شیءگرائی و کلاس

مهمترین تفاوت C و C++ اشیا هستند. این فصل مقدمه ای بر تئوری شیءگرائی است. شیءگرائی مفهوم کلاس ها را معرفی می کند که چهار ویژگی کلی را در شیءگرائی نشان می دهند: انتزاع، کپسوله کردن، توارث و چندریختی. تعریف کلاس و اجزای کلاس، نحوه تعریف تابع عضو و تابع دوست، سازنده ها و مخرب ها در اینجا توضیح داده خواهد شد.

مفاهیم شیء گرائی
کلاس
سازنده ها
مخرب ها
آرایه ای از اشیا

مفاهیم شیء گرائی

برنامه نویسی شیءگرائی (object oreinted programming) وسیله ای برای مدل کردن صحیح دنیای واقعی با استفاده از اشیا (objects) در برنامه و استفاده مجدد از کد است. یک شی در برنامه دقیقا همان طور تعریف می شود که در دنیای واقعی است؛ خواص معینی دارد که آن را توصیف می کند و متدهایی که می توانید برای انجام کار معینی روی شیء استفاده کنید.

هدف کلی C++ اضافه کردن شیءگرائی به زبان برنامه نویسی C است. یک شیء برای نگهداری داده استفاده می شود. داده و توابعی که روی داده کار می کنند به هم مربوط هستند بنابراین داده و توابع هردو با هم دریک بسته قرار می گیرند. شیءگرائی بیشتر روی داده تاکید دارد تا عملیات و توابعی که روی داده کار می کنند.

مثال. ماشین یک شی است دارای خواصی مثل رنگ، تعداد درها و غیره است متدهای معینی دارد مانند سرعت گرفتن، ترمز کردن و غیره. می توان این شی را با استفاده از متدهایش استفاده کرد.

شرحی از داده ها و توابعی که می توانند روی داده کار کنند را کلاس (class) می نامند. کلاس را به عنوان الگوئی برای تولید شیء می توان تصور کرد. کلاس در واقع یک نوع داده user-defined است . اشیاء نمونه هائی از کلاس ها هستند که در زمان اجرا ایجاد می شوند.

چهار مفهوم اصلی وجود دارند که اساس برنامه نویسی شیءگرائی را می سازند و توسط کلاس ها ارائه می شوند. این مفاهیم انتزاع (abstraction)، کپسوله کردن (encapsulation)، توارث (inheritance) و چندریختی (polymorphism) هستند.

انتزاع

شیء گرائی ابزاری را برای برنامه نویس فراهم می کند که اجزای فضای مسئله را توسط اشیا نمایش دهد. مسئله به بخش های تشکیل دهنده تجزیه می شود. هر مولفه یک شیء می شود که شامل داده های مرتبط و دستورالعمل های خود است. به این ترتیب مسئله به همان صورتی که در دنیای واقعی هست توصیف می شود نه به روشی کامپیوتری که مسئله را حل می کند. یعنی می توانید با همه چیز در یک ترتیب کلی سروکار داشته باشید. پیچیدگی عملیات کاهش یافته و جزییات پیاده سازی مخفی می ماند.

مثال. درباره خصوصیات کلی وسیله نقلیه بدون سروکارداشتن با یک وسیله و مدل خاص می توان بحث کرد. یک نمونه شیء MyVehicle از کلاس Vehicle که خواص کلی و توابع وسایل نقلیه را در بر دارد می توان ایجاد کرد. تابع Print مشخصات کلی وسیله نقلیه را نمایش می دهد.

```
Vehicle MyVehicle;

MyVehicle.Print();
```

کپسوله کردن

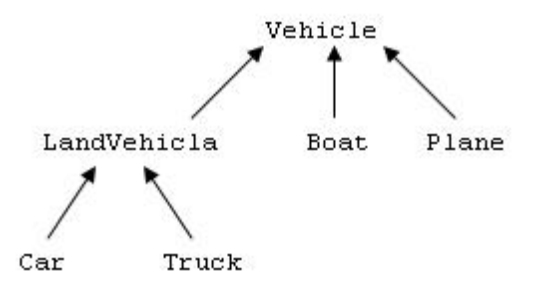
قرار دادن داده و توابعی که روی داده کار می کنند را در یک بسته کپسوله کردن می گویند. در برنامه نویسی رویه گرا (مشابه آنچه تا کنون انجام می دادید) معلوم نیست چه تابعی روی چه متغیری کار می کند. در برنامه های پیچیده تر این روابط تیره تر می شوند. در برنامه نویسی شیءگرائی داده و توابع مربوط به آن- که اغلب متد (method) نامیده می شوند- با در یک بسته به نام کلاس قرار می گیرند بنابراین کاملا مشخص است چه تابعی روی چه داده ای کار می کند.

کپسول کردن امکان پنهان کردن اطلاعات را نیز فراهم می کند. هر عضو کلاس را می توان به صورت عمومی، خصوصی یا محافظت شده مشخص کرد. شی ایجاد شده از کلاس به داده از طریق یک سری توابع عمومی دسترسی دارد و اجازه دسترسی مستقیم به اعضای خصوصی کلاس و خراب کردن آنها داده نمی شود. به این صورت جزئیات پیاده سازی هم مخفی می ماند و به طراح اجازه می دهد پیاده سازی را اساسی بدون تغییر در واسطه ها عوض کند.

وراثت

یکی دیگر از جنبه های مفید برنامه نویسی شیءگرائی قابلیت استفاده مجدد از کد است. یک کلاس می تواند اعضای عمومی را از کلاس دیگر را به ارث ببرد. توارث اجازه می دهد کلاس جدیدی شامل کلیه داده ها و توابع کلاس (های) پیاده سازی شود. کلاس موجود را کلاس پایه (base) و کلاس جدید که اعضای کلاس پایه را به ارث می گیرد را کلاس مشتق شده (derived) می نامند.

مثال. فرض کنید یک کلاس پایه به نام Vehicle برای وسایل نقلیه داریم. وسیله نقلیه می تواند ماشین، تراکتور، قایق و هواپیما را شامل شود که ممکن است هر یک از آنها احتیاج به اطلاعات یا توابع اضافی داشته باشند. بنابراین می توان برای هر کدام کلاس های فرعی را تعریف کرد که خواص کلاس Vehicle را به ارث می برند علاوه براین که دارای فیلدهای جدید دیگری هم هستند.



اگر کلاس ویژگی های تنها یک کلاس را به ارث ببرد وارثت منفرد(single inheritance) و اگر از چند کلاس به ارث گرفته شود وارثت چندگانه (multiple inheritance) نامیده می شود.

ارث از کلاس پایه ممکن است به صورت عمومی، خصوصی یا محافظت شده باشد. این مشخصه های دسترسی تعیین می کنند کلاس های مشتق شده می توانند به اعضای عمومی و محافظت شده کلاس پایه دسترسی پیدا کنند یا خیر. تنها وراثت عمومی است که با مفهوم توارث تطبیق دارد. دو فرم دیگر کمتر استفاده می شوند.

چندریختی

یک تابع در سلسله مراتب کلاس ها و زیر کلاس ها می تواند به طرق مختلف پیاده سازی شود و شکل های متعددی بگیرد. چندریختی یک رابط مشترک را برای پیاده سازی های مختلف از یک تابع در اختیار می گذارد که برای اشیا تحت شرایط مختلف متفاوت عمل کند. به عبارت دیگر فراخوانی تابعی که متعلق به کلاس است همیشه به شکل زیر خواهد بود.

```
object.function(parameter-list)
```

مثال. در مثال قبل تابع نمایش برای قایق ممکن است متفاوت از نمایش یک وسیله نقلیه باشد. دو تابع می توانند هم نام باشند ولی با کدهای متفاوت بسته به کلاس شیئی که تابع را فراخوانی می کند.

```
Vehicle MyVehicle;
Boat    YourBoat;
MyVehicle.Print();
YourBoat.Print();
```

در مثال قبل کامپایلر نوع تابع print را برای دو نوع فراخوانی شده بر اساس نوع شیء می تواند پیدا کند. اما گاهی پیدا کردن آن تا زمان اجرای واقعی برنامه ممکن نیست. مشکل زمانی بروز می کند که به شیء از طریق اشاره گر دسترسی می شود و چون اشاره گرها می توانند بطور پویا به انواع مختلفی از اشیا اشاره کنند کلاس شیء تا زمان اجرا مشخص نمی شود. برای حل این موضوع توابع عضو مجازی بکار می روند. توابع مجازی عضو (virtual member functions) اجازه پیاده سازی خاص تری از تابعی که فراخوانی می شود را مطابق با نوع شیء زمان اجرا می دهند.

کلاس

همانطور که قبلا دیدید ساختمان ها نوع داده جدیدی را ایجاد می کنند. کلاس ها به طور مشابه روش قدرتمند تری برای ایجاد یک نوع داده جدید هستند. وقتی یک کلاس تعریف می شود در اصل نوع داده جدیدی ساخته می شود که فیلدها و توابع خود را دارد. کلاس به عنوان قالبی برای تولید شی بکار می رود بنابراین از خود کلاس در برنامه استفاده نمی شود بلکه یک نمونه از آن که شیء نامیده می شود اعلان می شود. فرآیند تولید یک نمونه از کلاس یا ایجاد یک شی از کلاس به این معنا است که یک متغیر از نوع کلاس اعلان شود.

کلاس مشابه ساختمان تعریف می شود فقط کافی است ابتدای آن کلمه کلیدی class ذکر شود. دقت کنید در انتهای بلاک علامت سمیکولن را فراموش نکنید.

```
class classname
{
// members
};
```

وقتی یک کلاس ایجاد می کنید مانند هر متغیر دیگر می توانید یک نمونه از آن را بگیرید:

```
classname object_variable;
```

مثال.

```
class myclass {
  int number;
  void greeting();
};
```

حوزه اعضای کلاس

هر عضو ساختمان یا کلاس می تواند عمومی(public)، خصوصی (private) یا محافظت شده (protected) باشد. به طور پیش فرض تمام اعضای کلاس خصوصی هستند و امکان دسترسی به آنها خارج از کلاس ممکن نیست مگر اینکه توسط حوزه های دسترسی به عنوان عموی یا محافظت شده تعریف شوند.

حوزه های دسترسی (access modifiers) توسط کلمات زیر مشخص می شوند:

- private. اعضایی که به صورت خصوصی مشخص می شوند تنها درون کلاس توسط توابع عضو و توابع دوست قابل دسترسی هستند و نمی توانند خارج از کلاس دسترسی شود.
- public. اعضای عمومی از هر تابعی حتی خارج از کلاس قابل دستیابی هستند.
- protected. اعضای محافظت شده نمی توانند بیرون از کلاس دسترسی شوند اما توسط توابع عضو خودش و توابع دوست و از کلاسی که از آن مشتق شده قابل دسترسی هستند.

مثال. متغیر number عضو خصوصی و تابع greeting.

```
class myclass {
  int number;
public:
  void greeting();
};
```

نکته. در یک ساختمان همه اعضا عمومی هستند مگر اینکه آنرا به عنوان خصوصی یا محافظت شده تعریف کنید.

مثال. در ساختمان A توابع عضو به صورت عمومی تعریف شده اند و فیلدها به صورت خصوصی هستند.

```
struct A {
private:
  int i, j, k;
public:
  int f(){return i + j + k;}
  void g(){i = j = k = 0;}
};
```

توابع عضو

توابع عضو (member functions) توابعی هستند که درون کلاس تعریف می شوند و متعلق به کلاس هستند. توابع عضو می توانند به کلیه اعضای کلاس (اعم از عمومی، خصوصی و محافظت شده) بدون هیچ محودیتی دسترسی پیدا کنند.

مثال. تابع greeting تابع عضو کلاس است که به متغیر خصوصی عضو number دسترسی دارد.

```
class myclass {
  int number;
public:
  void greeting(){
     for(int i = 0; i<= number; i++ ) cout<< "Hello World \n";
  }
};
```

در مثال فوق کد تابع عضو به صورت درونی (inline) در کلاس قرار داده شده است. توابع درونی درنقطه فراخوانی به صورت خطی گسترش پیدا می کند به جای اینکه واقعا فراخوانی شود. توابع درونی درصورتی که بدنه تابع کوچک باشد روش کارآمدتری هستند. راه دیگر تعریف تابع عضو این است که بدنه تابع بعد از بلاک کلاس قرار گیرد. سپس برای ارتباط تابع عضو با کلاس قبل از نام تابع نام کلاس بدنبال علامت (::) باید ذکر شود. :: عملگر حوزه (scope operation) نام دارد و بیان کننده این است که تابع متعلق به کلاس است.

مثال. کلاس فوق را به صورت زیر نیز می توان نوشت.

```
class myclass {
  int number;
public:
  void greeting();
};
void myclass::greeting(){
  for(int i = 0; i<= number; i++ )
     cout<< "Hello World \n";
}
```

توابع دوست

توابع دوست (friend function) توابعی هستند که عضو کلاس نیستند اما به اعضای خصوصی کلاس دسترسی دارند. برای ایجاد یک تابع دوست در کلاس پروتوتایپ تابع را در بخش عمومی کلاس قرار داده و قبل از آن کلمه friend استفاده کنید.

مثال. دربرنامه زیر پیغام سه بار نمایش داده می شود. تابع set یک تابع دوست است که برای مقداردهی متغیر خصوصی عضو کلاس استفاده شده است.

```
#include <iostream.h>

class myclass {
   int number;
public:
   friend void set(myclass, int);
   void greeting();

};

void myclass::greeting(){
   for(int i = 0; i<= number;i++)
      cout<< "Hello World \n";
}

void set(myclass n, int value){
   n.number=value;
}

int main () {
   myclass myobject;

   set(myobject, 3);
   myobject.greeting();
   return 0;
}
```

نکته. یک تابع ممکن است دوست بیش از یک کلاس باشد.

سازنده ها

معمولا بعضی از اعضای کلاس قبل از استفاده نیاز به مقداردهی دارند. این عمل توسط سازنده (constractor) انجام می گیرد که به شیء این امکان را می دهد که هنگام ایجاد مقداردهی شود. سازنده تابعی است هم اسم کلاس که وقتی یک نمونه از کلاس گرفته می شود اتوماتیک فراخوانی می شود.

تابع سازنده می تواند دارای پارامتر باشد بنابراین زمان ایجاد شیء می توان به متغیرهای عضو مقادیر اولیه داد. برای ارسال آرگومان به تابع سازنده باید هنگام تعریف شیء مقدار آرگومان بعد از نام شیء درون پرانتز قرار گیرد.

یک کلاس می تواند دارای چند سازنده با پارامترهای مختلف باشد. بهتر است همیشه حداقل یک سازنده حتی اگر خالی باشد ساخته شود.

برای تابع سازنده مقدار برگشتی ذکر نمی شود (حتی viod).

مخرب ها

تابع مخرب کلاس (destructor) کم و بیش عکس سازنده عمل می کند. یک مخرب وقتی فراخوانی می شود که یک شی از بین می رود. یک مخرب مشابه سازنده ساخته می شود فقط قبل از اسم آن علامت مد (~)قرار می گیرد. تابع مخرب اتوماتیک وقتی متغیر شیء از حوزه دسترسی خارج می شود (برای متغیرهای سراسری وقتی از تابع اصلی خارج می شود و برای متغیر محلی هنگام خروج از بلاک تابع) فراخوانی می شود.

مشابه سازنده ها تابع مخرب نیز نوع برگشتی ندارد.

مثال. توابع myclass در کلاس زیر سازنده هستند. تابع ~myclass یک مخرب است که در انتهای تابع اصلی فراخوانی می شود و فایل متن را می بندد.

```
#include <fstream.h>
#include <iostream.h>
#include <string.h>

class myclass {
private:
   char msg[20];
   int loopcounter;
   fstream myfile;
public:
   void greeting();
   myclass();                    // Constructor
   myclass(char greeting[20]);   // Constructor
   ~myclass();                   // Destructor
};

myclass::myclass(){
   myfile.open("input.txt",ios::in);
   myfile.getline(msg,20);
}

myclass::myclass(char greeting[20]){
   myfile.open("input.txt",ios::in);
   strcpy(msg,greeting);
}

myclass::~myclass(){
   myfile.close();
}

void myclass::greeting(){
   cout << msg << "\n";
}

int main (){
   myclass myobject;
   myobject.greeting();
   return 0;
}
```

در برنامه فوق هنگام ایجاد شیء myobject تابع سازنده فراخوانی شده خطی را از فایل متن خوانده و به متغیر عضو تابع اختصاص می دهد. اگر در برنامه اصلی شیء به صورت زیر ایجاد شود سازنده دوم فراخوانی می شود و مقدار آرگومان را به متغیر msg اختصاص می دهد.

```
myclass myobject("Howdy from Texas!");
```

نکته. اغلب کلاس ها را در فایل های هدر تعریف می کنند. ترکیب کلاس و فایل هدر سطح بالاتری از قابلیت استفاده مجدد کد را فراهم می کند و می توان آن را در هر فایلی که به کلاس نیاز دارید ضمیمه کنید.

آرایه ای از اشیا

می توان به همان طریقی که آرایه ای از سایر انواع داده ایجاد می شود آرایه ای از اشیا تعریف کرد. اگر کلاس دارای تابع سازنده همراه با پارامتر باشد آرایه ای از اشیا را می توان مقداردهی کرد.

مثال. myarray آرایه ای از اشیا است که در برنامه اصلی با چهار عدد مقداردهی شده و نمایش داده می شود.

```
#include <iostream.h>

class display {
  int number;
public:
  display(int n) {this->number=n;}
  int show() { cout << this->number << endl; }
};

int main() {
   display myarray[4] = {1,2,3,4,};
   for (int i = 0; i<4 ;i++ )
      myarray[i].show();
   return 0;
}
```

هر بار که تابع عضوی فراخوانی می شود به طور خودکار اشاره گری به نام this را به شیئی که آن را فراخوانی کرده است ارسال می کند. اشاره گر this یک پارامتر ضمنی برای تمام توابع عضو است و داخل تابع برای رجوع به شیء فراخواننده می تواند مورد استفاده قرار گیرد.