۱-۳- متغیرها

در برنامه اول از کاربر خواستیم تا نام خود را وارد کند. سپس برنامه کلمه Hello را همراه با نام کاربر چاپ کرد. کامپیوتر میداند که بعد از Hello چه اسمی را قرار دهد، چون ما نام وارد شده توسط کاربر را توسط دستور زیر ذخیره کردیم :

```
std::cin >> firstName;
```

دستور std::cin >> از کاربر میخواهد تا متنی را وارد کنید و پس از وارد شدن متن، برنامه این متن را در یک متغیر رشته ای ذخیره میکند که firstName نام دارد. چون رشته ذخیره شده است، میتوانیم این رشته را به کمک خط زیر به خروجی بفرستیم :

```
std::cout << "Hello, " << firstName << std::endl << std::endl;
```

یک متغیر از ناحیه ای از حافظه فیزیکی سیستم استفاده میکند و مقداری از نوع مشخص را ذخیره میکند. چندین نوع متغیر در C++ وجود دارد که به ما امکان ذخیره سازی انواع مختلفی از مقادیر را میدهد، و بطوریکه بعدها خواهید آموخت، میتوانیم حتی انواع متغیر مخصوص خودمان را نیز بسازیم. جدول زیر جمع بندی کلی از انواع متغیر در C++ است :

نوع متغیر	توضیح
std::string	برای ذخیره سازی متغیرهای رشتخ ای استفاده میشود. دقت کنید که std::string بخشی از هسته زبان نیست، بلکه بخشی از کتابخانه استاندارد است.
char	برای ذخیره سازی یک کاراکتر منفرد مانند 'a','b','c',... استفاده میشود. معمولاً یک مقدار ۸ بیتی است که میتواند تا ۲۵۶ مقدار را در خود داشته باشد (برای نمایش مجموعه کاراکترهای توسعه یافته ASCII کافی است). ضمیمه الف مربوط به جدول اسکی است. همانطور که در جدول اسکی مشاهده میشود، کاراکترها با اعداد صحیح نمایش داده می‌شوند، پس نوع char در واقع یک عدد صحیح است، ولی بصورت کاراکتر تفسیر میشود.
int	نوع اصلی مورد استفاده برای ذخیره سازی اعداد صحیح است.
short	معمولاً محدوده کوتاه تری از اعداد صحیح را نسبت به int نگهداری میکند.
long	یک عدد صحیح علامت دار که معمولاً محدوده بالاتری را نسبت به int نگهداری میکند.
float	برای ذخیره سازی اعداد ممیز شناور استفاده میشود، که اعدادی مانند ۵۲/۲۵۱ و ۸/۲۵۴- از این جمله میباشند.
double	مانند float، ولی معمولاً اعداد ممیز شناور را با دقت بالاتری نسبت به float نگهداری میکند.
bool	برای ذخیره سازی مقادیر درستی استفاده میشود که درست یا نادرست میباشد. دقت کنید که true و false کلیدواژه های C++ میباشند. همچنین توجه کنید که در C++، صفر یعنی نادرست و هر رقم غیر صفر دیگر چع منفی و چه مثبت، یعنی درست.

محدوده دقیق مقادیر این نوع متغیرها یا مقدار حافظه ای که هر نوع اشغال میکند، در جدول بالا آورده نشده است چرا که این مقادیر اکثراً وابسته به بستر اجرای برنامه میباشند. یک کاراکتر ممکن است روی برخی بسترها ۸ بیت (یک بایت) باشد و روی برخی دیگر ۳۲ بیتی باشد. پس، شما بهتر است جهت قابل حمل تر بودن کد، در این مورد گمانه زنی نداشته باشید. برای مثال، اگر شکا فرض کنید که در کد شما char ها ۸ بیتی هستند و سپس برنامه خود را به یک سیستم دیگر با char های ۳۲ بیتی ببرید، با خطا مواجه خواهید شد.

توجه کنید که میتوان از std:cin و std:cout برای ورودی/خروجی دیگر انواع نیز استفاده کرد. برای مثال، برنامه زیر را در نظر بگیرید.

برنامه ۲ – ورودی/خروجی

```
// برنامه از کاربر میخواهد تا انواع مختلفی از مقادیر را وارد کند
// سپس این مقادیر را در خروجی کنسول چاپ میکند
#include <iostream>
#include <string>
using namespace std;
int main()
{
// تعیین نوع متغیرها و دادن مقادیر اولیه به آنها
char letter = 'A';
int integer = 0;
float dec = 0.0f;
cout << "Enter a letter: "; // خروجی
cin >> letter; // ورودی
cout << "Enter an integer: ";// خروجی
cin >> integer; // ورودی
cout << "Enter a float number: "; // خروجی
cin >> dec; // ورودی
cout << endl; // سرخط
// چاپ مقادیر گرفته شده در پنجره کنسول
cout << "Letter: " << letter << endl;
cout << "Integer: " << integer << endl;
cout << "Float: " << dec << endl;
}
```

و خروجی برنامه بالا بصورت زیر است :

```
Enter a letter: F
Enter an integer: 100
Enter a float number: -5.987123

Letter: F
Integer: 100
Float: -5.98712
Press any key to continue
```

۱-۳-۱- تعیین و تعریف متغیر

وقتی ما کد زیر را نوشتیم، منظورمان این است که ما یک متغیر را تعیین میکنیم.

```
int myVar;
```

در اینجا ما متغیری بنام myVar از نوع int تعیین کردیم. با اینکه myVar تعیین شده است، ولی تعریف نشده است. یعنی مقداری که درون این متغیر قرار دارد، ناشناخته است. معمولاً در اصطلاح برنامه نویسی میگویند که متغیر دارای مقدار زباله است. این برنامه کوچک را نوشته و اجرا کنید تا متوجه منظور من شوید :

```
#include <iostream>
using namespace std;
int main()
{
int myVar;
cout << myVar << endl;
}
```

وقتی مقداری را به متغیر میدهیم، آنرا تعریف یا آغازش می‌کنیم.

```
myVar = 10;
```

سمبل '=' در C++ عملگر تخصیص نام دارد. دقت کنید که عملگر تخصیص مقادیر را به متغیرها تخصیص میدهد و مفهوم مقایسه ای منطقی مورد نظر ما را ندارد. این نکته برای دانش آموزان مبتدی C++ بسیار گیج کننده است.

همچنین میتوان یک متغیر را همزمان هم تعیین و هم تعریف کرد :

```
int myVar = 10;
```

چون متغیرهایی که شامل مقادیر زباله باشند، هیچ استفاده ای ندارند و مستعد ایجاد خطا می‌باشند، پس توصیه من به شما این است که همیشه متغیرهای خود را در زمان تعیین توسط یک مقدار پیش فرض تعریف کنید. معمولاً، صفر مقدار پیش فرض بسیار خوبی برای مقادیر عددی است و یک رشته خالی بصورت "" مقداری خوب برای رشته هاست.

این امکان وجود دارد که در یک عبارت چندین متغیر را تعریف و تعیین کرد که از عملگر کاما استفاده میشود :

```
int x, y, z;
x = 1, y = 2, z = 3;
float i = 1.0f, j = 2.0f, k = 3.0f;
```

علاوه بر اینکه این حالت بسیار فشرده تر و جمع و جورتر است، خوانایی برنامه را نیز بالاتر میبرد.

نهایتاً اینکه میتوان تخصیص های زنجیره ای ایجاد کرد :

```
float num = 0.0f;
float a, b, c, d;
a = b = c = d = num;
```

۱-۳-۲- اسامی متغیرها

بطوریکه در برنامه ۲ دیدید، هنگام تعریف/تعیین یک متغیر ما باید نامی (معرف) برای آن انتخاب کنیم و بدین ترتیب میتوانیم درون برنامه به آن متغیر دسترسی پیدا کنیم. اسامی متغیرها را میتوان هر چیزی انتخاب کرد ولی چند استثناء نیز وجود دارد :

۱ – اسامی متغیرها باید با یک حرف شروع شوند. نام 2VarName معتبر نیست. توجه کنید که خط زیر یک حرف محسوب میشود و بدین ترتیب معرف myVar_ معتبر است.

۲ – در اسامی متغیرها میتوان از خط زیر، حروف و اعداد استفاده کرد ولی دیگر سمبلها مجاز نیستند. برای مثال، نمیتوان از سمبلهایی مانند ... '%', '$', '#', '@', '!' استفاده کرد.

۳ – اسامی متغیرها نباید از کلیدواژه های C++ باشند (ضمیمه ب). برای مثال نمیتوان از نام float برای یک متغیر استفاده کرد.

۴ – اسامی متغیرها نباید دارای فاصله باشند.

نکته : *زبان C++ حساس به بزرگی و کوچکی حروف است. برای مثال، این معرفها همگی منحصر بفرد هستند چون ترکیب حروف بزرگ و کوچک آنها متفاوت است :*

hello, Hello, HElIo, heLLo, HeLlO

۱-۳-۳- عملگر sizeof

بطوریکه قبلاً نیز گفتم، محدوده مقادیر انواع C++ و میزان حافظه مصرفی آنها وابسته به بستر است. برای اینکه اندازه یک نوع خاص را در مقیاس بایت روی بستر جاری بدست آوریم، میتوانیم از عملگر sizeof استفاده کنیم. برنامه زیر را در نظر بگیرید که روی بستر ۳۲ بیتی ویندوز نوشته شده است :

برنامه ۳ – عملگر sizeof

```
// برنامه اندازه های انواع مختلف متغیرها را چاپ میکند
#include <iostream>
using namespace std;
int main()
{
cout << "sizeof(bool) = " << sizeof(bool) << endl;
cout << "sizeof(char) = " << sizeof(char) << endl;
cout << "sizeof(short) = " << sizeof(short) << endl;
cout << "sizeof(int) = " << sizeof(int) << endl;
cout << "sizeof(long) = " << sizeof(long) << endl;
cout << "sizeof(float) = " << sizeof(float) << endl;
cout << "sizeof(double) = " << sizeof(double) << endl;
}
```

و خروجی برنامه ۳ به شکل زیر است :

```
sizeof(bool)      = 1
sizeof(char)      = 1
sizeof(short)     = 2
sizeof(int)       = 4
sizeof(long)      = 4
sizeof(float)     = 4
sizeof(double)    = 8
Press any key to continue
```

دقت کنید که نتیجه مخصوص بستری است که برنامه روی آن اجرا شده است. در واقع، از این نتایج میتوان جدول زیر را استخراج کرد :

نوع	محدوده	تعداد بایت مورد نیاز
char	۱۲۸- الی ۱۲۷+	۱
short	۳۲۷۶۸- الی ۳۲۷۶۷	۲
int	۲۱۴۷۴۸۳۶۴۸- الی ۲۱۴۷۴۸۳۶۴۷+	۴
long	۲۱۴۷۴۸۳۶۴۸- الی ۲۱۴۷۴۸۳۶۴۷+	۴
float	۳/۴ e -۳۸ ± الی ۱/۲ e ۳۸ ±	۴
double	۱/۸ e -۳۰۸ ± الی ۲/۲ e ۳۰۸ ±	۸

توجه داشته باشید که هیچ تفاوتی بین محدوده حافظه های مورد نیاز برای int و long وجود ندارد.

۱-۳-۴- کلیدواژه unsigned

یک int محدوده ای از ۲۱۴۷۴۸۳۶۴۸- الی ۲۱۴۷۴۸۳۶۴۷+ را دربر می‌گیرد. بهرترتیب، اگر شما فقط نیاز به کار با مقادیر مثبت باشید، میتوان حافظه رزرو شده برای مقادیر منفی را آزاد کرد و از آن برای نمایش اعداد مثبت بیشتری استفاده کرد. هزینه این کار توانایی نمایش اعداد منفی است. میتوانیم این کار را با قرار دادن پیشوند انواع صحیح با unsigned انجام دهیم. در ویندوز ۳۲ بیتی ما موارد زیر را داریم :

نوع	محدوده	بایت های مورد نیاز
unsigned char	**۰ تا ۲۵۵**	**۱**
unsigned short	**۰ تا ۶۵۵۳۵**	**۲**
unsigned int	**۰ تا ۴۲۹۴۹۶۷۲۹۵**	**۴**
unsigned long	**۰ تا ۴۲۹۴۹۶۷۲۹۵**	**۴**

فقط انواع صحیح را میتوان unsigned کرد. با استفاده از انواع بدون علامت، ما محدوده بزرگتری را بدست نمی آوریم، بلکه آنرا جابجا میکنیم. برای مثال، نوع int ۴۲۹۴۹۶۷۲۹۶ مقدار منحصر بفرد را شامل میشود (با احتساب صفر) حال چه ما از محدوده ۲۱۴۷۴۸۳۶۴۸- تا ۲۱۴۷۴۸۳۶۴۷+ استفاده کنیم و چه از محدوده صفر تا ۴۲۹۴۹۶۷۲۹۵+

۱-۳-۵- تخصیص لیترال ها

لیترال ها مقادیری هستند که متغیر نیستند. برای مثال، ۱۰ یعنی ده. به همین ترتیب، hello world در معنی به مفهوم **سلام جهان** است. کد زیر نشان دهنده چند تخصیص لیترال است :

```
bool b = true; // لیترال بولی
char letter = 'Z'; // لیترال کاراکتری
std::string str = "Hello"; // لیترال رشته ای
int num = 5; // لیترال صحیح
unsigned int uint = 5U; // لیترال صحیح بدون علامت
long longNum = 10L; // لیترال طولانی
unsigned long ulong = 50UL; // لیترال طولانی بدون علامت
float floatNum = 123.987f; // لیترال ممیز شناور
double dblNum = 567.432; // لیترال دوبل
// توجه کنید که پسوند لیترالها حساس به اندازه حروف نیستند
// و بنابراین موارد زیر نیز درست کار میکنند
unsigned long ulong2 = 50ul; // لیترال طولانی بدون علامت
float floatNum2 = 123.987F; // لیترال ممیز شناور
```

در مورد اعداد ممیز شناور کمی ابهام وجود دارد. برای مثال، روشن نیست که آیا عدد ۱۲۳/۹۸۷ باید بصورت float در نظر گرفته شود یا double. برای جلوگیری از این ابهام، C++ این امکان را میدهد تا یک پسوند نوع به لیترال اختصاص دهید. اگر لیترال دارای پسوند f باشد و بصورت 123.987f نمایش داده شود، بصورت float در نظر گرفته میشود. در غیر این صورت بصورت double دیده میشود. مشکل مشابهی نیز در مورد long های بدون علامت رخ میدهد. به uint, longNum و ulong در کد بالا دقت کنید که ما از U، L و ترکیب UL استفاده کرده اینم تا نوع لیترال مشخص شود.

۱-۳-۶- تبدیل انواع

این امکان وجود دارد که متغیرهای از انواع مختلف را به هم تبدیل کنیم. برای مثال، میتوانیم یک مقدار int را بع یک مقدار float اختصاص دهیم و برعکس. در این صورت به مواردی نیز باید دقت شود. برنامه ۴ این مورد را نشان میدهد.

برنامه ۴ : تبدیل انواع

```
// نمایش تبدیل انواع
#include <iostream>
using namespace std;
int main()
{
// مورد اول : تبدیل از یک نوع با دقت پایین به دقت بالا
char c = 10;
short s = c;
cout << "char to short: " << s << endl;
// مورد دوم : تبدیل از نوع با دقت بالا به پایین
unsigned char uc = 256;
cout << "int to uchar: " << (int)uc << endl;
// مورد سوم : تبدیل از شناور به صحیح با فرض اینکه
// مقدار صحیح میتواند مقادیر شناور را نگهداری کند
int i = 496512.546f;
cout << "float to int: " << i << endl;
// مورد چهارم : تبدیل از شناور به کوتاه
// این بار صحیح قادر به نگهداری شناور نیست
s = 496512.987123f;
cout << "float to short: " << s << endl;
}
```

خروجی برنامه ۴

```
char to short: 10
int to uchar: 0
float to int: 496512
float to short: -27776
Press any key to continue
```

- مورد اول : در اینجا ما از مقداری با دقت پایین به مقداری با دقت بالا تبدیل انجام دادیم. با توجه به اینکه دقت بالا میتواند دقت پایین را بطور کامل نمایش دهد، پس هیچ مشکل تبدیلی وجود نخواهد داشت و همه چیز بخوبی کار خواهد کرد.
- مورد دوم : در این مورد ما از یک مقدار صحیح با دقت بالا به یک مقدار صحیح با دقت پایین تبدیل انجام میدهیم. با این حال، unsigned char قادر به نمایش مقدار ۲۵۶ نیست. اتفاقی که می افتد، Wrapping یا بازپیچی نامیده میشود. unsigned char قادر به ذخیره سازی مقادیر بیش از ۲۵۵ نیست و بنابراین به صفر بازپیچی میشود. پس ۲۵۶ صفر میشود، ۲۵۷ نبدیل به ۱ میشود، ۲۵۸ تبدیل به ۲ میشود و به همین ترتیب. بازپیچی در جهت مثبت انجام میشود. برای مثال، اگر ما به unsigned char مقدار ۱- را بدهیم، این مقدار در جهت دیگر بازپیچی شده و مقدار ۲۵۵ را ارائه میکند.
- مورد سوم : در اینجا ما یک float را به یک int تخصیص دادیم. می بینید که float در علمی بنام برش یا truncation قرار گرفته است و قسمت اعشار خود را از دست داده است.
- مورد چهارم : اینجا یک float را به یک short تخصیص داده ایم، ولی short قادر به ذخیره سازی کل قسمت های عددی float نیست. پس ما یک بازپیچی خواهیم داشت.

نکته : *بازپیچی صحیح مشکلی جدی است که میتواند منجر به خطاهایی شود که یافتن آنها بسیار مشکل است. پس، شما همیشه باید اطمینان حاصل کنید که با مقادیری کار میکنید که انواع شما قادر به ذخیره سازی آنها می‌باشند.*

کامپایلر C++ این تبدیل انواع را بدون قید و شرط انجام میدهد یعنی بصورت خودکار انجام می-گیرد. بهرترتیب، گاهی نیاز خواهید داشت تا با کامپایلر بگویید که با یک نوع بصورت دیگری رفتار کند. میتوانیم این مورد را در مورد دوم از برنامه ۴ مشاهده کنیم. مخصوصاً در این خط :

```
cout << "int to uchar: " << (int)uc << endl;
```

دستور (int)uc به کامپایلر میگوید که با uc بصورت یک int رفتار کند، نه یک char. علت این است که cout نمایش کاراکتری uc را نمایش میدهد چون یک char است. ولی ما میخواهیم خروجی صحیح معادل کاراکتر نمایش داده شود. پس ما یک اعمال نوع (تبدیل بین انواع) انجام داده ایم و به کامپایلر گفته ایم که با uc بعنوان یک int رفتار کند. بطور کلی، این نوع از قالب بندی را میتوان با یکی از حالت های زیر بدست آورد :

```
result = static_cast<typeToConvertTo>(valueToConvert);
result = (typeToConvertTo) valueToConvert;
```

مثال :

```
int x = 5;
float result = static_cast<float>(x);
int y = (int)result;
```

۱-۳-۷- Typedef ها

گاهی، ممکن است انواع C++ بسیار طولانی باشند. برای مثال، unsigned int آنقدر طولانی است که نوشتن آن به سرعت سخت باشد. C++ این قابلیت را دارد که بتوانید در آن نامهای کوتاهی (مترادف) توسط کلیدواژه typedef تعریف کنید. برای مثال، ممکن است نامهای کوتاه زیر را برای انواع بدون علامت تعریف کنیم :

```
typedef unsigned char uchar;
typedef unsigned short ushort;
typedef unsigned int uint;
typedef unsigned long ulong;
```

پس، برای مثال، بجای اینکه بنویسیم :

```
unsigned int x = 19;
```

خواهیم نوشت :

```
uint x = 19;
```

۱-۳-۸- متغیرهای Const

گاهی میخواهیم متغیری را تعریف کنیم که قابل تغییر نباشد. چنین متغیرهایی را ثابت می نامیم. برای مثال، شاید بخواهیم ثابتی بنام Pi تعریف کنید که حاوی عدد پی ۳/۱۴ باشد. برای انجام این کار از کلیدواژه const استفاده میکنیم :

```
const float pi = 3.14f;
```

اگر برنامه نویس قصد تغییر pi را داشته باشد، با خطا موجه خواهد شد. ثابت ها معمولاً برای راحتی و دقت مورد استفاده قرار می‌گیرند. خواندن pi خیلی راحت تر از خواندن ۳/۱۴ است.

۱-۳-۹- ماکروها

گاهی قصد داریم تا یک نام سمبل (معرف) را ایجاد کنیم که بجای یک کد قرار گیرد. این کار با تعریف یک ماکرو انجام می‌گیرد. برای مثال، خط زیر یک ماکرو بنام PRINTHELLO تعریف میکند که وقتی نوشته شود، دستور معادل : cout << "Hello" << endl; را اجرا میکند.

```
#define PRINTHELLO cout << "Hello" << endl;
```

با استفاده از ماکروها، ما برنامه ای نوشتیم که چندین بار کلمه Hello را چاپ کند.

برنامه ۵ : ماکروها

```
#include <iostream>
using namespace std;
// Define a macro PRINTHELLO that means
// "cout << 'Hello' << endl;"
#define PRINTHELLO cout << "Hello" << endl;
int main()
{
PRINTHELLO
PRINTHELLO
PRINTHELLO
PRINTHELLO
}
```

دقت کنید که سمی کولون در تعرف ماکرو وجود دارد و بنابراین نیازی نیست در انتهای هر خط یکی قرار دهیم. وقتی کامپایلر این برنامه را کامپایل میکند و به یک ماکرو میرسد، کد مربوطه را جایگزین میکند. پس برنامه بالا بصورت زیر تغییر میکند :

```
int main()
{
      cout << "Hello" << endl;
      cout << "Hello" << endl;
      cout << "Hello" << endl;
      cout << "Hello" << endl;
}
```

-:- هدف این سایت ارتقای دانش کاربران ایرانی است، لذا انتشار این مطلب کاملاً آزاد است -:-
-:- ذکر منبع انتشار نشانگر احترام شما به فعالیت های ایرانی و مشوقی برای ادامه کار است -:-
-:- منبع انتشار مطلب با عنوان : دنیای زیبای کامپیوتر و پیوند : www.pcnova.ir میباشد -:-