مبانی کامپیوتر و برنامه سازی

اسلاید هشتم

«برنامه نویسی مقدماتی به زبان C++»

بخش اول

درس

مبانی کامپیوتر

۸

محمد سعید صفایی صادق

۱۴۰۲

تقدیر و تشکر

بخش عمده ای از این اسلاید، برداشتی از اسلاید های استاد محترم، **سرکار خانم دکتر لیلا اسماعیلی** می باشد، لذا مراتب قدردانی و سپاس خود را از حسن همکاری و زحمات بی شائبه این استاد والامقام اعلام نموده و از خداوند متعال توفیق و سربلندی بیش از پیش، برای ایشان مسئلت می‌نمایم.

محمد سعید صفایی صادق

مقدمات برنامه نویسی

تعریف برنامه

برنامه رایانه‌ای Computer Program :

مجموعه‌ای از الگوریتم یا دستورالعمل‌ها است که رایانه برای انجام یک کار مشخّص آن را اجرا می‌کند.

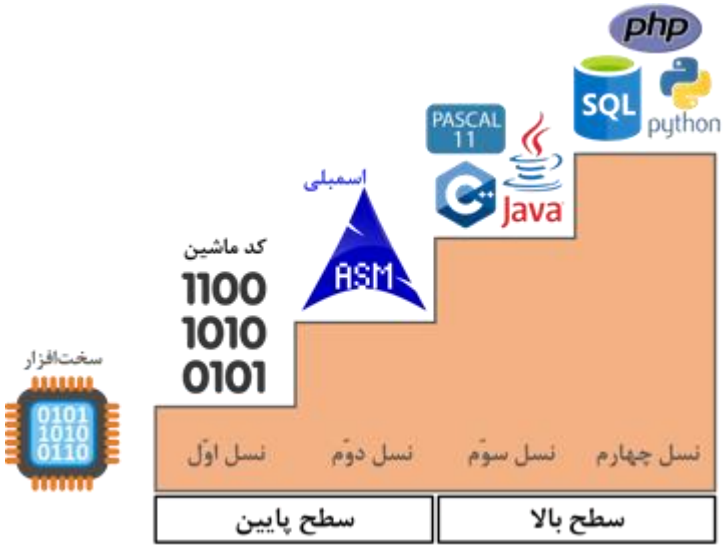
رایانه برای انجام کارهایش به برنامه‌ها نیاز دارد و معمولاً هر برنامه را در یک واحد پردازش مرکزی اجرا می‌کند.

```cpp
#include <iostream>
int main(){
    std::cout << "Hello World" << std::endl;
    return 0;
}
```

تولید برنامه

به فرایند نوشتن و ویرایش کد منبع، برنامه‌نویسی گفته می‌شود.

گاهی به فرایند طولانی مدّت برنامه‌نویسی، توسعه نرم‌افزار گفته می‌شود.



می‌توان برنامه‌ها را با توجّه به نوع زبان برنامه‌نویسی دسته‌بندی کرد.

۱- سطح بالا ۲- سطح پایین (که در اسلاید قبل توضیحات داده شده است)

نحوه اجرای برنامه

قسمتی از برنامه که برای انسان قابل درک است کد منبع و قسمتی که برای رایانه مستقیماً قابل اجرا است، کد ماشین نامیده می‌شود.

وظیفه تبدیل کد منبع به کد ماشین به عهده کامپایلر، مفسّر یا اسمبلر است.

پس برای نوشتن و اجرای هر برنامه به یک «ویرایش‌گر متن» و یک «کامپایلر» احتیاج داریم.

بستهٔ Visual C++محصول شرکت میکروسافت و بستهٔ C++ Builder محصول شرکت بورلند و Turbo C++، نمونه‌های جالبی از محیط مجتمع تولید برای زبان C++به شمار می‌روند.

محیط مجتمع تولید DEV C++

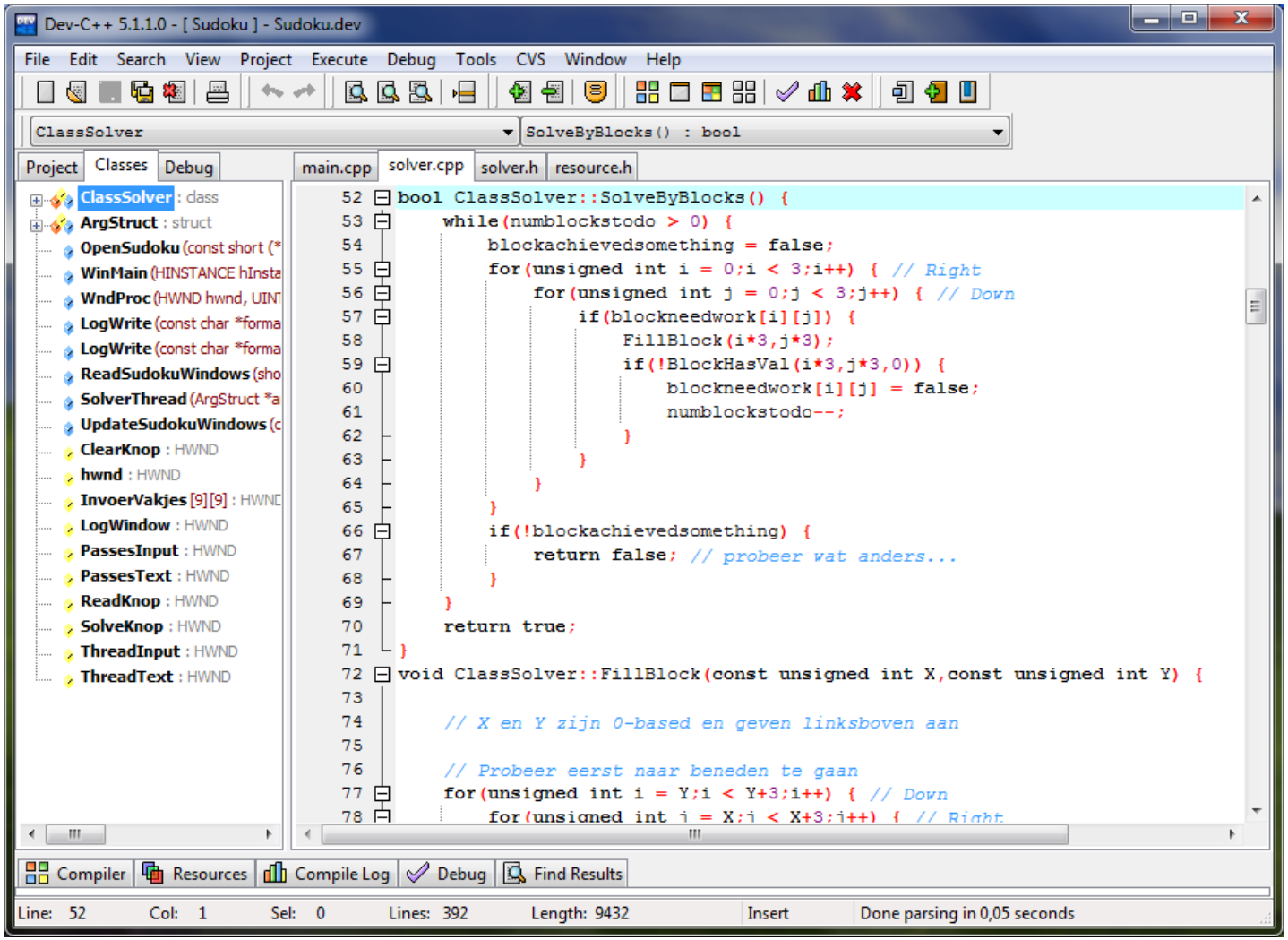
یک (IDE) محیط یکپارچه توسعه نرم‌افزار آزاد است که تحت پروانه عمومی همگانی GNU (General Public License) و برای برنامه‌نویسی در زبان‌های C و C++ منتشر می‌شود.

این بسته همراه با پورت MinGW یا TDM-GCC 64bit از GCC به عنوان کامپایلر آن استفاده می‌شود.

نکته ای که باید مورد توجه قرار دهید تفاوت IDE با کامپایلر است.

در واقع IDE (Integrated development environment) یک نرم افزار محیط توسعه یکپارچه است، که در آن کلیه امکانات لازم برای برنامه نویسی نظیر کامپایلر، دیباگر، ادیتور و ... قرار داده شده است، این درحالی است که کامپایلر یک برنامه زبان ترجمه است.

محیط مجتمع تولید DEV C++



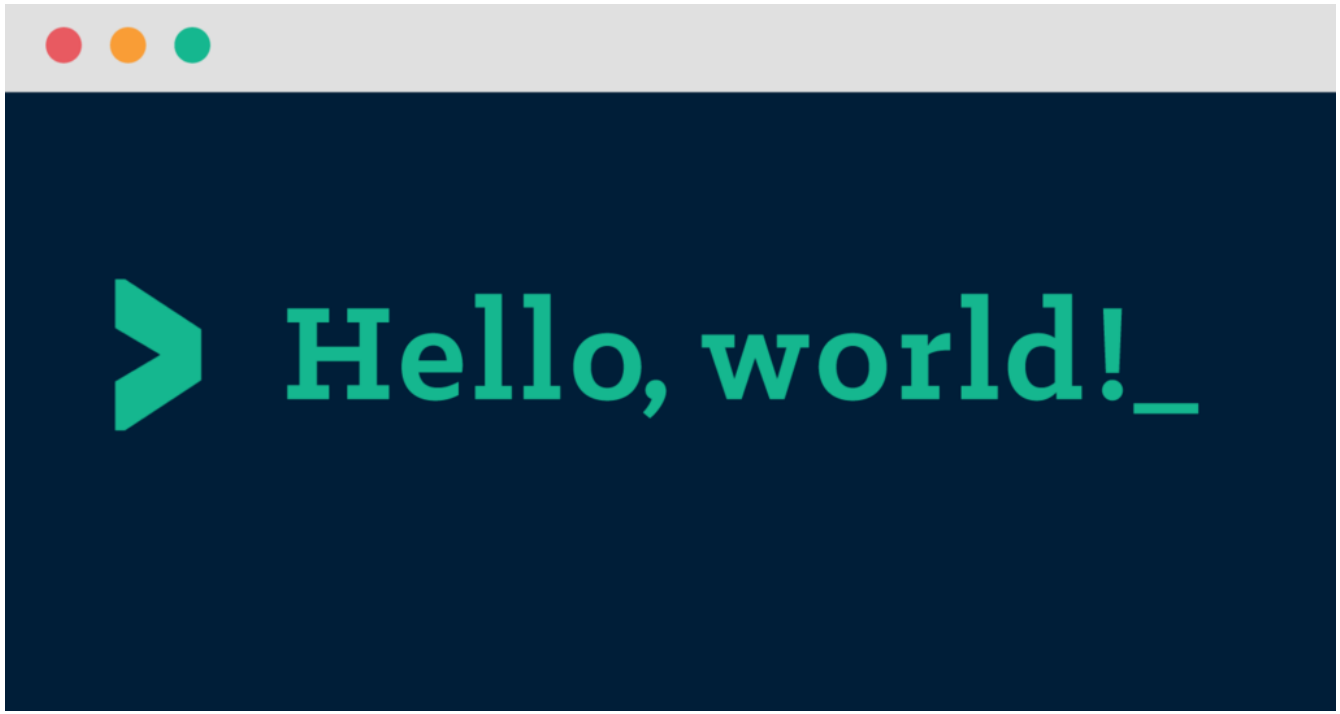
شروع کار با C++

دستورات مقدماتی

اولین برنامه با C++

نخستین برنامه C++ ما یک برنامه Hello, World!خواهد بود.

این برنامه‌ای است که در همه زبان‌های برنامه‌نویسی به عنوان نخستین برنامه نوشته می‌شود.



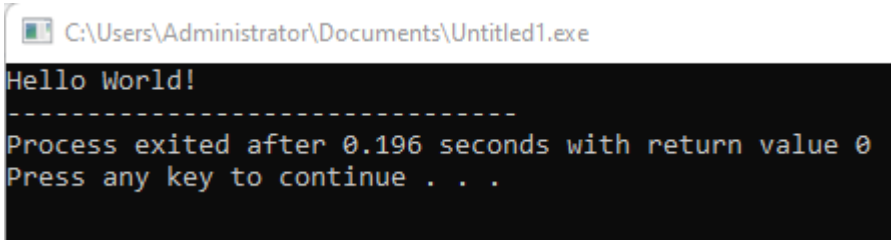
اولین برنامه با C++

پس شروع به برنامه نویسی می کنیم :

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      cout<<"Hello World!";
6      return 0;
7  }
```

این برنامه عبارت Hello World! را در صفحه خروجی نمایش می‌دهد.

برای کامپایل و اجرا و نمایش خروجی در DEV از کلید میانبر F11 استفاده میگیریم.



اولین خط از کد یک «راهنمای پیش‌پردازنده» است.

راهنمای پیش‌پردازنده شامل اجزای زیر است:

۱- کاراکتر # که نشان می‌دهد این خط، یک راهنمای پیش‌پردازنده است.

این کاراکتر باید در ابتدای همهٔ خطوط راهنمای پیش‌پردازنده باشد.

۲- عبارت include

۳- نام یک «فایل کتابخانه‌ای» که میان دو علامت <> محصور شده است.

۱-۱ #include چیست؟

برای گنجاندن فایل iostream در برنامه از دستور #include استفاده می‌شود.

این دستور تضمین می‌کند که می‌توان از عملیات درون فایل iostream مانند گرفتن ورودی از کاربر، نمایش خروجی روی صفحه در برنامه استفاده کرد.

۱-۲ iostream به چه معنا است؟

iostream نام فایل هدر ما است. این یک فایل کتابخانه استاندارد ورودی/خروجی C ++است.

این کتابخانه به همراه کامپایلر و IDE می‌آید و شامل سازوکارهایی است که اطلاعات را از کاربر می‌گیرد و خروجی را در یک فایل، صفحه نمایش یا هر رسانه دیگری نمایش می‌دهد.

۲- using namespace std به چه معنا است؟

این گزاره خود گویا است چون ما به وسیله آن از فضای نامی به نام std در برنامه استفاده می‌کنیم.

ما از namespace std برای تسهیل ارجاع به عملیات موجود در آن فضای نام بهره می‌گیریم.

اگر از این فضای نام استفاده نکنیم، باید به جای cout از std::cout استفاده کنیم.

این دستور به کامپایلر اعلام می‌کند که هر coutدر واقع یک std::cout است.

Std فضای نام استاندارد مورد استفاده از C++ است.

فضای نام به چه معنا است؟

این همان جایی است که کد برنامه قرار می‌گیرد.

تمام این کدهایی که مینویسیم در فایل ذخیره می شوند.

این گزینه دامنه کد شما را به یک یا چند فایل محدود کرده یا گسترش می‌دهد.

چرا باید از فضای نام استفاده کرد؟

همان طور که دو فرد ممکن است نام مشابهی داشته باشند، متغیرها و تابع‌ها نیز در C++ ممکن است نام‌های یکسانی داشته باشند.

استفاده از فضای نام برای جلوگیری از تداخل متغیرها و تابع‌ها و ارجاع صحیح به هر کدام از آن‌ها است.

نقطه‌ویرگول (;) سمیکالون

اگر از هر برنامه‌نویسی بپرسید، دست‌کم یک داستان ترسناک در مورد نقطه‌ویرگول برای شما تعریف می‌کند.

نقطه‌ویرگول یک کاراکتر پایانی محسوب می‌شود.

این کاراکتر یک گزاره را می‌بندد.

زمانی که آن را فراموش کنید یا به صورت نادرستی از آن استفاده کنید، موجب مشکلات زیادی خواهد شد.

```
int main() {}
```

این خط به کامپایلر می‌گوید که «بدنهٔ اصلی برنامه» از کجا شروع می‌شود.

همان طور که از نام این گزاره برمی‌آید، تابع اصلی برنامه است.

کد درون {} به نام «بدنه» body نامیده می‌شود و زمانی که برنامه C++ را اجرا کنید، قبل از همه اجرا می‌شود.

این تنها کدی است که در برنامه‌های C++ وجود آن اجباری است.

اگر تنها این کد را در برنامه خود داشته باشید، از نظر کامپایلر برنامه شما معتبر است.

دو پرانتز () نشان می‌دهد عبارت main یک «تابع» است.

هر برنامه فقط باید یک تابع main() داشته باشد .

```
cout<<"Hello World!";
```

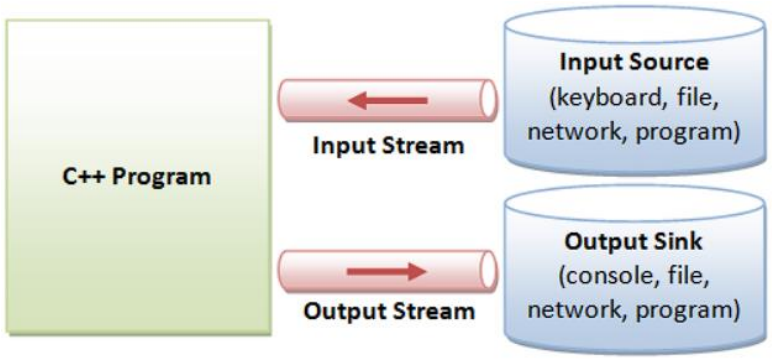
این گزاره عبارت Hello World! را در صفحه خروجی نمایش می‌دهد.

cout شیء استاندارد stream خروجی است.

این بدان معنی است که وظیفه آن خروجی/پرینت داده‌های موجود پس از >> در یک stream است.

Stream اساساً به یک توالی از اشیا و به طور معمول بایت‌ها گفته می‌شود. **(چندتا متغیر باهم چاپ یا دریافت شود)**

این توالی می‌تواند یک فایل، ترمینال ورودی/خروجی، سوکت و موارد دیگر را تعریف کند.



>> به چه معنا است؟

>> عملگر درج (یا خروجی) است که برای نوشتن داده‌های قالب‌بندی شده در stream استفاده می‌شود.

یک «عملگر» چیزی است که عملیاتی را روی یک یا چند شی object انجام می‌دهد.

عملگر خروجی، مقادیر موجود در سمت راستش را به خروجی سمت چپش می‌فرستد.

برای مثال دستور:

```
cout<< 66 ;
```

مقدار ۶۶ را به خروجی cout می‌فرستد که cout معمولا به صفحه نمایش اشاره دارد.

در نتیجه مقدار ۶۶ روی صفحه نمایش درج می‌شود.

return 0 به چه معنا است؟

این گزاره مقدار ۰ را بازگشت می‌دهد.

گزاره‌های return اجباری نیستند و الزامی برای بازگشت دادن یک مقدار از تابع main() وجود ندارد، اما نوشتن آن یک قرارداد است.

اگر این گزاره ذکر نشود، کامپایلر به طور خودکار یک وضعیت را بازگشت می‌دهد.

چرا در گزاره بازگشت از ۰ استفاده می‌شود؟

با توجه به اینکه نوع تابع اصلی int یعنی عددصحیح است باید مقدار بازگشت ازین تابع نیز عدد باشد.

مقدار ۰ به معنی وضعیت Exit از برنامه است که اساساً به سیستم اعلام می‌کند برنامه کار خود را به پایان برده است.

توضیح یا کامنت به چه معنا است؟

توضیح، متنی است که به منظور راهنمایی و درک بهتر به برنامه اضافه می‌شود و تاثیری در اجرای برنامه ندارد.

کامپایلر توضیحات برنامه را قبل از اجرا حذف می‌کند و در واقع آن ها را اجرا نمی کند.

بنابراین توضیحات در برنامه بی اثر هستند.

استفاده از توضیح سبب می‌شود که سایر افراد کد برنامهٔ شما را راحت‌تر درک کنند.

به دو صورت می‌توانیم به برنامه‌های C++ توضیحات اضافه کنیم:

۱ – با استفاده از دو علامت اسلش // :

هر متنی که بعد از دو علامت اسلش بیاید تا پایان همان سطر یک توضیح تلقی می‌شود .

۲ – با استفاده از حالت چند خطی:

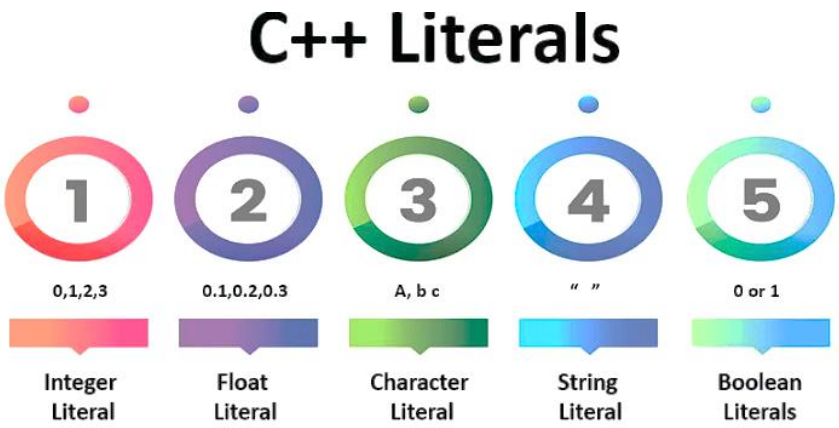
هر متنی که با علامت /* شروع شود و با علامت */ پایان یابد یک توضیح تلقی می‌شود.

```
//  Single line comment

/*  Multi-line comment  */
```

تفاوت لیترال رشته ای و کاراکتر

یک «لیترال» رشته، رشته‌ای از حروف، ارقام یا علایم چاپی است که میان دو علامت نقل قول " " محصور شده باشد.

یک «کاراکتر» یک حرف، رقم یا علامت قابل چاپ است که میان دونشانهٔ ' ' محصور شده باشد. پس 'w'و '!' و '۱' هر کدام یک کاراکتر است.



به تفاوت سه موجودیت «عدد» و «کاراکتر» و «لیترال رشته‌ای» دقت کنید: ۶ یک عدد است، '۶' یک کاراکتر است و "۶" یک لیترال رشته‌ای است.

متغیر و تعریف آن

«متغیر» مکانی در حافظه است که چهار مشخصه دارد:

نام، نوع، مقدار، آدرس

وقتی متغیری را تعریف می‌کنیم، ابتدا با توجه به نوع متغیر، آدرسی از حافظه در نظر گرفته می‌شود، سپس به آن آدرس یک نام تعلق می‌گیرد.

در C++ قبل از این که بتوانیم از متغیری استفاده کنیم، باید آن را اعلان نماییم.

یعنی باید آن را **تعریف** کنیم.

نحوه تعریف متغیر

type name *initializer;*

- عبارت **type** **نوع متغیر** را مشخص می‌کند. نوع متغیر به کامپایلر اطلاع می‌دهد که این متغیر چه مقادیری می‌تواند داشته باشد و چه اعمالی می‌توان روی آن انجام داد.

- عبارت **name** **نام متغیر** را نشان می‌دهد. این نام حداکثر می‌تواند ۳۱ کاراکتر باشد، نباید با عدد شروع شود، علایم ریاضی نداشته باشد و همچنین «کلمهٔ کلیدی» نیز نباشد.

- عبارت **initializer** /ɪˈnɪʃ.əl.aɪz/ عبارت «**مقداردهی اولیه**» نام دارد. با استفاده از این عبارت می‌توان مقدار اولیه‌ای در متغیر مورد نظر قرار داد.

کلمات کلیدی در C++

```
asm
auto
bool
break
case
catch
char
class
const
const_cast
continue
default
delete
do
double
dynamic_cast
else
enum
explicit
export
extern
and
and_eq
bitand
bitor
false
float
for
friend
goto
if
inline
int
long
mutable
namespace
new
operator
private
protected
public
register
reinterpret_cast
return
short
signed
compl
not
not_eq
or
sizeof
static
static_cast
struct
switch
template
this
throw
true
try
typedef
typeid
typename
union
unsigned
using
virtual
void
volatile
wchar_t
while
or_eq
xor
xor_eq
```

مقداردهی اولیه

دستور زیر تعریف یک متغیر صحیح را نشان می‌دهد:

```
int n = 50;
```

در بسیاری از موارد بهتر است متغیرها را در همان محلی که اعلان می‌شوند مقداردهی کنیم. استفاده از متغیرهای مقداردهی نشده ممکن است باعث ایجاد دردسرهایی شود.

دردسر متغیرهای مقداردهی نشده وقتی بزرگ‌تر می‌شود که سعی کنیم متغیر مقداردهی نشده را در یک محاسبه به کار ببریم. مثلا اگر x را که مقداردهی نشده در عبارت `y = x + 5;` به کار ببریم، حاصل y غیر قابل پیش‌بینی خواهد بود (زباله). برای اجتناب از چنین مشکلاتی عاقلانه است که متغیرها را همیشه هنگام تعریف، مقداردهی کنیم.

```
int x=45;
int y=0;
```

مقداردهی اولیه

برای این که بتوانیم هنگام اجرای برنامه مقادیری را وارد کنیم از عملگر ورودی **>>** استفاده می‌کنیم.

استفاده از دستور ورودی به شکل زیر است:

cin >> *variable*;

variable **نام یک متغیر است.**

این همان متوازی الاضلاع در فلوچارت است

Read (a)

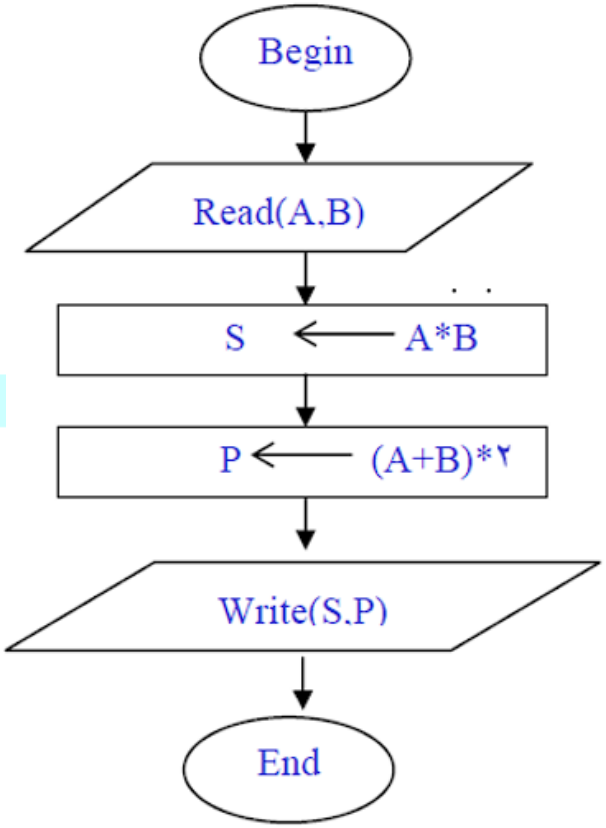
هنگامی که بخواهیم مقداری را از کاربر دریافت کنیم.

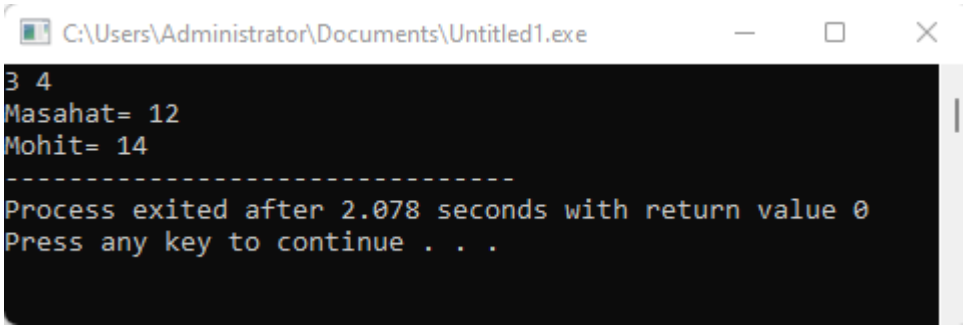
مثال

برنامه ای بنویسید که طول و عرض مستطیل را از ورودی دریافت و مساحت و محیط آنرا چاپ کند.

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int a,b,s,p;
6      cin>>a>>b;
7      s=a*b;
8      p=2*(a+b);
9      cout<<"Masahat= "<<s<<"\nMohit= "<<p;
10     return 0;
11 }
```

\n مثل Enter در تکست،
خط جدید تولید میکند
همچنین میتوانیم از endl استفاده کنیم

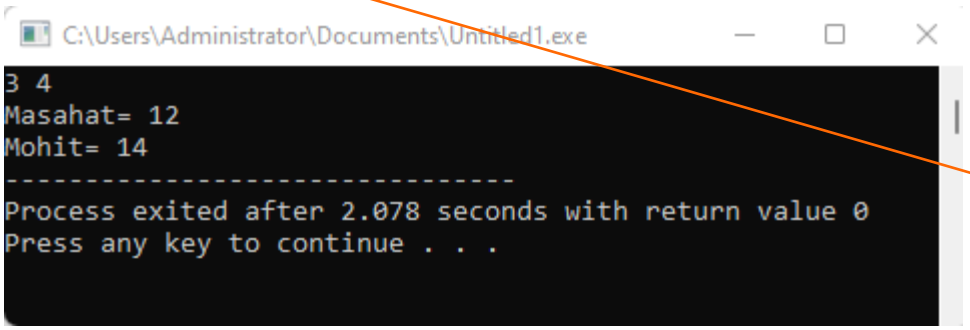


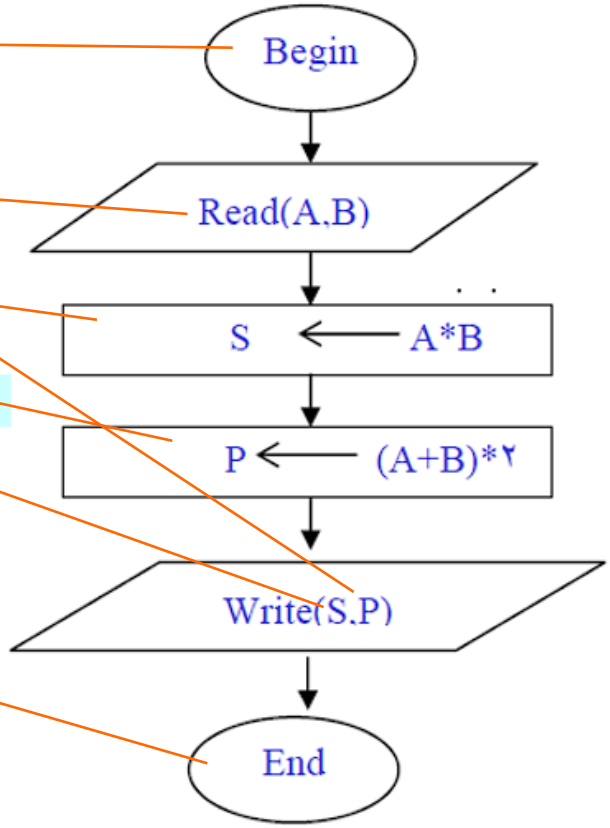


مثال

برنامه ای بنویسید که طول و عرض مستطیل را از ورودی دریافت و مساحت و محیط آنرا چاپ کند.

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int a,b,s,p;
6      cin>>a>>b;
7      s=a*b;
8      p=2*(a+b);
9      cout<<"Masahat= "<<s<<"\nMohit= "<<p;
10     return 0;
11 }
```





عملگر ورودی

عملگر ورودی نیز مانند عملگر خروجی به شکل جریانی رفتار می‌کند. یعنی همان طور که در عملگر خروجی می‌توانستیم چند عبارت را با استفاده از چند عملگر >> به صورت پشت سر هم چاپ کنیم، در عملگر ورودی نیز می‌توانیم با استفاده از چند عملگر << چند مقدار را به صورت پشت سر هم دریافت کنیم. مثلا با استفاده از دستور:

```
cin >> x >> y >> z;
```

سه مقدار x و y و z به ترتیب از ورودی دریافت می‌شوند. برای این کار باید بین هر ورودی یک فضای خالی (space) بگذارید و پس از تایپ کردن همهٔ ورودی‌ها، کلید enter را بفشارید. آخرین مثال جلسه، این موضوع را بهتر نشان می‌دهد.

مقدار دهی به متغیر ها

مقداردهی پیش فرض یا اولیه

```
int a = 1;
```

مقداردهی با عملگر تخصیص مقدار

```
int a, b=3;
a = 1;
a = 2 * b;
a = b
```

مقداردهی توسط کاربر

```
int a;
cin>>a;
```

انواع داده
Data Type

انواع داده

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.

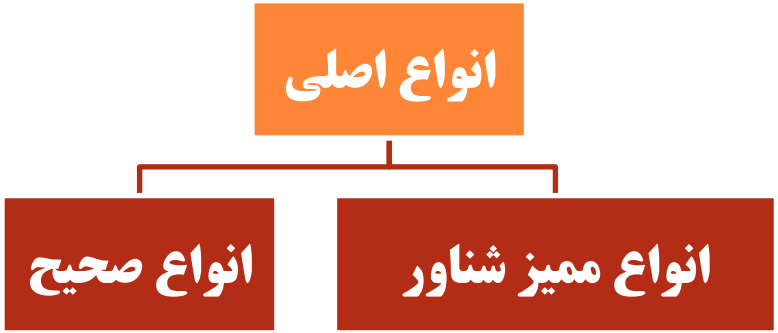
اما واقعاً چرا شناخت انواع داده برنامه‌نویسی مهم است ؟

اول از همه برای نگهداری داده‌ها نیاز است نوع آن‌ها را تشخیص دهیم.

در ثانی، برای پردازش مجبوریم عملیات‌هایی را روی این داده‌ها انجام دهیم. اینکه هر نوع داده چه عملگرهایی را قبول می‌کند بسیار مهم است.

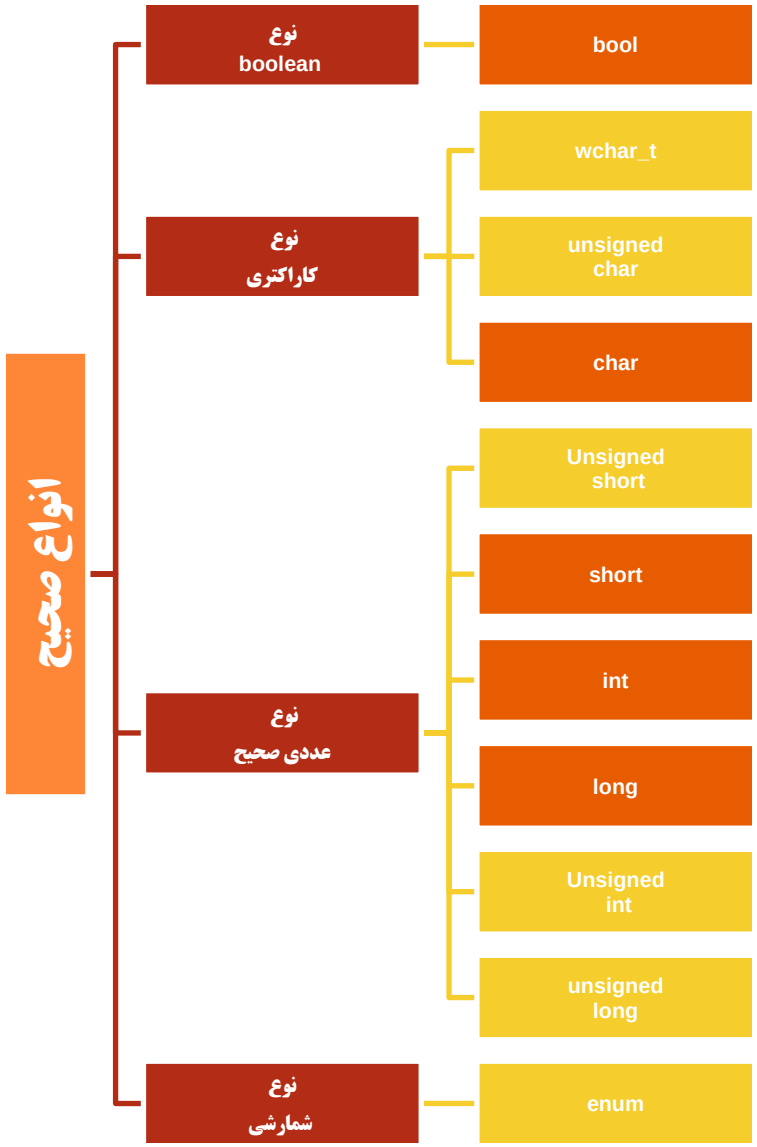
انواع داده عددی

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

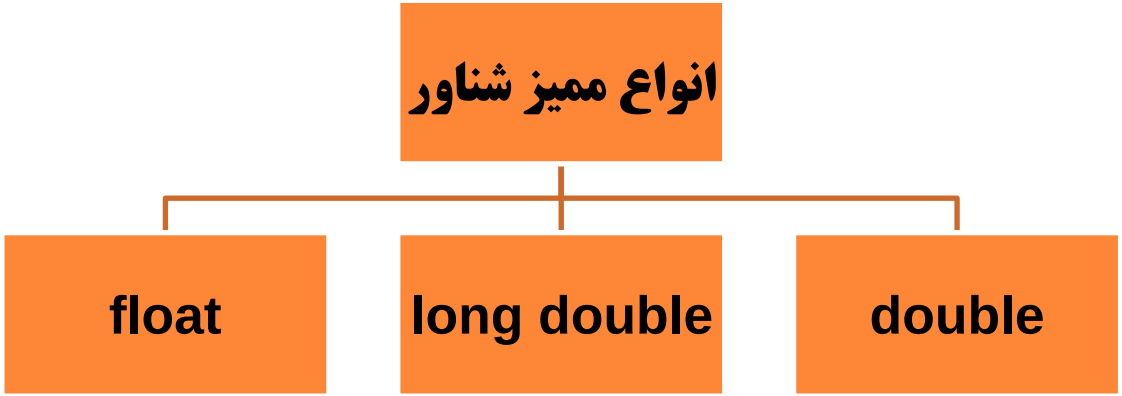


نوع صحیح برای نگهداری اعداد صحیح (اعداد ۰ و ۱ و ۲ و ...) استفاده می‌شود. این اعداد بیشتر برای شمارش به کار می‌روند و دامنه محدودی دارند.

انواع داده عددی صحیح



داده ممیز شناور



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 ۳۵۲/۱۸۷ را می‌توان به شکل ۳۵/۲۱۸۷×۱۰ یا ۳۵۲۱۸/۷×۱۰$^{-۲}$ یا ۳۵/۲۱۸۷×۱۰$^{-۱}$یا ۳/۵۲۱۸۷×۱۰^{۲} و یا ... نوشت .

به این ترتیب با کم و زیاد کردن توان عدد ۱۰ ممیز عدد نیز جابه‌جا می‌شود.

به همین دلیل است که به اعداد اعشاری «اعداد ممیز شناور» می‌گویند.

متغیر عدد صحیح

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

متغیر عدد صحیح

نوع متغییر	حداقل مقدار قابل پذیرش	حداکثر مقدار قابل پذیرش
short	-32768	32767
unsigned short	0	65535
int	-2147483648	2147483647
unsigned int	0	4294967295
long	-2147483648	2147483647
unsigned long	0	4294967295

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.

بدست آوردن تعداد بایت ها هر نوع داده

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      cout<<sizeof(int)<<endl;
6      cout<<sizeof(long int)<<endl;
7      cout<<sizeof(long long int)<<endl;
8      return 0;
9  }
```

خروجی:

- 4
- 4
- 8

اعداد بیش از ۱۰ رقم

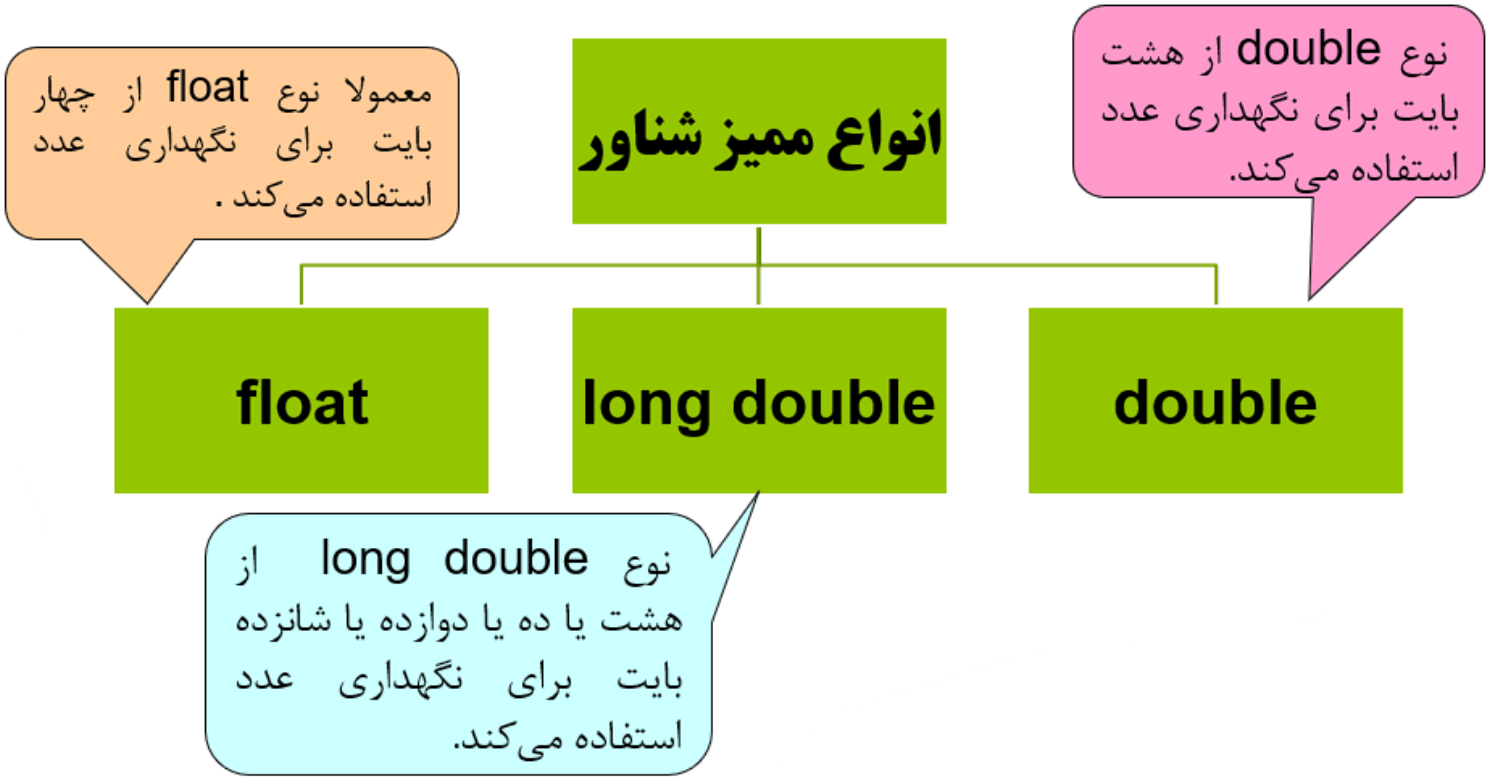
اگر IDE شما پشتیبانی می کند از نوع داده زیر استفاده می شود:

- long long int

- Min: -9,223,372,036,854,775,808
- max: +9,223,372,036,854,775,807

- Unsigned long long int
- 0 – 18,446,744,073,709,551,615

در C++ سه نوع ممیز شناور وجود دارد:



Float - double

- نوع داده‌ای به نام long float وجود ندارد ولی نوع داده double **دو برابر** float است.
- در مواردی که در برنامه‌نویسی از اعداد خیلی بزرگ یا خیلی کوچک استفاده می‌کنیم، برای داشتن دقت بیشتر اعداد اعشاری از نوع داده double استفاده می‌کنیم.
- نوع داده float تا ۶ رقم دقت و نوع داده Double تا ۱۵ رقم دقت را پشتیبانی می‌کند.
- به همین دلیل محاسبات double وقت‌گیرتر از محاسبات float است.

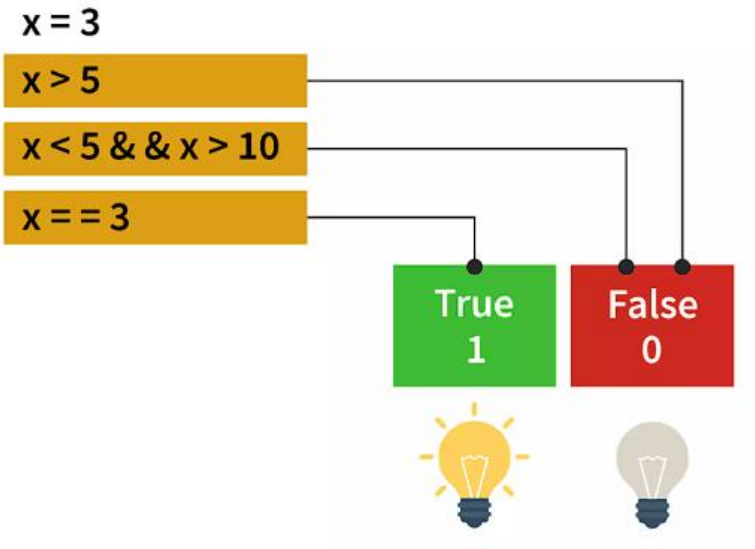
نوع داده بولین bool

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.

اما این مقادیر در اصل به صورت ۱ و ۰ درون رایانه ذخیره می‌شوند:

۱ برای true و ۰ برای false



نوع داده 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.

مثلا هر یک از حروف 'A' تا 'Z' و 'a' تا 'z' و هر یک از اعداد '۰' تا '۹' و یا نشانه‌های '~' تا '+' روی صفحه‌کلید را یک کاراکتر می‌نامند.

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:

```
char  c ='A';
```

مثال

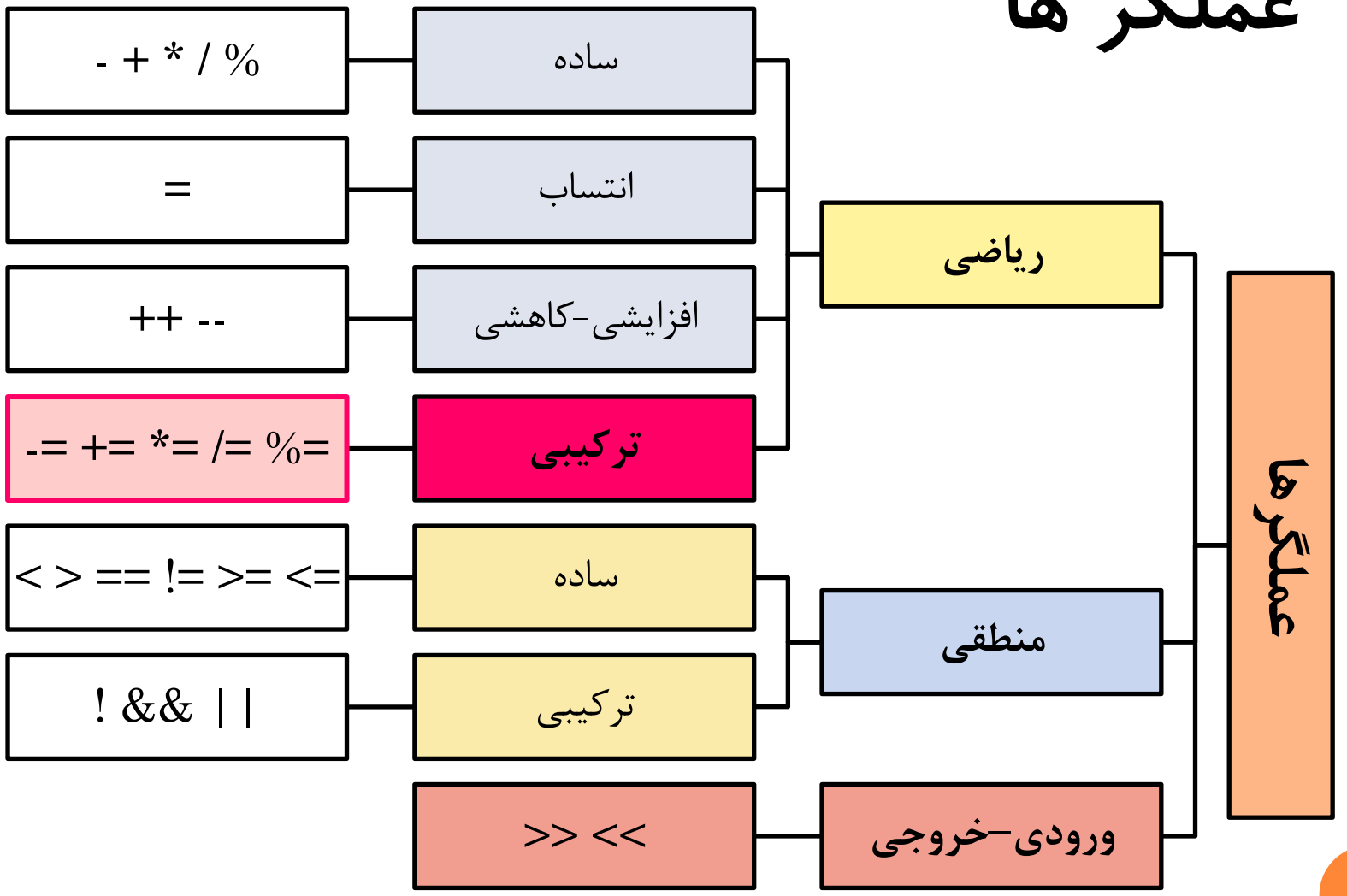
```
int a = 12;
char b='D';
Int D = 30;

cout<<a;
cout<<b;
cout<<D;
cout<<'w';
cout<<"w";
cout<<3;
```

خروجی :

```
12D30ww3
```

عملگرها

عملگر ها



C++ مانند اغلب زبان‌های برنامه‌نویسی برای محاسبات از عملگرهای جمع (+) ، تفریق (-) ، ضرب (*) ، تقسیم (/) و باقیمانده (%) استفاده می‌کند.

عملگرهای افزایشی و کاهشی

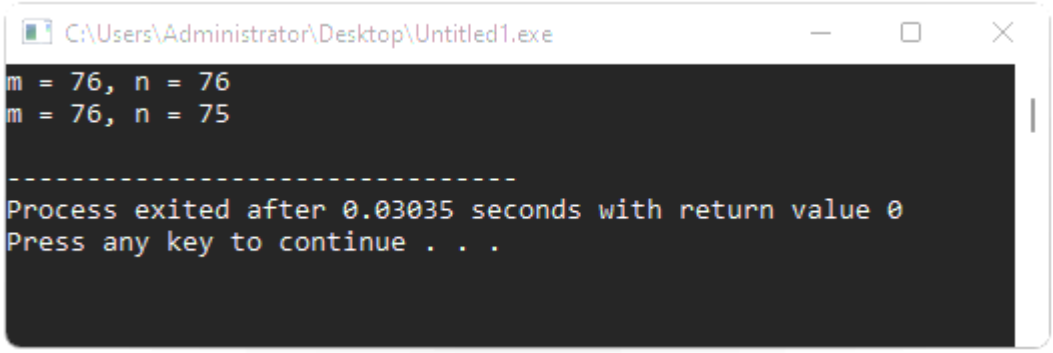
C++ برای دستکاری مقدار متغیرهای صحیح، دو عملگر جالب دیگر دارد:

عملگر ++ : مقدار یک متغیر را یک واحد افزایش می‌دهد.

عملگر -- : مقدار یک متغیر را یک واحد کاهش می‌دهد.

اما هر کدام از این عملگرها دو شکل متفاوت دارند: شکل «پیشوندی» و شکل «پسوندی».

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int m, n;
6      m = 75;
7      n = ++m;
8      cout << "m = " << m << ", n = " << n << endl;
9      m = 75;
10     n = m++;
11     cout << "m = " << m << ", n = " << n << endl;
12 }
```



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--

در شکل **پیشوندی** ابتدا متغیر، متناسب با عملگر، افزایش یا کاهش می‌یابد و پس از آن مقدار متغیر برای محاسبات دیگر استفاده می‌شود.

در شکل **پسوندی** ابتدا مقدار متغیر در محاسبات به کار می‌رود و پس از آن مقدار متغیر یک واحد افزایش یا کاهش می‌یابد.

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--

در شکل **پیشوندی** ابتدا متغیر، متناسب با عملگر، افزایش یا کاهش می‌یابد و پس از آن مقدار متغیر برای محاسبات دیگر استفاده می‌شود.

در شکل **پسوندی** ابتدا مقدار متغیر در محاسبات به کار می‌رود و پس از آن مقدار متغیر یک واحد افزایش یا کاهش می‌یابد.

عملگرهای مقدارگذاری مرکب

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.

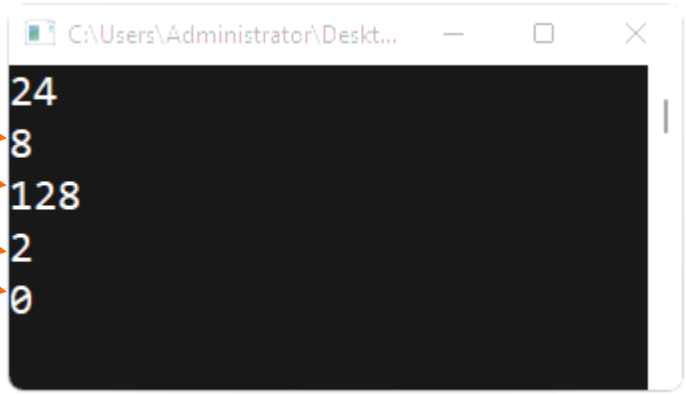
عملگرهای مقدارگذاری مرکب

قبلا از عملگر = برای مقدارگذاری در متغیرها استفاده کردیم.

C++ عملگرهای دیگری دارد که مقدارگذاری در متغیرها را تسهیل می‌نمایند.

عملگر مرکب در C++ عبارتند از: += و -= و *= و /= و %=

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int m=16,n=16,a=16,b=16,c=16;
6      m+=8;
7      n-=8;
8      a*=8;
9      b/=8;
10     c%=8;
11     cout << m << endl;
12     cout << n << endl;
13     cout << a << endl;
14     cout << b << endl;
15     cout << c << endl;
16 }
```



عملگرهای مرکب به شکل زیر عمل می کنند.

```
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;
```

نکته ای در مورد عملگر / (تقسیم) و نوع داده ها:

- به دستور زیر توجه کنید:

- float f = 122/11;

شما انتظار دارید عدد ۱۱.۰۹۰۹۰۹۰۹۰۹ با این دستور داشته باشید، ولی این دستور در واقع، عدد ۱۱ را به f نسبت می‌دهد.

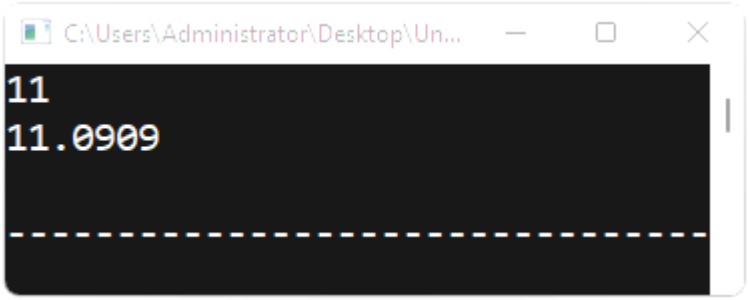
علت اینکار این است که عبارت سمت راست شامل دوعدد صحیح است، بنابراین این کد، از محاسبات عدد صحیح استفاده می‌کند و بخش کوچک را حذف کرده وعدد ۱۱ را به متغیر f نسبت می‌دهد.

برای نمایش عدد به صورت ۱۱.۰۹۰۹۰۹۰۹۰۹، باید دستور را به صورت زیر تغییردهیم :

- float f = 122.0/11

- با تغییراین دستوربه روش بالا می‌توان این کار را انجام داد.

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      float f = 122/11;
6      cout << f << endl;
7      float g = 122.0/11;
8      cout << g << endl;
9  }
```



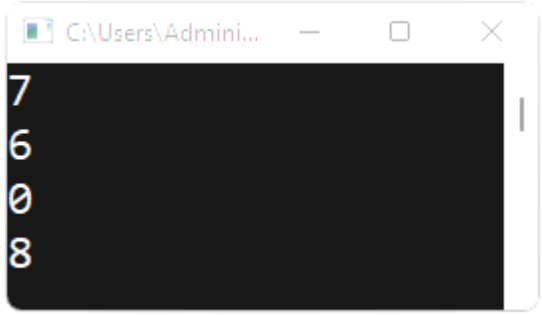
تقدم عملگر ها

اولویت	عملگر	کاربرد
۱	( )	پرانتز
۲	٪	درصد
۳	^	توان
۴	* و /	ضرب و تقسیم
۵	+ و -	جمع و تفریق

```
()
++      --      !
/       *       %
+       -
>       >=      <       <=
==              !=
&&
||
=       +=      -=      *=      /=      %=
```

مثال تقدم عملگر ها

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int number=3 ,number1,number2,number3,number4;
6      number1 = 3 + ++number;
7      cout<< number1 << endl;
8      number=3;
9      number2 = 3 + number++;
10     cout<< number2 << endl;
11     number3 = (1+2) * (3/4) % (5-(6*7));
12     cout<< number3 << endl;
13     number4 = 3*2+8/4;
14     cout<< number4 << endl;
15 }
```



ثابت ها و
تبدیل نوع(Cast)

ثابت ها

در بعضی از برنامه‌ها از متغیری استفاده می‌کنیم که فقط یک بار لازم است آن را مقداردهی کنیم و سپس مقدار آن متغیر در سراسر برنامه بدون تغییر باقی می‌ماند.

مثلا در یک برنامهٔ محاسبات ریاضی، متغیری به نام PI تعریف می‌کنیم و آن را با 3.14 مقداردهی می‌کنیم و می‌خواهیم که مقدار این متغیر در سراسر برنامه ثابت بماند.

در چنین حالاتی از «ثابت‌ها» استفاده می‌کنیم.

یک ثابت، یک نوع متغیر است که فقط یک بار مقداردهی می‌شود و سپس تغییر دادن مقدار آن در ادامهٔ برنامه ممکن نیست.

تعریف ثابت‌ها مانند تعریف متغیرهاست با این تفاوت که کلمه کلیدی **const** به ابتدای تعریف اضافه می‌شود.

متغییرهای **const** را نمی توان از طریق ورودی مقداردهی کرد. زمانی که تعریف می شوند باید مقدار آن ها مشخص شود.

مثال

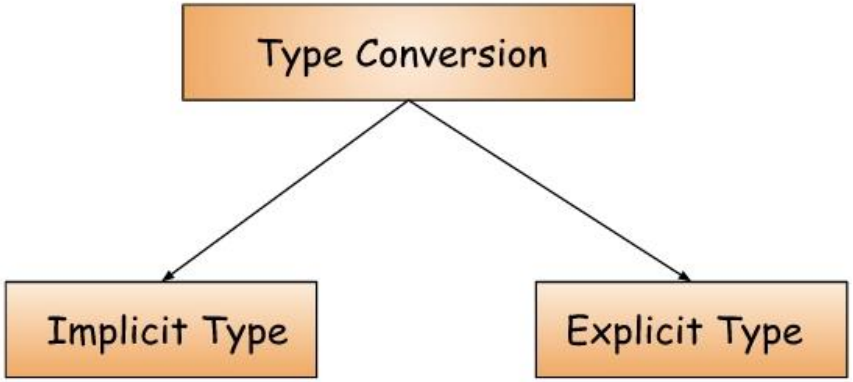
```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {  // defines constants; has no output:
5     const char BEEP ='\b';
6     const int MAXINT=2147483647;
7     const float DEGREE=23.53;
8     const double PI=3.14159265358979323846;
9     cout<<MAXINT<<endl;
10    MAXINT=MAXINT/2;
11    cout<<MAXINT<<endl;
12 return 0;
13 }
```

10 10 [Error] assignment of read-only variable 'MAXINT'

تبدیل نوع

دو نوع تبدیل cast وجود دارد:

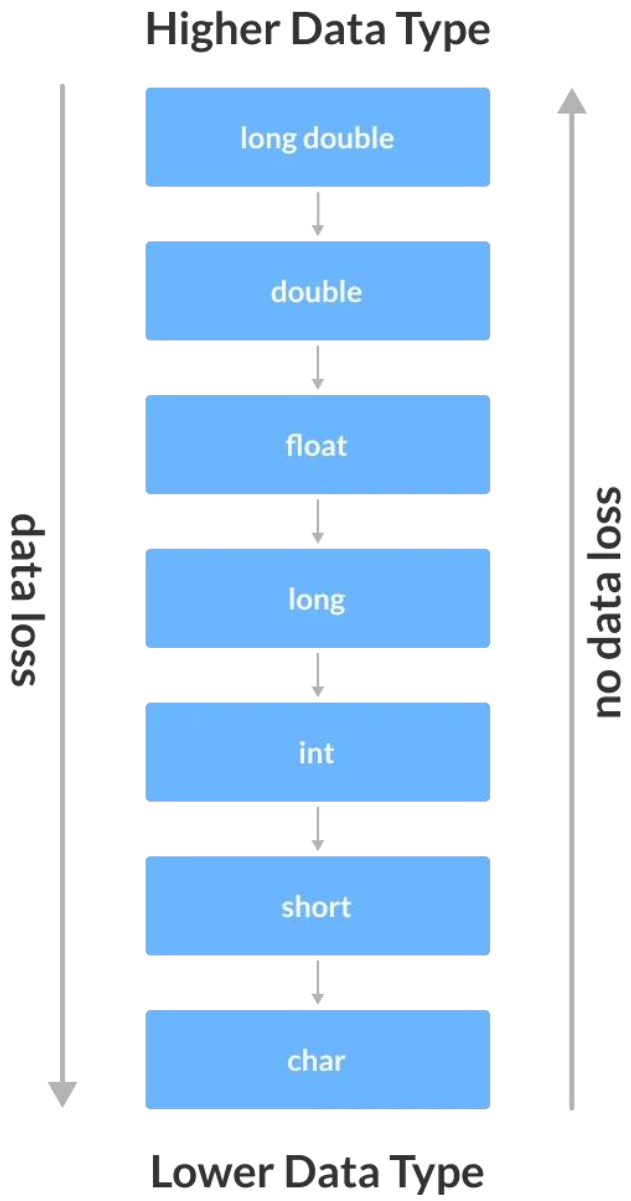
۱- (ضمنی)implicit و ۲- (صریح) explicit



اساس عمل تبدیل: هرجا که کامپایلر انتظار استفاده از نوعی را داشته باشد اما نوع دیگری استفاده شود، اگر امکان داشته باشد خودش تبدیل میکند implicit

وگرنه باید برنامه نویس خودش تبدیل کند explicit .

تبدیل نوع



تبدیل نوع

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 عمل **گسترش نوع** می‌گویند.

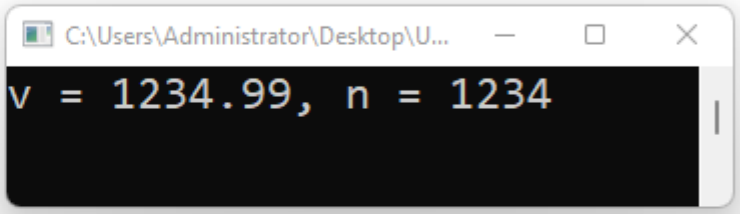
برای این که مقدار یک متغیر از نوع ممیز شناور را به نوع صحیح تبدیل کنیم از عبارت **int()** استفاده می‌کنیم به این عمل تبدیل نوع صریح گفته می شود

مثال‌های زیر تبدیل نوع و گسترش نوع را نشان می‌دهند.

مثال تبدیل نوع صریح:

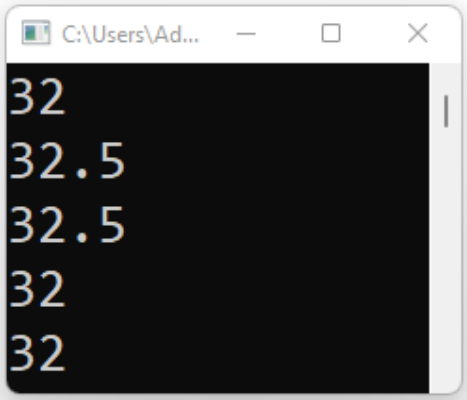
این برنامه، یک نوع double را به نوع int تبدیل می‌کند:

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      double v=1234.987;
6      int n;
7      n = int(v);
8      cout << "v = " << v << ", n = " << n << endl;
9      return 0;
10 }
```



مثال تبدیل نوع صریح:

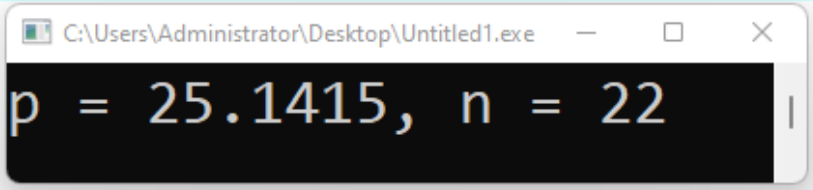
```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      cout<<  65 / 2<<endl;
6      cout<< (double) 65 / 2<<endl;
7      cout<< (float) 65 / 2<<endl;
8      cout<< (int) 32.6<<endl;
9      cout<< int(32.6)<<endl;
10     /*
11     (data type) value
12     (date type) variable name
13     Data type(value)
14     Data type(variable name)
15     */
16 }
```



تبدیل نوع ضمنی (گسترش نوع int به double)

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

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {  // adds an int value with a double value:
5     int n = 22;
6     double p = 3.1415;
7     p = p + n;
8     cout << "p = " << p << ", n = " << n << endl;
9     return 0;
10 }
```

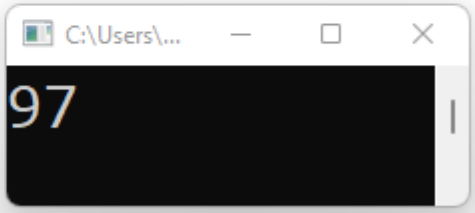


جدول کد ASCII حروف

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

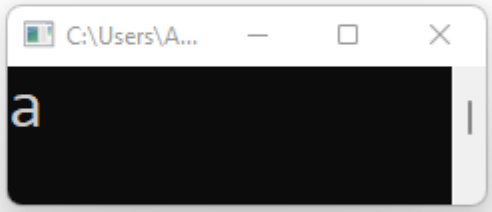
تبدیل نوع ضمنی (گسترش نوع char به int)

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {  // int-char
5      char c = 'a'; //'a'    97
6      int n = c;
7      cout<<n<<endl;
8
9      return 0;
10 }
```



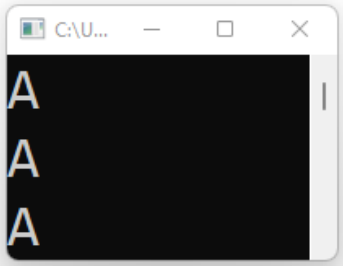
تبدیل نوع ضمنی (گسترش نوع int به char)

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {  // int-char
5      int n = 97;
6      char c = n;
7      cout<<c<<endl;
8      return 0;
9  }
```



تبدیل نوع صریح (گسترش نوع int به char)

```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {  // int-char
5     cout<<(char) 65<<endl;
6     cout<< char (65)<<endl;
7     cout<<static_cast<char> (65)<<endl;
8     return 0;
9 }
```



حوزه متغیرها

حوزه متغیرها

انتخاب نام‌های نامفهوم یا ناقص سبب کاهش خوانایی برنامه و افزایش خطاهای برنامه‌نویسی می‌شود.

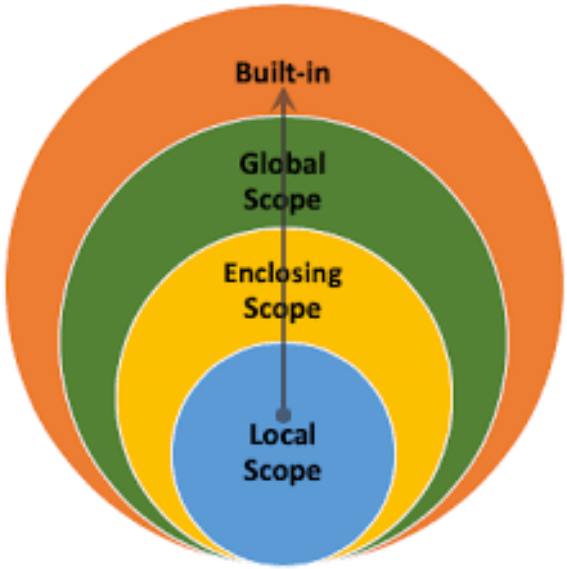
استفاده از متغیرها در حوزهٔ نامناسب هم سبب بروز خطاهایی می‌شود.

«**حوزه متغیر**» محدوده‌ای است که یک متغیر خاص اجازه دارد در آن محدوده به کار رود یا فراخوانی شود.

اصطلاح «بلوک» 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.

حوزه متغیرها

- 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.
- می‌توانیم در یک برنامه، چند متغیر متفاوت با یک نام داشته باشیم به شرطی که در حوزه‌های مشترک نباشند.



متغیرهای محلی (Local):

برای کار با متغیر های محلی در c++ در هر تابعی میتوان یکسری متغیر خاص همان تابع تعریف نمود که به آنها متغیرهای محلی یا local می گوییم.

متغیرهای محلی فقط در همان تابع که تعریف شده اند شناخته شده میباشند.

متغیر سراسری global در c++:

این نوع متغیرها در خارج از توابع هستند ودر کلیه توابعی که بعد از تعریف متغیر می آیند شناخته شده اند.

در مبحث توابع مفصل به این موضوع میپردازیم

معرفی برخی خطاها

«خطای زمان کامپایل»

این قبیل خطاها که اغلب خطاهای نحوی هستند، توسط کامپایلر کشف می‌شوند و به راحتی می‌توان آن‌ها را رفع نمود.

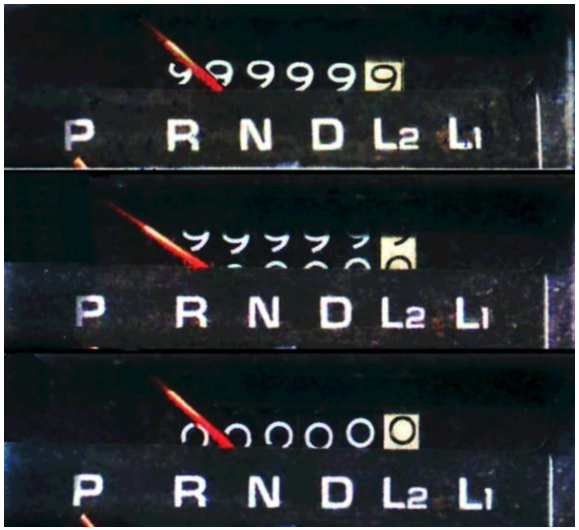
«خطای زمان اجرا»

کشف اینگونه خطاها به راحتی ممکن نیست و کامپایلر نیز چیزی راجع به آن نمی‌داند.

برخی از خطاهای زمان اجرا سبب می‌شوند که برنامه به طور کامل متوقف شود و از کار بیفتد.

خطای سرریزی

یک متغیر هر قدر هم که گنجایش داشته باشد، بالاخره مقداری هست که از گنجایش آن متغیر بیشتر باشد.

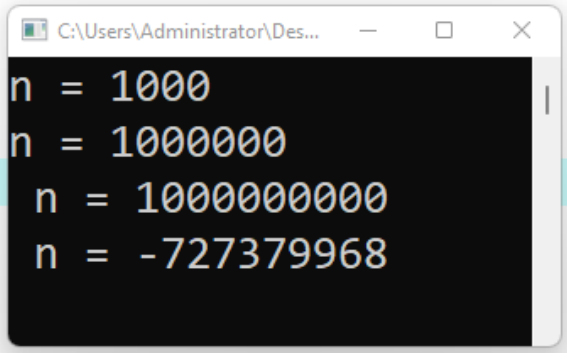


اگر سعی کنیم در یک متغیر مقداری قرار دهیم که از گنجایش آن متغیر فراتر باشد، متغیر «سرریز» می‌شود،در چنین حالتی می‌گوییم که خطای سرریزی **overflow** رخ داده است.

مثال: سرریزی عدد صحیح

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

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int n =1000;
7      cout << "n = " << n << endl;
8      n = n * 1000;       // multiplies n by 1000
9      cout << "n = " << n << endl;
10     n = n * 1000;       // multiplies n by 1000
11     cout << " n = " << n << endl;
12     n = n * 1000;       // multiplies n by 1000
13     cout << " n = " << n << endl;
14     return 0;
15 }
16
```



خطای گرد کردن (رندکردن)

خطای گرد کردن نوع دیگری از خطاست که اغلب وقتی رایانه‌ها روی اعداد حقیقی محاسبه می‌کنند، رخ می‌دهد.

برای مثال عدد ۱/۳ممکن است به صورت ۰.۳۳۳۳۳۳ ذخیره شود که دقیقا معادل ۱/۳ نیست.

این خطا از آن‌جا ناشی می‌شود که اعدادی مثل ۱/۳ مقدار دقیق ندارند و رایانه نمی‌تواند این مقدار را پیدا کند، پس نزدیک‌ترین عدد قابل محاسبه را به جای چنین اعدادی منظور می‌کند.

«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.

با تشکر از همراهی شما

محمد سعید صفایی صادق

www.SaeidSafaei.ir