برای نام ثابت‌ها از حروف بزرگ استفاده کنید

2 – اولین حرف از نام نوع شمارشی را با حرف بزرگ بنویسید.

3 – در هر جای دیگر از حروف کوچک استفاده کنید.

رعایت این توافق به خوانایی برنامه‌تان کمک می‌کند. همچنین سبب می‌شود که انواع شمارشی که کاربر تعریف می‌کند از انواع استاندارد مثل int و float و char راحت‌تر تمیز داده شود.

شمارشگرها قواعد خاصی دارند. نام شمارشگر باید معتبر باشد. یعنی کلمهٔ کلیدی نباشد، با عدد شروع نشود و نشانه‌های ریاضی نیز نداشته باشد. پس تعریف زیر غیرمعتبر است:

```
enum Score{A+,A,A-,B+,B,B-,C+,C,C-}
```

زیرا A+ و A- و B+ و B- و C+ و C- نام‌های غیرمعتبری هستند چون در نام آن‌ها از نشانه‌های ریاضی استفاده شده.

علاوه بر این شمارشگرهای هم‌نام نباید در محدوده‌های مشترک استفاده شوند. برای مثال تعریف‌های زیر را در نظر بگیرید:

```
enum Score{A,B,C,D}
enum Group{AB,B,BC}
```

دو تعریف بالا غیرمجاز است زیرا شمارشگر B در هر دو تعریف Score و Group آمده است.

آخر این که نام شمارشگرها نباید به عنوان نام متغیرهای دیگر در جاهای دیگر برنامه استفاده شود. مثلا:

```
enum Score{A,B,C,D}
float B;
char c;
```

در تعریف‌های بالا B و C را نباید به عنوان نام متغیرهای دیگر به کار برد زیرا این نام‌ها در نوع شمارشی Score به کار رفته است. پس اگر این سه تعریف در یک محدوده

باشند، دو تعریف آخری غیرمجاز خواهد بود. انواع شمارشی برای تولید کد «خود مستند» به کار می‌روند، یعنی کدی که به راحتی درک شود و نیاز به توضیحات اضافی نداشته باشد. مثلا تعاریف زیر خودمستند هستند زیرا به راحتی نام و نوع کاربرد و محدودهٔ مقادیرشان درک می‌شود:

```
enum Color{RED,GREEN,BLUE,BLACK,ORANGE}
enum Time{SECOND,MINUTE,HOUR}
enum Date{DAY,MONTH,YEAR}
enum Language{C,DELPHI,JAVA,PERL}
enum Gender{MALE,FEMALE}
```

12 – 2 تبدیل نوع، گسترش نوع

در قسمت‌های قبلی با انواع عددی آشنا شدیم و نحوهٔ اعمال ریاضی آن‌ها را مشاهده نمودیم اما در محاسبات ریاضی که انجام دادیم همهٔ متغیرها از یک نوع بودند. اگر بخواهیم در یک محاسبه دو یا چند متغیر از انواع مختلف به کار ببریم چه اتفاقی می‌افتد؟

قانون کلی این است که در محاسباتی که چند نوع متغیر وجود دارد، جواب همیشه به شکل متغیری است که دقت بالاتری دارد. یعنی اگر یک عدد صحیح را با یک عدد ممیز شناور جمع ببندیم، پاسخ به شکل ممیز شناور است. به این منظور ابتدا متغیرها و مقادیری که از نوع با دقت کمتر هستند به نوع با دقت بیشتر تبدیل می‌شوند و سپس محاسبه روی آن‌ها انجام می‌شود. پس اگر یک عدد صحیح را با یک عدد ممیز شناور جمع ببندیم، ابتدا عدد صحیح تبدیل به یک عدد ممیز شناور می‌شود، سپس این عدد با عدد ممیز شناور دیگر جمع بسته می‌شود و واضح است که پاسخ نیز به شکل ممیز شناور خواهد بود. این کار به شکل خودکار انجام می‌گیرد و C++ در چنین محاسباتی به شکل خودکار متغیرهای با دقت کمتر را به متغیرهایی با دقت بیشتر تبدیل می‌کند تا همه متغیرها از یک نوع شوند و آنگاه محاسبه را انجام می‌دهد و پاسخ را نیز به شکل نوع با دقت بیشتر به دست می‌دهد. به این عمل ***گسترش نوع*** می‌گویند.

اما اگر عکس این عمل مورد نظر باشد، یعنی اگر بخواهیم یک متغیر صحیح را با یک متغیر ممیز شناور جمع ببندیم و بخواهیم که حاصل از نوع صحیح باشد نه ممیز شناور، چه باید بکنیم؟ در چنین حالتی از عملگر تبدیل نوع استفاده می‌کنیم. این تبدیل خودکار نیست بلکه کاملا باید دستی انجام شود و برنامه‌نویس، خود باید مراقب این عمل باشد. برای این که مقدار یک متغیر از نوع ممیز شناور را به نوع صحیح تبدیل کنیم از عبارت int () استفاده می‌کنیم.

مثال‌های زیر تبدیل نوع و گسترش نوع را نشان می‌دهند.

× مثال 9 – 2 تبدیل نوع

این برنامه، یک نوع double را به نوع int تبدیل می‌کند:

```
int main()
{  // casts a double value as an int:
   double v=1234.987;
   int n;
   n = int(v);
   cout << "v = " << v << ", n = " << n << endl;
   return 0;
}
```

```
v = 1234.987, n = 1234
```

در این برنامه متغیر v از نوع double و با مقدار 1234.987 تعریف شده است. همچنین متغیر n از نوع int تعریف گشته است. در خط پنجم از کد بالا از تبدیل نوع استفاده شده:

```
n = int(v);
```

با استفاده از این دستور، مقدار v ابتدا به نوع int تبدیل می‌شود و سپس این مقدار درون n قرار می‌گیرد. خروجی برنامه نشان می‌دهد که وقتی از عملگر int () استفاده کنیم، عدد ممیز شناور «بریده» می‌شود، گرد نمی‌شود. یعنی قسمت اعشاری عدد به طور کامل حذف می‌شود و فقط قسمت صحیح آن باقی می‌ماند. بنابراین وقتی

عدد 1234.987 به نوع int تبدیل شود، حاصل برابر با 1234 خواهد بود و قسمت اعشاری آن (هر قدر هم بزرگ باشد) نادیده گرفته می‌شود.

در تبدیل نوع همواره نوع و مقدار متغیرهای تبدیل شده بدون تغییر می‌ماند. در برنامهٔ بالا مقدار v تا پایان برنامه به همان مقدار 1234.987 باقی مانده و نوع v نیز تغییر نکرده و همچنان از نوع double مانده است. تنها اتفاقی که افتاده این است که مقدار v در یک محل موقتی تبدیل به int شده تا این مقدار درون n قرار گیرد.

× مثال 10 – 2 گسترش نوع

برنامهٔ زیر یک عدد صحیح را با یک عدد ممیز شناور جمع می‌کند:

```
int main()
{  // adds an int value with a double value:
   int n = 22;
   double p = 3.1415;
   p += n;
   cout << "p = " << p << ", n = " << n << endl;
   return 0;
}
```

```
p = 24.1415, n = 22
```

در برنامهٔ بالا ابتدا مقدار n از مقدار صحیح 22 به مقدار اعشاری 22.0 گسترش می‌یابد و سپس این مقدار با مقدار قبلی p جمع می‌شود. حاصل یک عدد ممیز شناور است.

× مثال 11 – 2 گسترش نوع

این برنامه، یک char را به int ، float و double گسترش می‌دهد:

```
int main()
{  //prints promoted values of 65 from char to double:
   char c='A';     cout << "   char c = " << c << endl;
   short k=c;      cout << "  short k = " << k << endl;
   int m=k;        cout << "    int m = " << m << endl;
   long n=m;       cout << "   long n = " << n << endl;
   float x=n;      cout << "  float x = " << x << endl;
```

```
   double y=x;     cout << " double y = " << y << endl;
   return 0;
}
```

```
char c = A
short k = 65
 int m = 65
 long n = 65
float x = 65
double y = 65
```

در مثال بالا ابتدا متغیر c از نوع char تعریف شده و کاراکتر 'A' در آن قرار گرفته است. سپس مقدار c درون متغیر k که از نوع short است قرار گرفته. چون نوع k بالاتر از نوع c است، پس مقدار c به نوع short گسترش می‌یابد و مقدار 65 که معادل عددی کاراکتر 'A' است درون k قرار می‌گیرد.

در خط بعدی، مقدار k درون متغیر m قرار می‌گیرد. m از نوع int است که نوع بالاتری از short می‌باشد. پس مقدار k به int گسترش می‌یابد و این مقدار گسترش‌یافته درون m نهاده می‌شود.

به همین ترتیب در خطوط بعدی مقدار m به نوع long گسترش یافته و درون n قرار می‌گیرد. مقدار n نیز به نوع float گسترش یافته و درون x قرار می‌گیرد. مقدار x نیز به نوع double گسترش می‌یابد و درون y قرار می‌گیرد. دقت کنید که مقدار x و y در خروجی به جای آن که 65.0 باشد به شکل 65 نشان داده شده. این مقدار، یک عدد صحیح نیست اما چون قسمت اعشاری آن صفر است، اعشار حذف شده و 65 تنها نشان داده شده است.

13 – 2 برخی از خطاهای برنامه‌نویسی

اکنون که انواع متغیر در C++ را شناختیم، می‌توانیم از این انواع در برنامه‌های مفیدتر و جدی‌تر استفاده کنیم. اما باید دقت نمود که اگر از متغیرها به شکل نادرست یا کنترل‌نشده استفاده کنیم، برنامه دچار خطا می‌شود. البته عوامل دیگری نیز هست که باعث می‌شود اجرای برنامه مختل گردد، مثل استفاده از متغیری که تعریف نشده یا جا انداختن سمیکولن انتهای دستورها. این قبیل خطاها که اغلب خطاهای نحوی هستند و

توسط کامپایلر کشف می‌شوند «خطای زمان کامپایل» نامیده می‌شوند و به راحتی می‌توان آن‌ها را رفع نمود. اما خطاهای دیگری نیز وجود دارند که کشف آن‌ها به راحتی ممکن نیست و کامپایلر نیز چیزی راجع به آن نمی‌داند. به این خطاها «خطای زمان اجرا» می‌گویند. برخی از خطاهای زمان اجرا سبب می‌شوند که برنامه به طور کامل متوقف شود و از کار بیفتد. در چنین حالتی متوجه می‌شویم که خطایی رخ داده است و در صدد کشف و رفع آن برمی‌آییم. برخی دیگر از خطاهای زمان اجرا، برنامه را از کار نمی‌اندازند بلکه برنامه همچنان کار می‌کند اما پاسخ‌های عجیب و نادرست می‌دهد. این بدترین نوع خطاست زیرا در حالات خاصی رخ می‌دهد و گاهی سبب گیج شدن برنامه‌نویس می‌گردد. در بخش‌های بعدی برخی از خطاهای رایج زمان اجرا را نشان می‌دهیم تا در برنامه‌هایتان از آن‌ها پرهیز کنید؛ دست کم اگر با پاسخ‌های غیرمنتظره و غلط مواجه شدید، محل رخ دادن خطا را راحت‌تر پیدا کنید.

14 – 2 سرریزی[1] عددی

نوع صحیح long یا نوع ممیز شناور double محدودهٔ وسیعی از اعداد را می‌توانند نگهداری کنند. به بیان ساده‌تر، متغیری که از نوع long یا double باشد، گنجایش زیادی دارد. اما حافظهٔ رایانه‌ها متناهی است. یعنی هر قدر هم که یک متغیر گنجایش داشته باشد، بالاخره مقداری هست که از گنجایش آن متغیر بیشتر باشد. اگر سعی کنیم در یک متغیر مقداری قرار دهیم که از گنجایش آن متغیر فراتر باشد، متغیر «سرریز» می‌شود. مثل یک لیوان آب که اگر بیش از گنجایش آن در لیوان آب بریزیم، سرریز می‌شود. در چنین حالتی می‌گوییم که خطای سرریزی رخ داده است.

× مثال 12 – 2 سرریزی عدد صحیح

این برنامه به طور مکرر n را در 1000 ضرب می‌کند تا سرانجام سرریز شود:

```
int main()
{  //prints n until it overflows:
   int n =1000;
   cout << "n = " << n << endl;
```

1 – Overflow

```
   n *= 1000; // multiplies n by 1000
   cout << "n = " << n << endl;
   n *= 1000; // multiplies n by 1000
   cout << " n = " << n << endl;
   n *= 1000; // multiplies n by 1000
   cout << " n = " << n << endl;
   return 0;
}
```

```
n = 1000
n = 1000000
n = 1000000000
n = -727379968
```

این مثال نشان می‌دهد رایانه‌ای که این برنامه را اجرا کرده است، نمی‌تواند بیشتر از1,000,000,000 را با 1000 به طور صحیح ضرب کند.

× مثال 13 – 2 سرریزی عدد ممیز شناور

این برنامه شبیه چیزی است که در مثال قبل ذکر شد؛ به طور مکرر x را به توان می‌رساند تا این که سرریز شود.

```
int main()
{  //prints x until it overflows:
   float x=1000.0;
   cout << "x = " << x << endl;
   x *= x;   //multiplies n by itself; i.e., it squares x
   cout << "x = " << x << endl;
   x *= x;   //multiplies n by itself; i.e., it squares x
   cout << "x = " << x << endl;
   x *= x;   //multiplies n by itself; i.e., it squares x
   cout << "x = " << x << endl;
   x *= x;   //multiplies n by itself; i.e., it squares x
   cout << "x = " << x << endl;
   return 0;
}
```

```
x = 1000
x = 1e+06
x = 1e+12
```

```
x = 1e+24
x = inf
```

مثال بالا نشان می‌دهد که این رایانه نمی‌تواند x را با شروع از 1000 بیش از سه بار مجذور کند. آخرین خروجی یعنی inf نمادی است که به معنای بی‌نهایت می‌باشد (این نماد مخفف infinity به معنای بی‌انتها است).

به تفاوت سرریزی عدد صحیح و سرریزی ممیز شناور توجه کنید. وقتی یک عدد صحیح سرریز شود، عدد سرریز شده به یک مقدار منفی «گردانیده» می‌شود اما وقتی یک عدد ممیز شناور سرریز شود، نماد inf به معنای بی‌نهایت را به دست می‌دهد، نشانه‌ای مختصر و مفید.

15 – 2 خطای گرد کردن[1]

خطای گرد کردن نوع دیگری از خطاست که اغلب وقتی رایانه‌ها روی اعداد حقیقی محاسبه می‌کنند، رخ می‌دهد. برای مثال عدد 1/3 ممکن است به صورت 0.333333 ذخیره شود که دقیقا معادل 1/3 نیست. به این اختلاف، ***خطای گرد کردن*** می‌گویند. این خطا از آن‌جا ناشی می‌شود که اعدادی مثل 1/3 مقدار دقیق ندارند و رایانه نمی‌تواند این مقدار را پیدا کند، پس نزدیک‌ترین عدد قابل محاسبه را به جای چنین اعدادی منظور می‌کند. در بعضی حالات، این خطاها می‌تواند مشکلات حادی را ایجاد کند.

× مثال 14 – 2 خطای گرد کردن

این برنامه محاسبات ساده‌ای را انجام می‌دهد تا خطای گرد کردن را نشان دهد:

```
int main()
{  //illustrates round-off error:
   double x = 1000/3.0;
   cout << "x = " << x << endl;          // x = 1000/3
   double y = x-333.0;
   cout << "y = " << y << endl;          // y = 1/3
   double z = 3*y-1.0;
```

1 – Round-off

```
    cout << "z = " << z << endl;           // z = 3(1/3) - 1
    if (z == 0) cout << "z == 0.\n";
    else cout << "z does not equal 0.\n";  //z != 0
    return 0;
}
```

```
x = 333.333
y = 0.333333
z = -5.68434e-14
z does not equal 0.
```

منطق برنامه به این شکل است که ابتدا مقدار x برابر با 1000/3 یعنی $333\frac{1}{3}$ است. سپس قسمت صحیح x یعنی 333 از آن کسر می‌شود و حاصل که برابر با 1/3 است در y قرار می‌گیرد. حالا y در 3 ضرب می‌شود تا حاصل برابر با 1 شود. این مقدار از 1 کم می‌شود و حاصل در z قرار می‌گیرد. انتظار این است که z صفر باشد اما پاسخ برنامه به ما می‌گوید که z صفر نیست!

اشکال برنامهٔ بالا در کجاست؟ منطق برنامه که درست است، پس جایی در محاسبات باید غلط باشد. مشکل در مقدار y است. رایانه مقدار 1/3 را برابر با 0.333333 محاسبه نموده است، حال آن که می‌دانیم این مقدار دقیقا برابر با 1/3 نیست. این خطا از آن جا ناشی می‌شود که رایانه نمی‌تواند مقدار دقیق 1/3 را پیدا کند چون این مقدار به تعداد نامتناهی اعشار 3 دارد، پس رایانه این مقدار را گرد می‌کند و مقدار «نسبتا درست» 0.333333 را می‌دهد. این مقدار در محاسبات بعدی استفاده می‌شود اما چون دقیق نیست، پاسخ‌های بعدی نیز به تناسب بر میزان خطا می‌افزاید. نتیجه این است که مقدار z صفر نمی‌شود، هرچند که بسیار نزدیک به صفر باشد.

مثال بالا نکتهٔ مهمی را در استفاده از متغیرهای ممیز شناور نشان می‌دهد: «هیچ‌گاه از متغیر ممیز شناور برای مقایسه برابری استفاده نکنید» زیرا در متغیرهای ممیز شناور خطای گرد کردن سبب می‌شود که پاسخ با آن چه مورد نظر شماست متفاوت باشد. در حالت بالا گر چه مقدار z بسیار نزدیک صفر است، اما رایانه همین مقدار کوچک را صفر نمی‌داند. پس مقایسهٔ برابری شکست می‌خورد.

× مثال 15 – 2 خطای گرد کردن پنهان

برنامهٔ زیر با استفاده از رابطهٔ معادلات درجه دوم، ریشه‌های این معادله‌ها را پیدا می‌کند:

```
#include <cmath>   //defines the sqrt() function
#include <iostream>
using namespace std;
int main()
{  //implements the quadratic formula
   float a, b, c;
   cout << "Enter the coefficients of a quadratic equation:"
        << endl;
   cout << "\ta: ";
   cin >> a;
   cout << "\tb: ";
   cin >> b;
   cout << "\tc: ";
   cin >> c;
   cout << "The equation is: " << a << "*x*x + " << b
        << "*x + " << c << " = 0" << endl;
   float d = b*b - 4*a*c;   // discriminant
   float sqrtd = sqrt(d);
   float x1 = (-b + sqrtd /(2*a);
   float x2 = (-b - sqrtd /(2*a);
   cout << "The solutions are:" << endl;
   cout << "\tx1 = " << x1 << endl;
   cout << "\tx2 = " << x2 << endl;
   cout << "check:" << endl;
   cout << "\ta+x1*x1 + b*x1 + c = " << a*x1*x1 + b*x1 + c
        << endl;
   cout << "\ta+x2*x2 + b*x2 + c = " << a*x2*x2 + b*x2 + c
        << endl;
   return 0;
}
```

این برنامه ضرایب a و b و c را برای معادلهٔ $ax^2+bx+c=0$ می‌گیرد و سپس سعی می‌کند ریشه‌های x1 و x2 را پیدا کند. برای این کار سه متغیر a و b و c از نوع float تعریف شده‌اند تا بتوانند مقادیر اعشاری را هم از ورودی بگیرند. خط هشتم تا سیزدهم از برنامهٔ بالا مقادیر a و b و c را دریافت می‌کنند (دقت کنید که از کاراکتر خاص '\t' در پیغام‌های خروجی استفاده شده تا قبل از هر ورودی، هفت جای خالی قرار بگیرد. ولی خود حرف t چاپ نمی‌شود). پس از دریافت ضرایب، یک بار دیگر شکل کلی معادله‌ای که مورد نظر کاربر بوده است چاپ می‌شود.

در خط یازدهم، رابطهٔ دلتا یعنی b^2-4ac تشکیل شده است. این مقدار درون متغیر دیگری که d نام دارد و از نوع float است، قرار گرفته. در خط بعدی مقدار جذر دلتا با استفاده از تابع sqrt() محاسبه شده است. تابع sqrt() جذر عددی که درون پرانتزهایش قرار می‌گیرد را به دست می‌دهد. این تابع در سرفایل <cmath> تعریف شده. پس راهنمای پیش‌پردازندهٔ #include<cmath> به ابتدای برنامه افزوده شده است. مقدار جذر دلتا درون متغیر دیگری به نام sqrtd نگهداری شده تا با استفاده از آن در خطوط بعدی مقادیر x1 و x2 به دست آید.

در چهار خط آخر برنامه، مقادیر x1 و x2 که بدست آمده است دوباره در معادله جای‌گذاری می‌شود تا بررسی شود که آیا جواب معادله صفر می‌شود یا خیر. به این وسیله صحت پاسخ‌های x1 و x2 تحقیق می‌شود.

خروجی زیر نشان می‌دهد که برنامه، معادلهٔ $2x^2+1x-3=0$ را حل کرده است:

```
Enter the coeficients of a quadratic equation:
   a: 2
   b: 1
   c: -3
The equation is: 2*x*x + 1*x + -3 = 0
The solutions are:
   x1 = 1
   x2 = -1.5
check:
   a*x1*x1 + b*x1 + c = 0
   a*x2*x2 + b*x2 + c = 0
```

می‌بینید که برنامه پاسخ‌های x1=1 و x2=-1.5 را پیدا کرده است و آزمون پاسخ نیز جواب صفر داده است. خروجی دیگری از برنامه نشان می‌دهد که برنامه تلاش کرده معادله $2x^2 + 8.001x + 8.002 = 0$ را حل کند ولی شکست می‌خورد:

```
Enter the coeficients of a quadratic equation:
   a: 2
   b: 8.001
   c: 8.002
The equation is: 2*x*x + 8.001*x + 8.002 = 0
The solutions are:
   x1 = -1.9995
   x2 = -2.00098
check:
   a*x1*x1 + b*x1 + c = 5.35749e-11
   a*x2*x2 + b*x2 + c = -2.96609e-1
```

مقدار x1 که در اجرای بالا به دست آمده، در آزمون شرکت کرده و پاسخی بسیار نزدیک به صفر داده است. اما مقدار x2 در آزمون شکست خورده زیرا جواب معادله به ازای آن صفر نیست. چه چیزی باعث شده تا معادله پاسخ غلط بدهد؟ جواب باز هم در خطای گرد کردن است. x2 یک پاسخ گردشده است نه یک پاسخ دقیق. این پاسخ گردشده دوباره در یک محاسبه دیگر شرکت می‌کند. پاسخ این محاسبه هم گردشده است. پس انحراف از جواب افزایش می‌یابد و نتیجه‌ای دور از انتظار به بار می‌آورد.

× مثال 16 – 2 انواع دیگری از خطاهای زمان اجرا

دوباره به برنامهٔ محاسبهٔ ریشه‌ها برگردیم. به اجرای زیر نگاه کنید:

```
Enter the coeficients of a quadratic equation:
   a: 1
   b: 2
   c: 3
The equation is: 1*x*x + 2*x + 3 = 0
The solutions are:
   x1 = nan
   x2 = nan
check:
   a*x1*x1 + b*x1 + c = nan
   a*x2*x2 + b*x2 + c = nan
```

در این اجرا سعی شده تا معادله $1x^2+2x+3=0$ حل شود. این معادله جواب حقیقی ندارد زیرا دلتا منفی است. وقتی برنامه اجرا شود، تابع sqrt() تلاش می‌کند جذر یک عدد منفی را بگیرد ولی موفق نمی‌شود. در این حالت پاسخ nan داده می‌شود(nan مخفف عبارت not a number است یعنی پاسخ عددی نیست). سپس هر محاسبهٔ دیگری که از این مقدار استفاده کند، همین پاسخ nan را خواهد داشت. به همین دلیل در همهٔ خروجی‌ها پاسخ nan آمده است.

سرانجام به اجرای زیر دقت نمایید:

```
Enter the coeficients of a quadratic equation:
   a: 0
   b: 2
   c: 5
The equation is: 0*x*x + 2*x + 5 = 0
The solutions are:
   x1 = nan
   x2 = -inf
check:
   a*x1*x1 + b*x1 + c = nan
```

در این اجرا کوشش شده تا معادلهٔ $0x^2+2x+5=0$ حل شود. این معادله دارای جواب x=2.5 است اما برنامه نمی‌تواند این جواب را بیابد و با پاسخ‌های عجیبی روبرو می‌شویم. علت این است که a صفر است و در حین اجرای برنامه، سعی می‌شود عددی بر صفر تقسیم شود. یعنی برنامه معادلهٔ زیر را حل می‌کند:

$$x_1=\frac{-b+\sqrt{b^2-4ac}}{2a}=\frac{-(2)+\sqrt{(2)^2-4(0)(5)}}{2(0)}=\frac{-2+2}{0}=\frac{0}{0}$$

در چنین حالتی دوباره پاسخ nan بدست می‌آید. همچنین برای x2 داریم:

$$x_2=\frac{-b-\sqrt{b^2-4ac}}{2a}=\frac{-(2)-\sqrt{(2)^2-4(0)(5)}}{2(0)}=\frac{-2-2}{0}=\frac{-4}{0}$$

پاسخ این تقسیم، عبارت inf- یعنی بی‌نهایت منفی است.

سه نشانهٔ nan و inf و -inf ثابت‌های عددی هستند. یعنی می‌توانید این مقادیر را در محاسبات به کار ببرید اما نتیجه معمولا بی‌فایده است. مثلا می‌توانید عددی را با inf جمع کنید یا از آن تفریق نمایید اما نتیجه باز هم inf خواهد بود.

16 – 2 حوزهٔ متغیرها

متغیرها بخش مهمی از هر برنامه هستند. استفاده از متغیرهایی با نوع نامناسب سبب هدر رفتن حافظه و کاهش سرعت و افزایش خطاهای زمان اجرا می‌شود. انتخاب نام‌های نامفهوم یا ناقص سبب کاهش خوانایی برنامه و افزایش خطاهای برنامه‌نویسی می‌شود. استفاده از متغیرها در حوزهٔ نامناسب هم سبب بروز خطاهایی می‌شود. «حوزه[1] متغیر» محدوده‌ای است که یک متغیر خاص اجازه دارد در آن محدوده به کار رود یا فراخوانی شود.

اصطلاح «بلوک[2]» در C++ واژه مناسبی است که می‌توان به وسیلهٔ آن حوزهٔ متغیر را مشخص نمود. یک بلوک برنامه، قسمتی از برنامه است که درون یک جفت علامت کروشه { } محدود شده است. در برنامه‌هایی که تاکنون دیدیم از بلوک استفاده کرده‌ایم. همیشه بعد از عبارت int main() یک کروشه باز { گذاشته‌ایم و در پایان برنامه یک کروشه بسته } قرار دادیم. پس تمام برنامه‌هایی که تا کنون ذکر شد، یک بلوک داشته. به طور کلی می‌توان گفت که حوزهٔ یک متغیر از محل اعلان آن شروع می‌شود و تا پایان همان بلوک ادامه می‌یابد. خارج از آن بلوک نمی‌توان به متغیر دسترسی داشت. همچنین قبل از این که متغیر اعلان شود نمی‌توان آن را استفاده نمود. مثال زیر را بررسی کنید.

× مثال 17 – 2 حوزهٔ متغیرها

برنامهٔ زیر خطادار است:

```
int main()
{  //illustrates the scope of variables:
   x = 11;        // ERROR: this is not in the scope of x
   int x;
```

```
{
    x = 22;     // OK: this is in the scope of x
    y = 33;     // ERROR: this is not in the scope of y
    int y;
```

---

1 – Scope 2 - Block

```
    x = 44;     // OK: this is in the scope of x
    y = 55;     // OK: this is in the scope of y
}
x = 66;         // OK: this is in the scope of x
y = 77;         // ERROR: this is not in the scope of y
return 0;
}
```

برنامهٔ بالا دو بلوک تودرتو دارد. اولین بلوک بعد از عبارت `int main()` شروع می‌شود و در خط آخر برنامه بسته می‌شود. بلوک داخلی نیز از خط پنجم آغاز می‌شود و در خط دهم پایان می‌یابد. نحوهٔ تورفتگی خطوط برنامه به درک و تشخیص شروع و پایان بلوک‌ها کمک می‌کند. خط پنجم تا دهم تورفتگی بیشتری دارد، یعنی این خطوط تشکیل یک بلوک می‌دهند. همچنین خط دهم به بعد تورفتگی به اندازهٔ خط سوم و چهارم دارد، یعنی مجموعه این خطوط هم در یک حوزهٔ مشترک قرار دارند.

اولین خطا در خط سوم رخ داده. متغیر `x` در خط چهارم اعلان شده است. پس حوزهٔ `x` از خط چهارم به بعد شروع می‌شود، در حالی که در خط سوم متغیر `x` فراخوانی شده و این خارج از محدودهٔ `x` است.

دومین خطا در خط ششم اتفاق افتاده است. متغیر `y` در خط هفتم اعلان شده. پس حوزهٔ `y` از خط هفتم به بعد است، در حالی که در خط ششم `y` فراخوانی شده و این خارج از محدودهٔ `y` است.

سومین خطا که در خط دوازدهم روی داده نیز مربوط به `y` است. گرچه `y` در خطوط قبلی تعریف شده اما این تعریف در یک بلوک داخلی بوده است. این بلوک داخلی در خط دهم به پایان رسیده است. پس تمام تعاریفی که در این بلوک وجود

داشته نیز فقط تا خط دهم اعتبار دارد. یعنی حوزهٔ y فقط از خط هفتم تا خط دهم است. لذا نمی‌توان در خط دوازدهم که خارج از محدوده y است، آن را به کار برد.

مثال بالا مطلب ظریفی را بیان می‌کند: می‌توانیم در یک برنامه، چند متغیر متفاوت با یک نام داشته باشیم به شرطی که در حوزه‌های مشترک نباشند. آخرین برنامهٔ فصل اول این موضوع را به خوبی نشان می‌دهد.

× مثال 18 – 2 متغیرهای تودرتو

```
int x = 11;                          // this x is global

int main()
{  //illustrates the nested and parallel scopes:
   int x = 22;
   {  //begin scope of internal block
      int x = 33;
      cout << "In block inside main() : x = " << x << endl;
   }                                 //end scope of internal block
   cout << "In main() : x = " << x << endl;
   cout << "In main() : ::x = " << ::x << endl;
   return 0;
}                                    //end scope of main()
```

```
In block inside main() : x = 33
In main() : x = 22
In main() : ::x = 11
```

در برنامهٔ بالا سه شیء متفاوت با نام x وجود دارد. اولین x که مقدار 11 دارد یک متغیر سراسری است زیرا داخل هیچ بلوکی قرار ندارد. پس حوزهٔ آن سراسر برنامه (حتی خارج از بلوک main()) است. دومین x درون بلوک main() با مقدار 22 تعریف شده است. پس حوزهٔ آن تا پایان بلوک main() است. این x حوزه x قبلی را کور می‌کند. یعنی درون بلوک main() فقط x دوم دیده می‌شود و x اول مخفی می‌شود. پس اگر درون این بلوک به x ارجاع کنیم فقط x دوم را خواهیم دید.

سومین x در یک بلوک داخلی تعریف شده است. حوزه این x فقط تا پایان همان بلوک است. این x حوزهٔ هر دو x قبلی را کور می‌کند. پس اگر درون این بلوک

x را فراخوانی کنیم فقط x سوم را خواهیم دید. وقتی از این بلوک خارج شویم ، x قبلی آزاد می‌شود و دوباره می‌توان به مقدار آن دسترسی داشت. اگر از بلوک main() نیز خارج شویم ، x اول آزاد خواهد شد. برنامه را از اول دنبال کنید و خروجی را بررسی نمایید تا متوجه شوید که کدام x معتبر بوده است. در خط دهم از عملگر :: استفاده شده. به آن عملگر «جداسازی حوزه» می‌گویند. این عملگر را در فصل‌های بعدی بررسی می‌کنیم. در این‌جا فقط می‌گوییم با عملگر جداسازی حوزه می‌توان به یک شی که خارج از حوزهٔ فعلی است دسترسی پیدا کنیم. پس x:: یعنی متغیر x که در حوزهٔ بیرونی است.

می‌بینید که تعریف چند متغیر در یک برنامه با نام یکسان به شیوهٔ بالا ممکن و مجاز است. اما سعی کنید از این کار اجتناب کنید زیرا در غیر این صورت همیشه مجبورید به خاطر بسپارید که الان داخل کدام حوزه هستید و کدام متغیر مورد نظر شماست. این طوری راحت‌ترید؟ اختیار با شماست!

پرسش‌های گزینه‌ای

1 – از میان انواع زیر، کدام یک نوع صحیح حساب نمی‌شود؟

الف) double ب) int ج) char د) bool

2 – اگر m و n هر دو از نوع short باشند و m=6 و n=4 باشد، آنگاه حاصل m/n برابر است با:

الف) 1 ب) 1.5 ج) 2 د) nan

3 – اگر m از نوع double و n از نوع int باشد و m=6.0 و n=4 باشد، حاصل m/n چقدر است؟

الف) 1 ب) 1.5 ج) 2 د) nan

4 – متغیر m از نوع float و متغیر n از نوع short است. اگر بخواهیم حاصل m*n را در متغیری به نام k نگهداریم، آنگاه k باید از نوع ..... باشد.

الف) short ب) long

ج) float د) int

5 – در عبارت z += ++y اگر مقدار اولیهٔ y برابر با 5 و مقدار اولیهٔ z برابر با 7 باشد،حاصل z پس از اجرای آن دستور عبارت است از:

الف) 12 ب) 6 ج) 5 د) 13

6 – عدد 1.23e-1 معادل کدام یک از اعداد زیر است؟

الف) 12.3 ب) 12.3- ج) 0.123 د) 1.23-

7 – اگر a متغیری از نوع float با مقدار 5.63 باشد، آنگاه حاصل int(a) برابر است با:

الف) 5 ب) 6 ج) 0.63 د) 5.6

8 – برای این که حاصل m/n از نوع صحیح باشد باید:

الف) m از نوع صحیح باشد ب) n از نوع صحیح باشد

ج) یا m و یا n از نوع صحیح باشد د) هم m و هم n از نوع صحیح باشد

9 – برای تعریف انواع شمارشی از چه کلمهٔ کلیدی استفاده می‌شود؟

الف) include ب) enum ج) sqrt د) const

10 – اگر بخواهیم کاراکتر M را درون متغیر ch که از نوع کاراکتری است بگذاریم از چه دستوری استفاده می‌کنیم؟

الف) `ch = M;` ب) `ch = "M";`

ج) `ch = 'M';` د) `ch M`

11 – خطای گرد کردن مربوط به کدام نوع در C++ است؟

الف) نوع ممیز شناور ب) نوع صحیح

ج) نوع کاراکتری د) نوع شمارشی

12 – اگر یک متغیر از نوع int سرریز شود، چه مقداری در آن قرار می‌گیرد؟

الف) `-inf` ب) `inf` ج) `nan` د) عدد صحیح منفی

پرسش‌های تشریحی

1- قبل از اجرای دستورات زیر، مقدار m برابر 5 و مقدار n برابر 2 است. بعد از اجرای هر یک از دستورات زیر مقدار جدید m و n چیست؟

a. `m *= n++;`
b. `m += --n;`

2- مقدار هر یک از عبارات زیر را پس از مقداردهی برآورد کنید. ابتدا فرض کنید که m برابر 25 و n برابر 7 است.

a. `m - 8 - n`
b. `m = n = 3`
c. `m%n`
d. `m%n++`
e. `m%++n`
f. `++m - n--`

3- دو دستور زیر چه تفاوتی با هم دارند؟

```
char ch = 'A';
char ch = 65;
```

4- برای پیدا کردن کاراکتری که کد اسکی آن 100 است، چه کدی را می‌توانید اجرا کنید؟

5- معنای «ممیز شناور» چیست و چرا به این نام نامیده می‌شود؟

6- سرریزی عددی چیست؟

7- فرق سرریزی عدد صحیح با سرریزی عدد ممیز شناور چیست؟

8- خطای زمان اجرا چیست؟ مثال‌هایی برای دو نوع متفاوت از خطاهای زمان اجرا بنویسید.

9- خطای زمان کامپایل چیست؟ مثال‌هایی برای دو نوع متفاوت از خطاهای زمان کامپایل بنویسید.

10- کد زیر چه اشتباهی دارد؟

```
enum Semester {FALL, SPRING, SUMMER};
enum Season {SPRING, SUMMER, FALL, WINTER};
```

11- کد زیر چه اشتباهی دارد؟

```
enum Friends {"Jerry", "Henry", "W.D"};
```

تمرین‌های برنامه‌نویسی

1- چهار دستور متفاوت C++ بنویسید که 1 را از متغیر عدد صحیح n کم کند.

2- یک بلوک کد C++ بنویسید که مشابه جملهٔ زیر عمل کند بدون این که از عملگر ++ استفاده کنید.

```
n = 100 + m++;
```

3- یک بلوک کد C++ بنویسید که مشابه جملهٔ زیر عمل کند بدون این که از عملگر ++ استفاده کنید.

```
n = 100 + ++m;
```

4- یک دستور C++ تکی بنویسید که مجموع x و y را از z کم کند و سپس y را افزایش دهد.

5- یک دستور C++ تکی بنویسید که متغیر n را کاهش بدهد و سپس آن را به total اضافه کند.

7- برنامه‌ای را نوشته و اجرا کنید که موجب خطای پاریزی متغیری از نوع short شود.

8- برنامه‌ای مانند مثال 8 – 2 نوشته و اجرا کنید که تنها کد اسکی ده حرف صدادار بزرگ و کوچک را چاپ می‌کند.(برای بررسی خروجی، از ضمیمهٔ «الف» استفاده کنید)

9- برنامهٔ مثال 15-2 را طوری تغییر دهید که از نوع double به جای float استفاده کند. سپس مشاهده کنید که این برنامه چطور با ورودی‌هایی که خطای زمان اجرا را نشان می‌دهند، بهتر اجرا می‌شود.

10- برنامه‌ای بنویسید که اینچ را به سانتیمتر تبدیل کند. برای مثال اگر کاربر 16.9 را برای یک طول بر حسب اینچ وارد کند، خروجی 42.946 cm چاپ شود.(یک اینچ برابر 2.54 سانتیمتر است)