به نام خدا

آموزش برنامه نویسی C++
مورد نیاز دوره ی مقدماتی Geant4

تهیه کننده: محمد تقی بطیار

کارگاه آموزشی مقدماتی Geant4 انجمن هسته ای با همکاری دانشگاه خواجه نصیر الدین طوسی
تابستان ۱۳۹۴

مقدمات برنامه‌نویسی با C++

- ✓ چرا C++ ؟
- ✓ تاریخچهٔ C++
- ✓ آماده‌سازی مقدمات
- ✓ شروع کار با C++
- ✓ عملگر خروجی
- ✓ لیترال‌ها و کاراکترها
- ✓ متغیرها و تعریف آن‌ها
- ✓ مقداردهی اولیه به متغیرها
- ✓ ثابت‌ها
- ✓ عملگر ورودی

مقدمه

- در دهه ۱۹۷۰ در آزمایشگاه‌های بل زبانی به نام C ایجاد شد. انحصار این زبان در اختیار شرکت بل بود تا این که در سال ۱۹۷۸ توسط Kernighan و Richie شرح کاملی از این زبان منتشر شد و به سرعت نظر برنامه‌نویسان حرفه‌ای را جلب نمود.

- هنگامی که بحث شی‌گرایی و مزایای آن در جهان نرم‌افزار رونق یافت، زبان C که قابلیت شی‌گرایی نداشت ناقص به نظر می‌رسید تا این که در اوایل دهۀ ۱۹۸۰ دوباره شرکت بل دست به کار شد و زبان C++ را طراحی نمود.

- C++ ترکیبی از دو زبان C و Simula بود و قابلیت‌های شی‌گرایی نیز داشت. از آن زمان به بعد شرکت‌های زیادی کامپایلرهایی برای C++ طراحی کردند. این امر سبب شد تفاوت‌هایی بین نسخه‌های مختلف این زبان به وجود بیاید و از قابلیت سازگاری و انتقال آن کاسته شود.

- به همین دلیل در سال ۱۹۹۸ زبان C++ توسط موسسهٔ استانداردهای ملی آمریکا (ANSI) به شکل استاندارد و یک‌پارچه درآمد.

چرا C++؟

- ✓ زبان C به طور گسترده ای در دسترس است و مفسر های تجاری آن در بیشتر کامپیوتر های شخصی قابل استفاده می باشند.

- ✓ در این زبان عملگرهایی تعبیه شده که برنامه‌نویسی سطح پایین و به زبان ماشین را نیز امکان‌پذیر می‌سازد.

- ✓ C زبانی است همه منظوره، ساخت یافته و انعطاف پذیر که ویژگی های زبان های سطح پایین که معمولا در زبان ماشین موجود است را دارا می باشد.

- زبان C برای اجرای بسیاری از دستوراتش از توابع کتابخانه‌ای استفاده می‌کند و بیشتر خصوصیات وابسته به سخت‌افزار را به این توابع واگذار می‌نماید.

- به طور کلی جامعیت، عمومیت، خوانایی، سادگی، کارایی و پیمانه ای بودن که از ویژگی های برنامه ی ایده ال هستند در C پیاده سازی می شوند.
- C++ که از نسل C است، تمام ویژگی‌های جذاب C را به ارث برده است .
- دلیل آشنایی ما با C++ ورود به دنیای GEANT4 می باشد.

آماده سازی مقدمات

- ✓ یک «برنامه» دستورالعمل‌های متوالی است که می‌تواند توسط یک رایانه اجرا شود. برای نوشتن و اجرای هر برنامه به یک «کامپایلر» احتیاج داریم.

- ✓ C++ بین حروف بزرگ و کوچک تمایز قائل می شود.

- ✓ انتهای هر خط ; می گذاریم.

اولین برنامه با C++

```
#include <iostream>
#include <conio.h>

int main()
{
    cout<< "salam" <<endl;
   getch();

}
```

- ✓پیش پردازنده برنامه ی جداگانه ای است که قبل از کامپایلر واقعی اجرا می گردد.
- ✓با شروع کامپایل برنامه پیش پردازنده به طور خودکار اجرا می شود.
- ✓تمام فرامین پیش پردازنده با علامت (#) شروع می گردند.
- ✓وظیفه ی پیش پردازنده اینست که فایل درخواستی را آماده و وارد برنامه کند.
- ✓پایان جمله های پیش پردازنده به (;) ختم نمی شود.

دو دستور متداول از پیش پردازنده عبارتند از:

#include
#define

فرمان #include

این فرمان موجب می گردد که کامپایلر همزمان با فایلی که ترجمه می کند یک متن را نیز از فایل دیگر بخواند.

تعریف این فرمان به شکل زیر است:

#include “filename”

در دستور فوق پیش پردازنده محل خاصی را که با عامل مشخص شده است نگاه می کند این محل جایی است که فایل های include سیستم نگهداری می شود.

فرمان #define

با استفاده از این فرمان می توان اسمی را با یک مقدار ثابت وابسته کرد.

مثال

```
#define  area  20
```

با استفاده از این فرمان مقدار area در حافظه برابر مقدار ۲۰ خواهد شد.

توضیحات

- ✓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.

- ✓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:

- ✓ با استفاده از دو علامت اسلش // : هر متنی که بعد از دو علامت اسلش بیاید تا پایان همان سطر یک توضیح تلقی می‌شود .

- ✓با استفاده از حالت C : هر متنی که با علامت /* شروع شود و با علامت */ پایان یابد یک توضیح تلقی می‌شود.

عملگر خروجی

- علامت >> عملگر خروجی در C++ نام دارد (به آن عملگر درج نیز می‌گویند).
- یک «عملگر» چیزی است که عملیاتی را روی یک یا چند شی انجام می‌دهد. عملگر خروجی، مقادیر موجود در سمت راستش را به خروجی سمت چپش می‌فرستد.
- به این ترتیب دستور

```
cout<< “salam” ;
```

- Salam را به خروجی cout می‌فرستد که معمولا به صفحه نمایش اشاره دارد. در نتیجه salam روی صفحه نمایش درج می‌شود.

لیترال ها و کاراکترها

✓ یک «لیترال» 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 دو نشانهٔ ' ' محصور شده باشد. پس 'w' و '!' و '1' هر کدام یک کاراکتر است.

✓ به تفاوت سه موجودیت «عدد» و «کاراکتر» و «لیترال رشته‌ای» دقت کنید: 6 یک عدد است، '6' یک کاراکتر است و "6" یک لیترال رشته‌ای است.

متغیر ها و تعریف آنها

«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.

نحوه ی اعلان یک 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 عبارت «مقداردهی اولیه» نام دارد. با استفاده از این عبارت می‌توان مقدار اولیه‌ای در متغیر مورد نظر قرار داد.

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 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.

مانند

```
const  int  n = 50;
```

مثال تعریف ثابت ها

```
int main()
{
   const  char  BEEP ='\b';
   const  int  MAXINT=2147483647;
   const  float  DEGREE=23.53;
   const  double  PI=3.14159265358979323846
   return 0;
}
```

۱- انواع دادهٔ عددی
۲- متغیر عدد صحیح
۳- محاسبات اعداد صحیح
۴- عملگرهای افزایشی و کاهشی
۵- عملگرهای مقدارگذاری مرکب
۶- انواع ممیز شناور
۷- تعریف متغیر ممیز شناور
۸- نوع بولین bool
۹- نوع کاراکتری char

انواع داده ی عددی

- ✓در C++ دو نوع اصلی داده وجود دارد: «نوع صحیح» و «نوع اعشاری». همهٔ انواع دیگر از روی این دو ساخته می‌شوند.
- ✓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.
- ✓نوع اعشاری برای نگهداری اعداد اعشاری استفاده می‌شود. اعداد اعشاری بیشتر برای اندازه‌گیری دقیق به کار می‌روند و دامنهٔ بزرگ‌تری دارند.

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.

نوع داده	حافظه (بایت)	محدوده
char	۱	۱۲۸- تا ۱۲۷
signed char	۱	۱۲۸- تا ۱۲۷
unsigned char	۱	۰ تا ۲۵۵
int	۲	برای سیستم ۱۶ بیتی بین ۳۲۷۶۸- تا ۳۲۷۶۷
unsigned int	۲	برای سیستم ۱۶ بیتی بین ۰ تا ۶۵۵۳۵
short int	۲	۳۲۷۶۸- تا ۳۲۷۶۷
unsigned short int	۲	۰ تا ۶۵۵۳۵
long int	۴	۲۱۴۷۴۸۳۶۴۸- تا ۲۱۴۷۴۸۳۶۴۷
unsigned long int	۴	۰ تا ۴۲۹۴۹۶۷۲۹۵
float	۴	اعداد اعشاری کوچک
double	۸	اعداد اعشاری بزرگ
long double	۱۰	اعداد اعشاری خیلی بزرگ

محاسبات عدد صحیح

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

علامت عملگر در زبان C++	نام عملگر
+	جمع
-	تفریق
*	ضرب
/	تقسیم
%	باقیمانده

عملگر های رابطه ای و منطقی

✓عملگر های رابطه ای عبارتند از:

== (مساوی یا برابر)، =! (مخالف)، =>(بزرگتر مساوی)، > (بزرگتر)، <(کوچکتر)، =<(کوچکتر مساوی)

✓ عملگر های منطقی عبارتند از:

عملگر && (معادل با و منطقی یا And)، عملگر || (معادل با یا منطقی یا OR)

عملگر ! (عملگر نقیض یا NOT)

مثال:

```
int a=0  if(!a)
if (a ==10 && b<15)
```

تقدم عملگر های منطقی و محاسباتی

- ترتیب عملگر های منطقی بدین صورت است که عملگر نقیض یا (!) بیشترین تقدم و عملگر های (&&) و (||) در مرتبه های بعدی تقدم قرار دارند.
- ترتیب تقدم عملگرهای محاسباتی: (++،--) (*،/،٪) (+،-)

در حاتی که مثلا ضرب و تقسیم و مد ارزش یکسانی دارند هریک زودتر آمده باشد در ابتدا اجرا می شود.

مثال:

```
int m, x=5, y=2, z=4;
m=x + 2*y/z;
m=6;
```

تقدم کلی عملگر ها

بالاترین تقدم
( )
! ~ ++ -- *(محتوی) &(آدرس)
* (ضرب) / %
+ -
<< >>
< <= > >=
== !=
&
^
|
&&
||
?
= += -= *= /= %=

عملگر (؟)

این عملگر با تست یک شرط، عبارتی را به یک متغیر نسبت می دهد

متغیر = exp1 ? exp2 = exp3;

exp1? مجموعه دستورات۱ : مجموعه دستورات۲ ;

اگر حاصل شرط exp1 صحیح باشد مجموعه دستورات ۱ و در غیر این صورت مجموعه دستورات ۲ اجرا می شود.

مثال:

```
int a = 6, b = 16, c = 0;
c = (a<b) ?  8:6;
```

c=8; خروجی

عملگرهای افزایشی و کاهشی

- **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».

- در شکل پسوندی، عملگر بعد از نام متغیر می‌آید مثل i++ یا i-- .
  در شکل پیشوندی، عملگر قبل از نام متغیر می‌آید مثل ++i یا --i .

- 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.

مثال

```
a = 10;
C = a++;
C = 10;
a = 11;
//***************
C = ++a;
a = 11;
C = 11;
```

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.

علامت عملگر	مثال	شکل دیگر عملگر
+=	a+=5	a=a+5
-=	a-=5	a=a-5
*=	a*=5	a=a*5
/=	a*=5	a=a/5
%=	a%=5	a=a%5

نوع 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.

ساختار های تکرار و تصمیم گیری

- ✓ حلقه ی for
- ✓ حلقه ی while
- ✓ ساختار تکرار do…while
- ✓ ساختار تصمیم گیری if
- ✓ ساختار تصمیم گیری else if
- ✓ ساختار تصمیم گیری switch

حلقه ی for

✓ساختار این حلقه ی تکرار به صورت زیر است:

```
for( مقدار اولیه ;شرط حلقه  ;گام حرکت شمارنده)
{
    دستور یکم;
        .
        .
        ام nدستور
}
```

حلقه ی while

ساختار حلقه ی while به صورت زیر است:

```
While (شرط حلقه)
  {
        دستور اول;
            .
            .
          دستور nام
    }
```

ساختار تکرار do…while

این ساختار دقیقا مانند ساختار تکرار while می باشد با این تفاوت که شرط حلقه در پایان حلقه می آید.

```
Do {
        دستور اول;
            .
            .
        دستور nام ;
}while(شرط حلقه)
```

ساختار تصمیم گیری if

ساختار تصمیم گیری if دارای دو شکل کلی است. در حالت اول اگر شرط ساختار if دارای ارزش درستی باشد دستورات داخل بلاک اجرا می شود و در صورت نادرست بودن شرط مذکور کنترل اجرای برنامه به بعد از علامت } می رود.

```
if(شرط)
{
        دستور یکم;
            .
        دستور n ام
 }
```

ساختار تصمیم گیری else if

```
if(شرط)
{
        دستور یکم;
            .
                دستور n ام
    }
else {      دستور یکم;
                .
                دستور n ام ;
        }
```

ساختار تصمیم switch

- ✓این ساختار برای تصمیم گیری های چندگانه بر اساس مقادیر مختلف یک عبارت استفاده می شود.
- ✓در تمام تصمیم گیری هایی که بیش از ۳ انتخاب وجود دارد از این ساختار استفاده می شود.
- ✓چند ساختار switch می توانند به صورت تو در تو بکار روند و هر case می تواند دارای ساختار switch باشد.
- ✓در ساختار if می توان عبارت منطقی یا رابطه ای را مورد بررسی قرار داد ولی در ساختار switch فقط مساوی بودن مقادیر بررسی می شود.
- ✓اگر در یک case از دستور break استفاده نشود با مقادیر case بعدی or می شود.

```
Switch(عبارت) {
    case <مقدار۱> :
        <دستورات>
      break;

      case <مقدار ۲>:
          <دستورات>
        break;

          …
          default:
              <دستورات n >
```

توابع و کلاس های حافظه

تابع

شکل کلی برنامه ها در C++ به همراه توابع به صورت زیر است:

```
#include <iostream>
الگوی تابع
int main()
{
دستورات برنامه;
}
تعریف توابع
```

مزایای استفاده از توابع

- ✓هر تابع برای حل بخشی از مساله بکار می رود و تعداد توابع به برنامه نویس و نوع مساله بستگی دارد.
- ✓با استفاده از توابع برنامه های پیچیده به بخش های کوچکتری تقسیم شده و هر بخش توسط تابعی نوشته می شود.
- ✓اشکال زدایی برنامه های حاوی تابع ساده تر است اگر در برنامه اشکالی وجود داشته باشد بررسی تابعی که این اشکال را دارد ساده تر است.
- ✓با تعریف توابع همکاری بین افراد فراهم می شود و هر کس می تواند بخشی از برنامه را بنویسد.
- ✓برنامه نویسی با استفاده از توابع موجب صرفه جویی در وقت می شود زیرا می توان تابعی را که عملی خاص انجام می دهد در برنامه ی دیگری که به آن نیاز دارد مورد استفاده قرار داد.

نوشتن تابع

ساختار و اجزای توابع در C++ به صورت زیر است:

```
<نوع تابع>  نام تابع  (لیست پارامتر ها)
{
مجموعه دستورات;
}
```

نوشتن تابع

- ✓نام تابع یکی از انواع توابع تعریف شده در C++ یا انواع دیگری است که کاربر تعریف می کند.
- ✓اگر تابعی بخواهد مقداری را به تابع فراخواننده برگرداند آن مقدار در نام تابع قرار می گیرد.
- ✓چون هر مقداری دارای نوع است پس نام تابع نیز باید دارای نوع باشد.
- ✓پارامتر ها اطلاعاتی هستند که هنگام فراخوانی تابع، از تابع فراخواننده به آن ارسال می شوند.
- ✓پارامتر ها وسیله ای برای تبادل اطلاعات بین تابع فراخواننده و فراخوانی شونده هستند.
- ✓در لیست پارامتر ها نوع پارامتر ها نیز باید مشخص گردد.

نوشتن تابع

❖هر تابع دارای سه جنبه است:

1) جنبه ی تعریف
2) جنبه ی فراخوانی
3) جنبه ی اعلان

- ✓ اعلان تابع را الگوی تابع می گویند و باید خارج از تابع main به کامپایلر اعلان گردد و مشخص می کند تابع چگونه فراخوانی شود.
- ✓ به بخشی از الگوی تابع که شامل نام تابع و لیست پارامتر هاست امضای تابع گویند.
- ✓ فراخوانی تابع داخل بدنه ی main و با استفاده از نام آن انجام می گیرد.
- ✓ آرگومان تابع اطلاعاتی است که هنگام فراخوانی تابع جلوی نام آن قرار می گیرد.

شیوه ی بکارگیری تابع در برنامه

```
#include  <iostream>
void function (int x, int y);
int main() {
 int a,b;
…
function (a,b);
…
getch();
}
void function (int x, int y)
{
     cout << " x = " << x << "   " << "y = " << y;
…
}
```

الگوی تابع

ارگومان های تابع

پارامتر های تابع

عنوان تابع

بدنه ی تابع

نحوه ی کار تابع

```
void function1 ();
void function2 ();
void function3();
int main() {
 …
function 1();
…
function 2 ();
…
function3 ();
…
return 0;
}
```

روش های ارسال پارامتر ها به توابع

به دو طریق می توان پارامتر ها را از تابع فراخواننده به تابع فراخوانی شونده ارسال کرد:

۱. فراخوانی با مقدار

- ✓ توابعی که هیچ مقداری را به تابع فراخواننده بر نمیگردانند.
- ✓ توابعی که فقط یک مقدار را به تابع فراخواننده بر می گردانند.

۲. فراخوانی با ارجاع

- ✓ در این روش توابع می توانند چندین مقدار را به تابع فراخواننده باز گردانند.
- ❖ در روش فراخوانی با مقدار هنگام فراخوانی مقادیر آرگومان ها در پارامتر ها کپی می شوند.
- ❖ در روش فراخوانی با ارجاع آدرس آرگومان ها به پارامتر ها منتقل می شود.

فراخوانی با مقدار

❖توابعی که هیچ مقداری را باز نمیگردانند:

✓ممکن است در برنامه از تابعی استفاده کنیم که آن تابع پس از فراخوانی، عملیاتی را انجام داده و خروجی مورد نظر را تولید کرده و چاپ کند و هیچ مقداری را به تابع فراخواننده تحویل ندهد.

❖توابعی که یک مقدار را بر می گردانند:

✓برای نوشتن این گونه توابع باید نوع آنها را در الگوی تابع و عنوان تابع مشخص کرد.

✓برای برگرداندن مقداری توسط تابع از دستور return استفاده می کنیم.

return <عبارت>;

کلاس های حافظه و حوزه ی متغیر ها

❖کلاس حافظه ویژگی ای از متغیر است که دو چیز را مشخص می کند:

1. حوزه ی متغیر
2. طول عمر متغیر

✓ حوزه ی متغیر مکان هایی از برنامه که متغیر قابل دستیابی است را مشخص می کند.

✓ منظور از طول عمر متغیر مدت زمانی است که متغیر در حافظه وجود دارد.

❖ انواع کلاس های حافظه در C++ عبارتند از:

1. کلاس حافظه ی اتوماتیک (automatic)
2. کلاس حافظه ی ثبات (register)
3. کلاس حافظه ی (static)
4. کلاس حافظه ی خارجی (extern)

<کلاس حافظه> <نوع متغیر> نام متغیر;

```
static  int  y;
```

کلاس حافظه ی اتوماتیک

❖کلاس حافظه ی اتوماتیک

✓برای تعیین این کلاس از کلمه ی کلیدی auto استفاده می شود:

```
auto double x;
```

✓ این نوع متغیر ها فقط در همان تابعی که تعریف می شوند قابل استفاده هستند.

✓با فراخوانی تابع به آنها حافظه تخصیص داده می شود و با خاتمه ی اجرای تابع از بین می روند.

کلاس حافظه ی استاتیک

❖متغیر استاتیک به دو دسته تقسیم می شود:

۱. متغیر استاتیک محلی:

- ✓ که فقط در همان تابعی که تعریف می شوند قابل استفاده هستند.
- ✓ هنگام فراخوانی تابع ایجاد می شوند و هنگام خروج از تابع، آخرین مقدارشان را حفظ می کنند.
- ✓ فقط یکبار مقدار اولیه می گیرند.

۲. متغیر استاتیک سراسری:

- ✓ این متغیر ها در خارج از توابع تعریف می شوند و در توابعی که پس از آنها تعریف می شوند قابل استفاده هستند.
- ✓ در صورتی که برنامه پیچیده بوده و کل برنامه در چند فایل موجود باشد متغیر استاتیک سراسری در همه ی فایل ها تعریف نمی شود و باید نوع متغیر را به صورت extern به کامپایلر اعلان کرد.

کلاس حافظه ی خارجی

- ❖ متغیر هایی که در خارج از توابع تعریف می شوند دارای کلاس حافظه ی خارجی هستند
- ✓ این نوع متغیر ها با شروع اجرای برنامه ایجاد می شوند و تا پایان اجرای برنامه حضور دارند و در سرتاسر برنامه قابل استفاده هستند:

```
extern  int  x,y;
```

کلاس حافظه ی ثبات

- ✓ ثبات ها حافظه هایی در داخل پردازنده هستند که کامپیوتر برای انجام محاسبات بر روی متغیر ها، آنها را از حافظه ی RAM به ثبات ها ارسال می کند و پس از انجام محاسبات به حافظه ی RAM بر می گرداند.
- ✓ استفاده از این نوع متغیر سرعت اجرای برنامه را افزایش می دهد.
- ✓ تعداد ثبات های پردازنده محدود بوده و فقط برای متغیر های مهمی که نیاز به انجام سریع محاسبات دارند با این کلاس حافظه اعلان می شوند.
- ✓ این کلاس فقط برای متغیر های محلی قابل استفاده است.
- ✓ آدرس متغیر هایی با کلاس حافظه ی ثبات، معنی ندارد.
- ✓ این متغیر ها با کلمه ی کلیدی register مشخص می شوند:

```
register float x;
```

توابع کتابخانه ای ریاضی

- ✓ توابع ریاضی در فایل سرآیند <cmath> قرار دارند و عبارتند از:

fabs(); acos(); asin(); exp(); pow(); sqrt()

کلاس ها و اشیاء

نوع داده ی انتزاعی

- ✓زبان برنامه نویسی C++ شیء گرا می باشد. در این نوع برنامه ها امکان تعریف نوع جدید وجود دارد.
- ✓برای تعریف نوع جدید از کلاس ها استفاده می شود.
- ✓نوع جدیدی که به این صورت تعریف می شود، نوع داده ی انتزاعی نام دارد.

- ❖ویژگی های نوع انتزاعی عبارتند از:
- ✓شامل مجموعه ای از مقادیر می باشند.
- ✓شامل مجموعه ای از عملیات ها می باشند که بر روی مقادیر انجام می شود.
- ✓ویژگی بسته بندی که باعث می شود عملیات فقط روی مجموعه مقادیر خاصی صورت گیرد.

کلاس ها و اشیاء

- ✓کلاس نوع جدیدی است که برنامه نویس آن را برای حل مسائل دنیای واقعی ایجاد می کند که حاوی داده ها و تعریف عملیات هاست.
- ✓داده های عضو کلاس را فیلد یا صفت می گویند.
- ✓عملیات های کلاس را تابع عضو یا متد می گویند. متد ها بر روی صفات اجرا می شوند.
- ❖شیء یا object، نمونه ای از کلاس است که در تکنیک برنامه نویسی شیء گرا موجودیت زمان اجرا می باشد.
- ✓کلاس گروهی از اشیاء است که ویژگی های مشترکی دارند و رفتارهای یکسانی از خود نشان می دهند و شیء نمونه ی خاصی از کلاس است.
- ✓از هر کلاس می توان چندین شیء تولید کرد و داده های هر شیء فقط از طریق متد های آن شیء قابل دستیابی می باشد.
- ✓اشیاء از طریق متد های خود می توانند باهم در ارتباط باشند.

برنامه نویسی شیء گرا

ویژگی های برنامه نویسی شی گرا عبارتند از:

- ✓ بسته بندی (encapsuation)
- ✓ وراثت (inheritance)
- ✓ انتزاع داده ها (data abstraction)
- ✓ چند ریختی(polymorphism)

بسته بندی و وراثت

❖بسته بندی

✓بسته بندی به معنای این است که تمام جنبه های یک موجودیت در داخل کلاس جمع آوری و از سایر موجودیت ها تفکیک شود.

✓بسته بندی مانع از این می شود که داده های یک شیء توسط متد های شیء دیگر قابل دستیابی باشد

❖وراثت

✓این ویژگی شبیه وراثت بیولوژیکی است که در آن فرزندان صفاتی را از والدین به ارِ می برند.

✓در برنامه نویسی شیء گرا می توان از کلاس موجود (کلاس پایه)، کلاس جدید(کلاس فرزند) را ایجاد کرد به طوری که کلاس جدید ، داده ها و متد های کلاس پایه را به ارِث می برد.

انتزاع داده ها و چند ریختی

❖انتزاع داده ها

✓با انتزاع داده ها فقط ویژگی های اساسی انواع، بدون ارائه ی اطلاعات جزئی قابل نمایش است. کلاس ها از مفهوم نوع داده ی انتزاعی پیروی می کنند.

❖چند ریختی

✓چند ریختی به این مفهوم است که موجودیتی مثل تابع، شیء، متغیر معانی یا کاربرد های مختلفی داشته باشند.

امتیازات برنامه نویسی شیء گرا

امتیازات برنامه نویسی شیء گرا عبارتند از:

- ✓**سهولت**: اشیاء نرم افزاری اشیاء دنیای واقعی را مدل سازی می کنند و موجب کاهش پیچیدگی می شوند.
- ✓**قطعه بندی**: هر شیء موجودیت مستقلی است که عملکرد آن در داخل شیء گنجانده می شود.
- ✓**تغییر پذیری**: در این نوع برنامه نویسی به راحتی می توان تغییراتی در نمایش داده ها و متد ها ایجادکرد.
- ✓**قابلیت نگهداری**: اشیاء می توانند به طور جداگانه نگهداری شوند.
- ✓**قابلیت استفاده ی مجدد**: اشیاء می توانند در برنامه های مختلفی مورد استفاده قرار گیرند

تعریف کلاس و اشیاء در C++

- ✓در اصطلاحات C++ داده های کلاس را عضو داده ای و متد های کلاس را تابع عضو می نامند.
- ✓برای نوشتن توابع عضو کلاس، باید نام کلاس را به همراه عملگر تعیین کننده ی حوزه (::) قبل از نام تابع عضو ذکر کنیم
- ✓تعریف کلاس به صورت زیر است:

```
class  classname
{
داده ها و توابع اختصاصی
Public:
داده ها و توابع عمومی
Private:
داده ها و توابع اختصاصی
Protected:
داده های محافظت شده
};
```

detector کلاس

```
class   detector
{
        int  numberoflayers;
public:
void  detectorname(string  s);
string  detectormaterial();
private:
double  detectorsize;
}
string detector :: detectorname()
{
 ….
}
```

تعریف شیء

- ✓قدم اول در مدل سازی اشیاء دنیای واقعی تعریف کلاس است.
- ✓با تعریف کلاس در C++ اشیاء واقعی ایجاد نمی شوند.
- ✓کلاس توصیف انتزاعی از یک شیء است.
- ✓دومین قدم برای مدل سازی اشیاء دنیای واقعی، نمونه سازی از یک کلاس است تا شیء ای از آن کلاس بوجود آید.
- ✓برای اعلان شیء از کلاس به صورت زیر عمل می کنیم:

```
className   objectName;
detector   det1,det2,det3;
```

که اشیاء det1, det2,det3 از نوع کلاس detector اعلان می شوند.

دستیابی به اعضای شیء

- ✓پس از ایجاد شیء می توان به اعضای آن دسترسی داشت.
- ✓برای دسترسی به اعضای شیء به صورت زیر عمل می کنیم:

objectName.member

det1. detectorname(s);

det1. detectormaterial();

det1. detectorsize;

- ❖محدودیت های اعضای کلاس عبارتند از:
- ✓عضوی از کلاس که کلاس حافظه ی آن static نباشد نمی تواند مقدار اولیه بگیرد.
- ✓هیچ عضوی از کلاس نمی تواند شیء از از همان کلاس باشد مگر اینکه یک اشاره گر باشد.
- ✓هیچ عضوی نمی تواند با کلاس حافظه ی auto، extern یا register تعریف شود.

بسته بندی و کنترل دستیابی

- ✓ بسته بندی یا پنهان سازی اطلاعات مفهوم مهمی در کار کردن با اشیاء می باشد.
- ✓ مجموعه مقادیری که صفات یک شیء در زمان خاص دارد، حالت شیء نامیده می شود. هر وقت متدی بر روی شیء اجرا شود حالت آن تغییر می کند.
- ✓ برای اینکه مفهوم بسته بندی رعایت شود هیچ متدی غیر از متد های خود کلاس نباید به صفات اشیاء دستیابی داشته باشد.
- ✓ کلاس ممکن است شیوه ی ذخیره ی داده هایش را به طور کلی عوض کند، ولی تا زمانی که از همان مجموعه از متد هایش برای دستکاری داده هایش استفاده می کند برای اشیاء دیگر مشکلی پیش نمی آید.
- ✓ اگر بخواهیم مقادیر متغیر نمونه را بازیابی کنیم و یا تغییر دهیم از متد های بازیابی و تغییر دهنده استفاده می کنیم.

متد های بازیابی

- برای دستیابی به فیلد اختصاصی یک شیء در خارج از آن، باید متدی بنویسیم که مقدار آن فیلد را در اختیار ما قرار دهد. این متد را متد بازیابی می نامیم. مثلا برای نوشتن متدی برای بازیابی صفت detectorsize از متدی به نام getdetectorsize استفاده می کنیم. این متد را می توان به صورت زیر نوشت.

```
int  getdetectorsize()
{
    return detectorsize;
}
```

متد های تغییر دهنده

- ✓ اگر بخواهیم مقدار یک صفت اختصاصی شیء ای را تغییر دهیم، از متد های تغییر دهنده استفاده می کنیم.
- ✓ این متد معمولا دارای پارامتری است که مقدار جدید صفت اختصاصی را تعیین می کند.
- ✓ مثلا برای تغییر متد detectorsize می توان به صورت زیر عمل کرد:

```
void setdetectorsize(int det_x)
{
    detectorsize = det_x;
}
```

تفکیک واسط کلاس از پیاده سازی آن

- ✓ اعلان توابع عضو کلاس و فیلد های آن، به نام واسط یا intrface کلاس خوانده می شود و بخش تعریف توابع عضو کلاس، به نام پیاده سازی یا implementetion کلاس خوانده می شود.
- ✓ واسط کلاس مشخص می کند که کلاس چه سرویس هایی می تواند ارائه کند ولی چگونگی ارائه ی این سرویس ها را مشخص نمی کند.
- ✓ در Geant4 بخش واسط کلاس در یک فایل با پسوند hh. و بخش پیاده سازی کلاس در فایل دیگری با پسوند cc. قرار دارد.

کلاس ها، اشیاء و وراثت

اعضای کلاسی با ویژگی استاتیک

- ✓وقتی یک شیء از یک کلاس ایجاد می شود، این شیء یک کپی از تمام اعضای داده ای آن کلاس را داراست.
- ✓در صورتی که بخواهیم فقط یک کپی از یک عضو داده ای یا متغیر بین تمام اشیاء کلاس مشترک باشد باید آن متغیر را داخل کلاس و در بخش public با واژه ی static مشخص نمایید.
- ✓عضو داده ای استاتیک می تواند داخل کلاس مقدار اولیه بگیرد.
- ✓وقتی شیء ای از یک کلاس ساخته می شود هر یک از توابع عضو آن را می توان فراخوانی کرد:

```
className  objectName;
objectName.show();
```

- ✓اگر تابع عضو به صورت استاتیک اعلان شود بدون نمونه سازی شیء ای از آن کلاس و مستقیما از طریق نام کلاس قابل فراخوانی است:

```
className::show();
```

اشاره گر هایی به اشیاء

- ✓می توان اشاره گر هایی از اشیاء را تعریف کرد برای دستیابی به اعضای اشیاء از طریق اشاره گر به جای نقطه از علامت <- استفاده می شود.

```
class  G4Element;
G4Element*  mat = new G4Element;
mat  -> AddIsotopes(”O“ , 90.*percent);
```

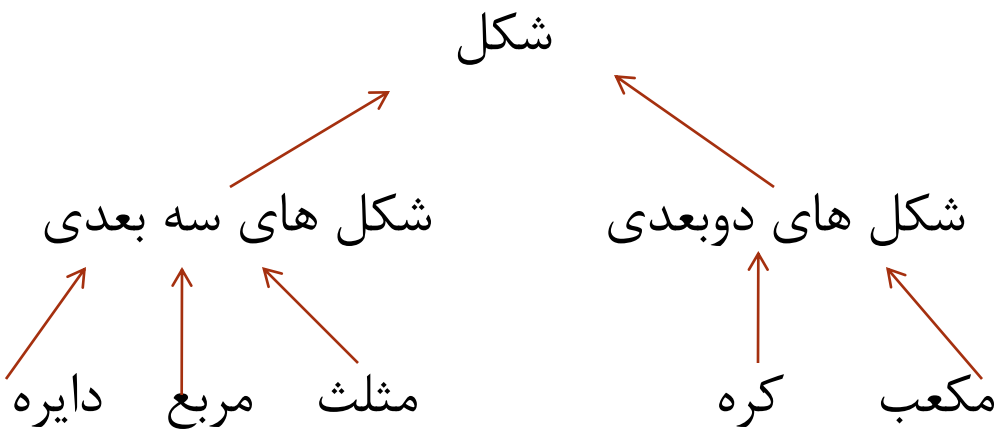
تخصیص پویای اشیاء

- ✓همانطور که متغیر ها را می توان با استفاده از عملگر new در زمان اجرای برنامه ایجاد کرد، اشیاء را نیز می توان با این عملگر ایجاد نمود.
- ✓بدین ترتیب شیء ای ایجاد می شود و اشاره گری به آن اشاره خواهد کرد.
- ✓پس از ایجاد این شیء، تابع سازنده ی آن در صورت وجود فراخوانی می شود و هنگام آزاد شدن شیء، تابع مخرب آن ایجاد می شود.
- ✓برای حذف اشیاء ای که به طور پویا تخصیص یافته اند از عملگر delete استفاده می شود.

وراثت

- ✓وراثت فرایندی است که در آن می توان کلاس های جدیدی را از روی کلاس های موجود ایجاد کرد به طوری که کلاس جدید صفات و رفتار کلاس موجود را به خودش اختصاص دهد، یا با استفاده از آنها صفات و ویژگی های دیگر را اصلاح و یا جایگزین نماید.
- ✓کلاس موجود که می توان رفتار و صفات آن را به ارث برد، کلاس پایه و کلاس جدید که صفات و رفتار ها را از این کلاس به ارث می برد کلاس مشتق نامیده می شود.
- ✓کلاس مشتق می تواند خودش به عنوان یک کلاس پایه برای کلاس های دیگر باشد، در این حالت وراثت چندگانه رخ می دهد.
- ✓در وراثت چندگانه کلاس مشتق، صفات و رفتارها را از چند کلاس به ارث می برد.
- ✓کلاس مشتق علاوه بر اعضای داده ای و توابع عضوی که به ارث می برد، می تواند توابع عضو دیگری برای خودش تعریف کند.
- ✓وراثت ساختار سلسله مراتبی درختی را ایجاد می کند.

ساختار سلسله مراتبی وراثت



کنترل دستیابی به کلاس پایه

- نمایش کلاس مشتق به صورت زیر است:

```
class derivedClass : accessControl baseClass
{
        تعریف کلاس
}
```

- کنترل دستیابی می تواند public، private و یا protected منظور شود.
- اگر کنترل دستیابی ذکر نشود private منظور خواهد شد.
- اگر کنترل دستیابی public باشد تمام اعضای عمومی کلاس پایه، اعضای عمومی کلاس مشتق خواهند بود و تمام اعضای محافظت شده ی کلاس پایه نیز به عنوان اعضای محافظت شده ی کلاس مشتق منظور می شوند.
- کلاس مشتق نمی تواند به اعضای خصوصی کلاس پایه دسترسی داشته باشد.

سازنده و مخرب در کلاس های مشتق

- ✓چون کلاس مشتق، اعضای کلاس پایه ی خود را به ارث می برد، وقتی شیء ای از کلاس مشتق ایجاد می شود، سازنده ی کلاس پایه باید فراخوانی شود تا اعضای کلاس پایه ای را که در شیء کلاس مشتق وجود دارند را مقدار اولیه دهد

ارسال پارامتر ها به سازنده ی های کلاس پایه

- ✓سازنده ی کلاس پایه می تواند دارای آرگومان باشد.
- ✓اعلان سازنده ی کلاس مشتق با کولن (:) از مشخصات کلاس پایه جدا می شود و مشخصات کلاس پایه نیز در حالتی که کلاس مشتق از چند کلاس به ارث برده باشد با کاما از هم جدا می شوند.

,(لیست آرگومان ها)کلاس پایه ی ۱ : (لیست آرگومان ها)سازنده ی کلاس مشتق
,(لیست آرگومان ها)کلاس پایه ی ۲

...

کلاس پایه ی n

{

بدنه ی سازنده ی کلاس مشتق

}

مفهوم تابع مجازی

- تابع مجازی یک تابع عضو کلاس است که در کلاس پایه اعلان می شود و در کلاس مشتق دوباره تعریف می گردد.
- برای تعریف تابع مجازی پیش از اعلان نوع تابع واژه ی virtual را مورد استفاده قرار می دهیم:

```
virtual  void  show();
```

- فرض کنید کلاس پایه و کلاس های مشتق همگی دارای تابع یکسانی باشند، اگر فرض کنیم کلاس های مشتق همگی اشیائی از کلاس های پایه باشند می توان تابع عضو کلاس پایه را فراخوانی کرده و به برنامه اجازه دهیم که در زمان اجرا خودش تشخیص دهد از کدام تابع مربوط به کلاس مشتق استفاده نماید.
- بدین منظور تابع مورد نظر باید مجازی باشد.
- تابع مجازی توسط کلاس مشتق دوباره تعریف می شود.
- تعریف مجدد توابع مجازی توسط کلاس مشتق را اصطلاحا overriding می گویند.
- تعریف مجدد توابع همنام نیز overloading نامیده می شود.

تفاوت های overriding و overloading

- ✓الگوی تابع مجازی دقیقا باید با الگوی مشخص شده در کلاس پایه یکسان باشد در حالی که در توابع همنام عادی اینگونه نیست.

- ✓در تعریف توابع مجازی الگوها باید دقیقا یکسان باشند، اگر یکسان نباشند ماهیت مجازی بودن آن ها از بین می رود و تابع همنام تلقی خواهند شد.

- ✓توابع مجازی باید اعضای غیر استاتیک کلاس ها باشند.

- ✓توابع سازنده نمی توانند مجازی باشند ولی توابع مخرب می توانند مجازی باشند.

صفت مجازی موروثی است

- ✓وقتی تابع مجازی توسط کلاسی به ارث برده می شود ماهیت مجازی بودن آن نیز به ارث برده می شود.

توابع مجازی محض

- ✓اگر یک تابع مجازی در کلاس مشتق تعریف نشود تابع موجود در کلاس بالایی آن در سلسله مراتب وراثت آن مورد استفاده قرار می گیرد.

- ✓تابع مجازی محض نوعی تابع است که در کلاس پایه اعلان شده است ولی تعریف نشده باشد.

- ✓وقتی تابع مجازی به صورت محض باشد هر کلاس مشتق باید تعریف خاص خودش را ارائه نماید.

کلاس انتزاعی

- ✓کلاسی که دارای یک تابع مجازی محض باشد کلاس انتزاعی (abstract) نامیده می شود.

- ✓چون کلاس انتزاعی حاوی یک یا چند تابع است که تعریفی برای آنها ارائه نشده است، هیچ شیء ای از کلاس انتزاعی نمی تواند ایجاد شود.

- ✓کلاس انتزاعی یک نوع ناقص را ایجاد می کند که به عنوان مبنایی برای کلاس های مشتق مورد استفاده قرار می گیرد.

توابع سازنده (constructor)

- ❖ برای مقدار اولیه دادن به اشیاء از تابع عضو ویژه ای به نام سازنده استفاده می شود.
- ❖ سازنده تابع عضوی با مشخصات زیر است:
  - ✓ همنام با کلاسی است که برای آن تعریف می شود و کامپایلر از این طریق آن را از توابع عضو دیگر کلاس تشخیص می دهد.
  - ✓ هیچ مقداری را بر نمی گرداند.
  - ✓ با کنترل دستیابی public مشخص می شوند.
  - ✓ به طور خودکار هنگام ایجاد شیء ای از کلاس اجرا می شوند و صریحا فراخوانی نمی شوند.
- ❖ C++ برای هر شیء ای که ایجاد می کند نیاز به سازنده دارد تا اطمینان حاصل کند که هر شیء ای که بوجود می آید دارای مقدار اولیه است.
- ❖ اگر در کلاس صریحا سازنده قید نشود کامپایلر از سازنده ی پیش فرض استفاده می کند که فاقد پارامتر است