فصل دوم
مقدمه ای بر برنامه نویسی شی گرا

نوع(Type)

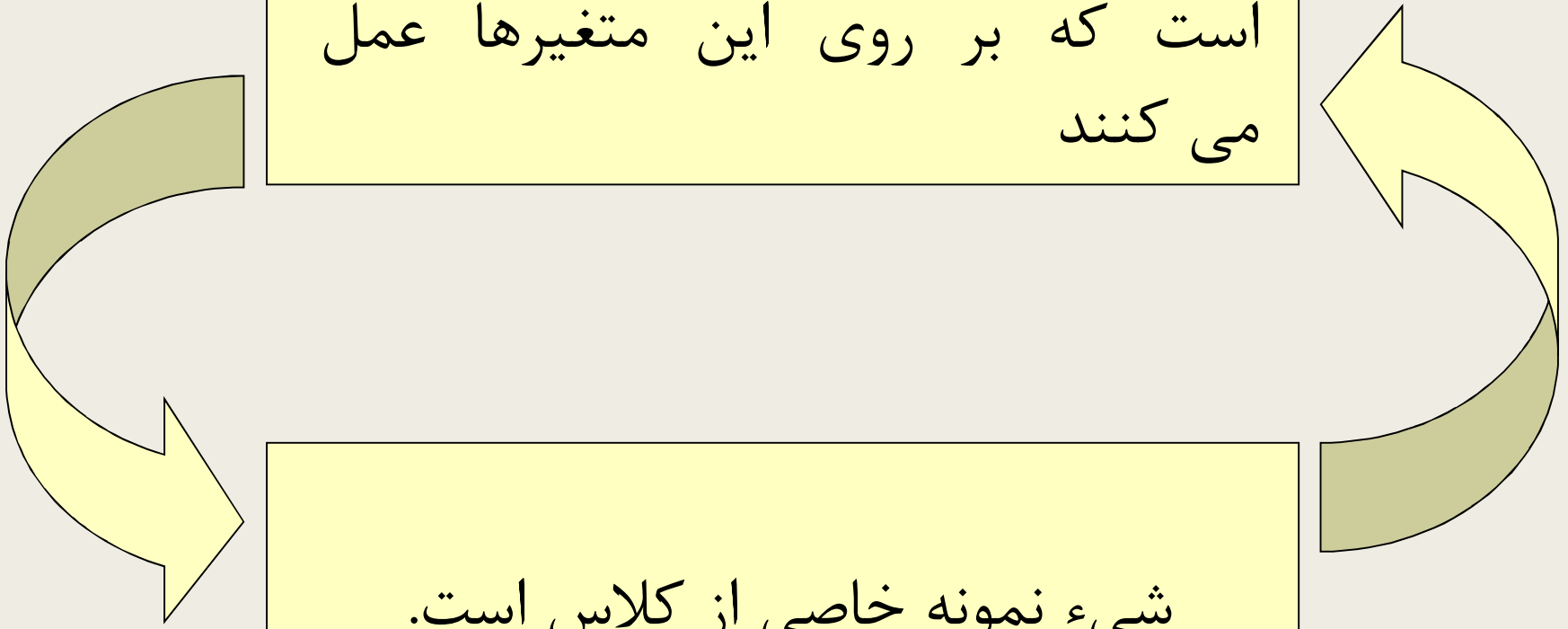
- **یک دسته بندی یا طبقه است**
- **یکی از توانمندی های انسان توانایی طبقه بندی است**
- **مثال: انسان، حیوان، گیاه و ...**

در ++c، شیئی با اندازه، حالت و مجموعه ای از توانایی ها را نوع(Type) می گویند

- **برنامه ها معمولا برای حل مسائل واقعی نوشته می شوند**
- **برای تعریف انواع جدید از کلاس استفاده می شود**

شی(Object) – کلاس(Class)

کلاس، مجموعه ای از متغیرها و توابعی است که بر روی این متغیرها عمل می کنند



شیء نمونه خاصی از کلاس است.

مولفه های برنامه‌نویسی شی گرا

- **سه مولفه اصلی برنامه نویسی شی گرا**
  1. ***بسته بندی اطلاعات***
  2. ***پنهان سازی اطلاعات***
  3. ***وراثت***
- **بسته بندی اطلاعات با تعریف یک شی، رفتارهایش و پیاده سازی بصورت کلاس انجام می‌شود**
- **پنهان سازی اطلاعات با تعریف رفتارها، صفتهای عمومی و خصوصی و پیاده سازی بصورت تعیین نوع عضویت در کلاس انجام میگیرد**
- **وراثت شبیه وراثت بیولوژیکی است که در آن فرزندان ، صفاتی را از والدین به ارث می برند. در این رابطه می توان از کلاس موجود (پایه)، کلاس جدید(فرزند) را ایجاد کرد.**

انتزاع داده ها (data abstraction)

- فقط ویژگی های اساسی انواع، بدون ارائه اطلاعات جزئی قابل نمایش است.
- کلاس ها از مفهوم نوع داده انتزاعی(ADT) پیروی می کند

بسته بندی(Encapsulation)

- در برنامه نویسی شی گرا هر شی از یک سری متغیر های عضو به نام صفت (attribute) و یک سری توابع که به تابع عضو معروف‌اند تشکیل می‌شود
- باید توجه داشت که برای اعلان کلاس از کلمه کلیدی class استفاده می‌شود

دسترسی خصوصی و عمومی – پنهان سازی

- برخی ویژگی ها و یا پیچیدگی های یک شی باید از دیگر اشیا پنهان بماند و هر شی تنها یک واسط (interface) که برای دیگر اشیا لازم است را به نمایش می گذارد.
- یک صفت و یا یک رفتار می تواند درون شی پنهان باشد و دیگر اشیا از آن بی اطلاع باشند.
- مکانیسم اولیه پنهان سازی داده ها قرار دادن آن در یک کلاس و خصوصی سازی آن است.
- داده ها یا توابع خصوصی را تنها می توان از داخل کلاس در دسترس قرار گیرد. از سوی دیگر داده ها یا توابع عمومی از خارج کلاس در دسترس قرار می گیرند.

تابع عضو(Member Function)

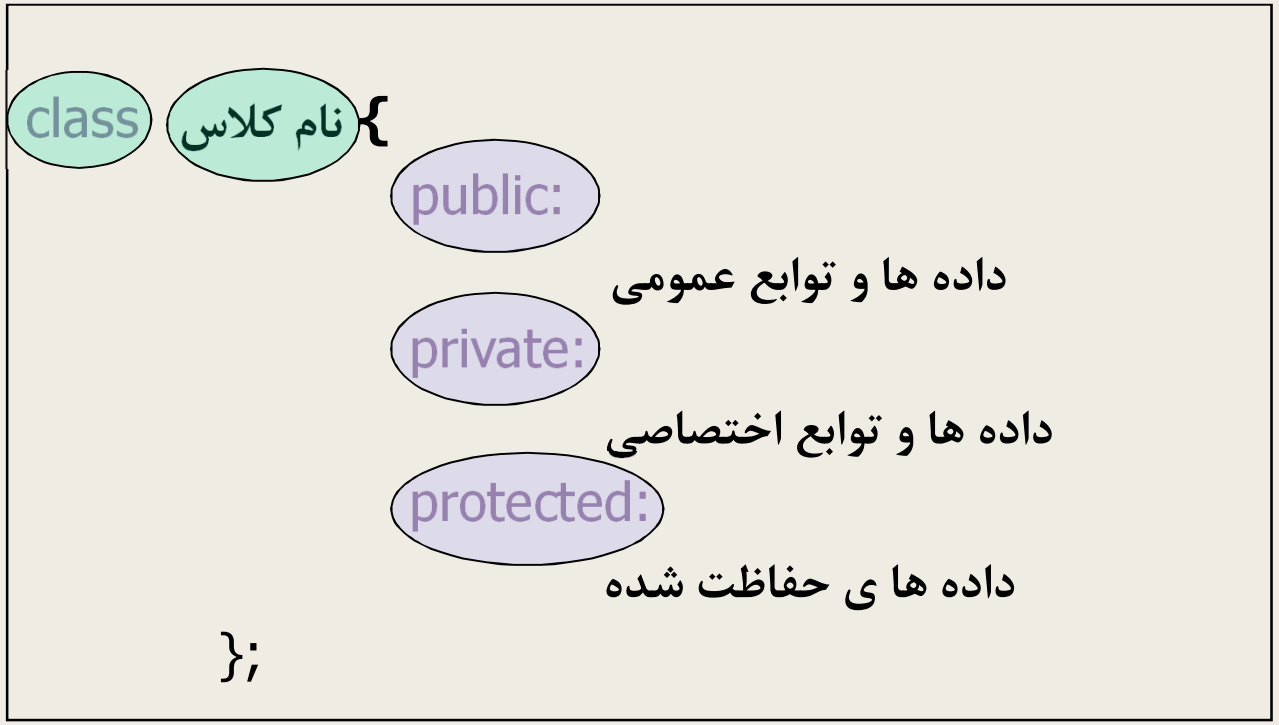
- معمولا توابع عمومی وداده ها به صورت خصوصی تعریف می‌شوند از این رو از داده ها فقط در توابع عضو کلاس استفاده می‌شوند ولی از توابع عضو درخارج از کلاس هم می توان استفاده کرد.
- باید توجه داشت که در بعضی مواقع مجبور می شویم از توابع خصوصی و داده های عمومی استفاده کنیم.
- مکانیسم اولیه پنهان سازی داده ها قرار دادن آن در یک کلاس و خصوصی سازی آن است.

تابع عضو

- روش های اعلان و پیاده سازی تابع عضو:
  - در داخل کلاس اعلان کرد ودر همان جا پیاده سازی شود (در صورتی که تعریف تابع کوتاه باشد)
  - در داخل کلاس اعلان کرد و در خارج کلاس پیاده سازی شود

تعریف کلاس(Class) در c ++

- برای تعریف کلاس از کلمه کلیدی class و ساختاری به صورت زیر استفاده می شود:



- نام گذاری کلاس همانند متغیرها انجام می شود.
- نام کلاس باید بیانگر موجودیت باشد و تا حد امکان از اسامی بی ربط استفاده نشود.

تعریف کلاس

- تعریف نوع دسترسی ها:
  - توابع یا داده هایی که بعد از تعریف کلاس یا بعد از کلمه کلیدی private اعلان می‌شوند ، برای کلاس اختصاصی خواهند بود. فقط و فقط اجزای همان کلاس حق استفاده و دسترسی به آنها را دارند.
  - داده ها و توابعی که بعد از کلمه کلیدی public تعریف می‌شوند به صورت عمومی خواهند بود. هر قسمت دیگر برنامه می تواند به آنها دسترسی داشته باشد.
  - توابع و داده هایی که پس از کلمه کلیدی protected تعریف می‌شوند محافظت شده هستند و در وراثت مورد استفاده قرار می گیرند.

شی(Object)

- برای تعریف شی از یک کلاس در هر جایی از برنامه به صورت زیر عمل می کنیم:

- نام کلاس نام شی ء;

- همانطور که می بینید تعریف اشیاء همانند تعریف متغیرها است

مثالی از کلاس و شی

- **مثال۲–۱: برنامه شامل یک کلاس و دو شیء از آن کلاس است**

- ***تعریف کلاس***

```
#include <iostream.h>
class smallobj
{
    private:
       int somedata;
    public:
       void setdata(int d)
       {
          somedata=d;
    }
       void showdata()
       {
       cout<<"data is"<<somedata;
    }
};
```

مثالی از کلاس و شی

- تعریف اشیاء

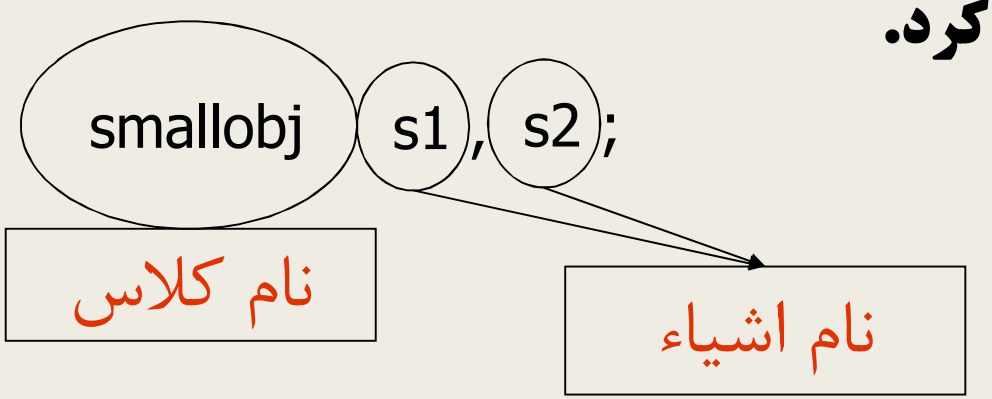
```
int main()
{
    smallobj s1,s2;
    s1.setdata(1006);
    s2.setdata(876);
    s1.showdata();
    s2.showdata();
    return 0;
}
```

تشریح مثال

- کلاس smallobj که در این برنامه اعلان شده حاوی یک عنصر داده‌ای و دو تابع عضو می‌باشد که این دو تابع با عضو داده ای داخلی کار می کنند و از خارج کلاس در دسترس هستند
- تابع عضو اول به عضو داده ای یک مقدار نسبت می دهد.
- تابع عضو دوم این مقدار را در خروجی نمایش می دهد.

ساخت شی از کلاس

- پس از اعلان کلاس می توان در داخل تابع main() از این کلاس ها برای تعریف شیء استفاده کرد.



- تعریف یک شیء شبیه تعریف یک متغیر از هر نوع داده ای است که برای حافظه گرفته می شود.
- چون شیء در واقع نمونه ای از یک کلاس است به تعریف شی نمونه سازی شیء نیز می گویند (توجه داشته باشید که اتلاقهای فوق در هنگام پیاده سازی معتبر هستند)

فراخوانی توابع عضو

- دستور زیر تابع عضو ()setdata را احضار می کند:

```
s1.setdata(1006)
```

- از آنجا که تابع ()setdata یک تابع عضو از کلاس می‌باشد باید در ارتباط با یک شیء از این کلاس (s1) احضار شود پس نوشتن دستور زیر باعث تولید پیغام خطای کاربر می شود:

```
setdata(1006)
```

- برای استفاده از یک تابع عضو عملگر نقطه، نام شیء و تابع عضو را به هم وصل می کند

نکته: توابع عضو کلاس حتماً باید برای یک شیءساخته شده فراخوانی شوند

مثالی از تعریف کلاس

```
class employee {
      char name[20];    // private by default
      public :
            void  putName (char  * s);  //publics
            void  getName (char  * s);
            void  putWage (double w);
            double getWage ();
      private :
            double Wage;    //private again
} em1 , em2 ;
employee em3, em4 ;
void  employee::putName ()
{
      . . .
}
void  employee::getName ()
{
      . . .
}
```

اپراتور محدوده کلاس

مثالی از تعریف کلاس

- برنامه ای بنویسید که یک شی دایره در آن تعریف شود. شعاع دایره را از ورودی خوانده، مساحت آن را محاسبه کرده و در خروجی ببرد. (تمام اعضای داده ای خصوصی اند)

```
#include <conio.h>
#include <iostream.h>
class circle{
          int radius;
     public:
          void get_radius();
          void print();
     };
void   circle : : get_radius()
{
   cout<<"Enter radius";
   cin>> radius;
}
```

مثالی از تعریف کلاس

```
void   circle : : print ()
{
   float   s;
   s = radius * radius * 3.14;
   cout << " Area = " << s;
}
int main ()
{
          circle    c1;
          c1.get_radius();
          c1.print();
          getch();
          return 0;
}
```

تمرین ۲

- بخش ۱:
- یک کلاس(CLesson) برای مشخصات درس تعریف نمایید
  - صفات: نام درس(LName) – تعداد واحد(LUnit) – نمره(LGrade)
  - رفتارها: قراردادن نام برای درس(LSetName) – قرار دادن تعداد واحد برای درس(LSetUnit) – قرار دادن نمره برای درس(LSetGrade) – توابعی برای برگرداندن مقادیر نسبت داده شده(LShowName- LShowGrade- LShowUnit)
  - دیگر رفتارها: محاسبه نمره در واحد و برگرداندن آن(LSetValue-LShowValue)

تمرین ۲

- **بخش ۲:**
  - **برای تست کلاس نوشته شده با نام CLesson، درون تابع main اشیا مناسب از نوع کلاس را تعریف نموده و متغیرهای عضو آن را مقداردهی نمایید و مقادیر نسبت داده شده را نمایش دهید.**

تمرین ۲

- **بخش ۳:**
- **یک کلاس با نام CStudent تعریف نمایید.**
- **صفات: نام دانشجو(SName) – شماره دانشجویی(SNo) – سه درس در نیمسال (از نوع Clesson که قبلاً طراحی شده)**
- **رفتارها: توابع مورد نیاز جهت مقدار دهی به نام و شماره دانشجویی و برگرداندن این مقادیر**
- **دیگر رفتارها: تعیین نام برای هر یک از دروس دانشجو و نمره برای آن درس**
- **دیگر رفتارها: محاسبه معدل با توجه به نمره دانشجو در دروس.**

روش ایجاد برنامه شی گرا

- تعیین نیازمندی های مسئله
- تحلیل مسئله(ورودی و خروجی)
- طراحی کلاس ها
- پیداکردن کلاس های موجود و اصلاح شده
- اصلاح کردن کلاس های موجود، درصورت نیاز
- طراحی کلاس های جدید، در صورت لزوم
- پیاده سازی کلاس های جدید
- تست و بررسی کل برنامه
- نگهداری و بازسازی برنامه

سازنده ها (Constructor)

- برای پی بردن به مفهوم سازنده ها (constructor) روش مقدار دادن به یک متغیر را به یاد آورید. دستور زیر ضمن تعریف متغیر y از نوع int مقدار اولیه آن را صفر تعیین می کند.

```
int  y = 0;
```

- کلاس می تواند تابع عضو ویژه ای به نام سازنده داشته باشد ولی این تابع هیچ مقداری را نمی تواند برگرداند و حتی از نوع voidهم نیست.
- سازنده ، همنام با کلاسی است که در آن تعریف می شود و هنگام ایجاد اشیایی از آن کلاس به طور خودکار اجرا می شود.

سازنده ها – مثال

```
class myClass {
      int a;
      int b;
      public:
            myClass();
            void show();
                  };
myClass ::myClass ( )
{
      a = 0 ;
      b = 10;
}
void myClass : : show ( )
{
      cout << "a = " << a ;
      cout << " b = " << b ;
}
int main ()
 {
            myClass ob1;
            ob1.show();
            return 0;
}
```

سازنده ها

- تمامی کلاس ها دارای سازنده هستند و اگر ما برای یک کلاس، سازنده ننویسیم کامپایلر از سازنده پیش فرض (Default Constructor) استفاده می کند.
- فراخوانی سازنده توسط کامپایلر هنگام تعریف متغیر بصورت خودکار انجام می شود.

سازنده با پارامتر

- توابع سازنده می توانند پارامتر داشته باشند. معمولاً این پارامترها برای مقدار اولیه دادن به متغیر های عضو شیء به کار می روند. مثال زیر را در نظر بگیرید:

```
class myClass {
     int x;
     int y;
     public:
               myClass ( int i, int j);
               void show();
};
myClass : : myClass (int i, int j )
{
     x = i ;
     y = j ;
}
```

سازنده با پارامتر

```
void myClass : : show ( )
{
    cout << "x = " << x ;
    cout << "y = " << y << endl ;
}
int main () {
    myClass   ob1 (10 , 15);
    myClass   ob2 (20 , 25);
    ob1.show();
    ob2.show();
    return 0;
}
```

- خروجی: ؟؟؟

سازنده – مقداردهی اولیه

- روش های مختلف برای مقداردهی صفات کلاس
  - روش اول: فراخوانی سازنده برای cnt

```
class counter
{
  private:
    int cnt;
  public:
    counter():cnt(0)
    {
    }
};
```

سازنده – مقداردهی اولیه

- روش های مختلف برای مقداردهی صفات کلاس
  - روش دوم: دستور انتساب

```
class counter
{
   private:
      int cnt;
   public:
      counter(){
            cnt=0;
      }
};
```

- نتیجه عملی که انجام می شود در هر دو روش یکسان است.
- پس از تعریف یک شی از کلاس counter مقدار cnt بصورت خودکار برابر صفر خواهد شد.
- باید توجه داشت که توابع سازنده هیچ نوع برگشتی ندارند زیرا سازنده به طور اتوماتیک توسط سیستم فراخوانی می شود.

سازنده – آرگومان های پیش فرض

- سازنده ها می توانند حاوی آرگومانهای پیش فرض باشند.
- با تعیین آرگومانهای پیش فرض برای سازنده حتی در صورت عدم مقدار دهی در فراخوانی سازنده، این اطمینان وجود دارد که اشیاء مقداردهی اولیه شده اند.
- سازنده ای که برنامه نویس ایجاد می کند و تمام آرگومانهای آن را به طور پیش فرض انتخاب کرده است (یا به هیچ آرگومانی نیاز ندارد)، **سازنده پیش فرض** نام دارد.
- هر کلاس فقط **یک** سازنده پیش فرض می تواند داشته باشد.

سازنده – مثال آرگومان های پیش فرض

```
class myClass {
   int x;
   int y;
   public:
     myClass (int i = 0 , int j = 10);
     void show();
};
myClass : : myClass (int i , int j )
{
   x = i ;
   y = j ;
}
```

سازنده-مثال آرگومان های پیش فرض(ادامه)

```
void myClass : : show ( )
{
    cout << "x = " << x ;
    cout << "y = " << y << endl ;
}
int main () {
    myClass   ob1 (11 , 99);
    myClass   ob2 ();
    ob1.show();
    ob2.show();
    return 0;
}
```

- خروجی: ؟؟؟

تمرین ۳

- یک کلاس برای ذخیره سازی زمان تعریف نمایید با نام CMyTime
  - ***صفات: ساعت، دقیقه و ثانیه***
  - ***سازنده:***
    - سازنده پیش فرض که مقادیر صفر برای تمامی متغیرها لحاظ کند
    - سازنده کپی که با توجه به شی ورودی و مقادیر آن متغیرهای عضو را مقداردهی نماید
  - **توابع عضو:**
    - تابعی که تفاضل زمانی شی و شی ورودی را محاسبه نموده و بصورت یک شی CMyTime برگرداند `CMyTime CMyTime::Diff(const CMyTime &t)`
    - تابعی که حاصل جمع زمانی شی و شی ورودی را محاسبه نموده و بصورت یک شی CMyTime برگرداند `CMyTime CMyTime::Add(const CMyTime &t)`
    - تابعی که یک شی از نوع CMyTime را بعنوان پارامتر گرفته و اگر مقدار آن بزرگتر از پارامتر ورودی باشد مقدار ۱ و اگر کوچکتر بود ۱– و اگر برابر بودند مقدار صفر برگرداند.

      `int CMyTime::Compare(const CMyTime &t)`

مخرب ها (Destructor)

- همان طور که وقتی یک شی برای اولین بار اجرا می شود تابع سازنده آن احضار می‌شود با نابود شدن شیء نیز تابع دیگری به نام تابع مخرب (destructor) به طور خودکار فراخوانی می‌شود.
- تابع مخرب همنام با سازنده (یعنی همنام با کلاس) است اما قبل از آن علامت ~ (tilde)قرار می گیرد.
- مخرب ها مانند سازنده ها مقدار برگشتی ندارند و هیچ آرگومانی نیز نمی گیرند.

مخرب – مثال

```
class exObj
{
private:
    int data;
public:
    exObj() : data(0)
    {  }
    ~exObj()
    {  }
};
```

تابع دوست (friend function)

- همانطور که گفته شد، تابعی که عضو کلاسی نباشد، نمی تواند به اعضای اختصاصی آن کلاس دستیابی داشته باشد. اما اگر تابعی دوست کلاسی تعریف شود، می تواند به تمام اعضای آن کلاس دستیابی داشته باشد.
- برای اعلان تابع دوست باید الگوی آن را داخل کلاس قرار داده و کلمه کلیدی friend را قبل آن ذکر کنید.

نکته: یک تابع می تواند دوست چند کلاس باشد.

تابع دوست – مثال

```
class myClass {
    int x;
    int y;
    public:
        friend int eq ( myClass i);
        void show();
};
int eq (myClass i){
    cout << i.x;
}
```

کلاس دوست (friend class)

- کلاس ها نیز می توانند دوست کلاس دیگری تعریف شوند. در این صورت کلاس دوست و تمام توابع عضو آن، به اعضای اختصاصی کلاس دیگر دسترسی دارند.

```
class myClass {
     int x;
     int y;
     public:
          find ( int i);
          friend class min
};
class min{
     public:
          int f1();
};
```

متغیر ایستا در کلاس(static variable)

- اگر یک عنصر داده ای در یک کلاس به صورت static اعلان شده باشد صرف نظر از تعداد اشیاءی که از کلاس ساخته شده‌اند، برای متغیر عضو static تنها یک عنصر ایجاد می‌شود.
- برای یک عضو static کلاس حافظه جداگانه‌ای اخذ می‌شود و تمامی نمونه های ساخته شده از کلاس به این متغیر دسترسی دارند و هر تغییری که در این نوع عضو داده ای ایجاد شود تمامی اشیاء دیگر به آخرین مقدار دسترسی دارند
- یک متغیر عضو استاتیک بسیار شبیه به یک متغیر سراسری است

متغیر ایستا- مثال

```
#include<iostream.h>
class exObj
{
    private:
        static int count;
    public:
        exObj(){
            count++
        }
        int getcount(){
            return count;
        }
};
exObj::count = 0;
```

متغیر ایستا- مثال(ادامه)

```
int main()
{
    exObj f1,f2,f3;
    cout<<"count is:"<<f1.getcount();
    cout<<"count is"<<f2.getcount();
    cout<<"count is"<<f3.getcount();
    return 0;
}
```

متغیر ایستا – توضیح مثال

- در این مثال کلاسexObj یک عنصر داده ای به نام count ازنوع static دارد سازنده این کلاس باعث می شود count افزایش یابد
- در main() سه شیء از کلاس exObj تعریف کردیم از آنجا که سازنده سه بار فراخوانی می شود count سه بار افزایش می یابد و از آن جایی که داده count از نوع static می‌باشد در نتیجه برای هر سه شی که تابع getcount() فراخوانی می شود یک مقدار(یعنی ۳) را برمی گرداند
- اگردر این مثال از یک متغیر معمولی به جای متغیر استاتیک استفاده می کردیم مقداری که تابع getcount() برمی گرداند برای هر سه تابع عدد ۱ بود زیرا در هر شیء فقط یک بار به count اضافه می‌شود

توابع عضو ایستا (static member functions)

- توابع عضو نیز می توانند static باشند این ویژگی باعث می‌شود که بتوان بدون ساخت یک شیء، یک تابع را فراخوانی کرد
- توابع static تنها به متغیر های عضو از نوع static دسترسی دارند

توابع عضو ایستا- مثال

```
class MyMath
{
public:
      static double SQUARE(double d)
      {return sqrt(d);}
      static double POW(double d,double n)
      {return pow(d,n);}
};
int main(int argc, char* argv[])
{
      cout<<"sqrt(100)="<<MyMath::SQUARE(100.0)<<endl;
      getch();
      return 0;
}
```

بدون اینکه یک شیء از کلاس MyMath ایجاد کنیم از تابع عضوSQUARE استفاده میکنیم قبل از تابع عضو ابتدا باید نام کلاس وعلامت:: را بنویسیم

موارد کاربرد توابع و متغیرهای static

- در سیستم انتخاب واحد، تمامی دانشجویان دروس ارائه شده در ترم را می بینند و باید این اطلاعات بین همه مشترک باشد.
- در سیستم حمل و نقل، تمامی خودروها اطلاعات یکسانی از مسیرها دارند. اگر کلاسی به صورت وسیله نقلیه داشته باشیم می توانیم اطلاعات مربوط به خیابانها و تقاطع ها را بصورت متغیر static تعریف نماییم
- در بازی فوتبال، تمامی بازیکنان موقعیت توپ و مالک

اشیاء به عنوان آرگومان های تابع

- می توان از اشیاء به عنوان آرگومان ورودی یک تابع استفاده کرد به مثال زیر توجه کنید:(با رنگ قرمز مشخص شده است)

```
1.  class distance
2.  {
3.  private:
4.      int feet;
5.      float inches;
6.  public:
7.      distance() : feet(0),inches(0.0)
8.     {  }
9.     distance(int ft, float in) : feet(ft),inches(in)
10.    {  }
11.    void add_dist(distance ,distance);
12. };
```

اشیاء به عنوان آرگومان های تابع

```
13.  void distance::add_dist (distance d2,distance d3)
14.  {
15.    inches=d2.inches+d3.inches;
16.    feet=0;
17.    if(inches>=12.0)
18.    {
19.       inches-=12.0;
20.       feet++;
21.    }
22.    feet+=d2.feet+d3.feet;
23.  }
```

- در سطرهای ۷ تا ۱۱ می بینیم کلاس distance دو سازنده دارد دلیل وجود این دو سازنده این است که:

1. می خواهیم متغیر هایی از نوع distance بدون مقدار دهی اولیه به آن ها تعریف کنیم (شماره ۷و۸)
2. می خواهیم متغیر هایی که از نوع distance برای اولین بار ایجاد می کنیم مقدار دهی کنیم (شماره های ۹و۱۰و۱۱)

برگرداندن اشیاء از توابع

- تابع add_dist() که در مثال قبل تعریف کردیم را می توانیم به صورت زیر تعریف کنیم در این صورت مقدار برگشتی این تابع یک شیء می‌باشد

```
distance distance::add_dist2(distance d2)
{
    distance temp;
    temp.inches=inches + d2.inches;
    if(temp.inches>=12.0)
    {
        temp.inches-=12.0;
        temp.feet=1;
    }
    temp.feet+=feet+d2.feet;
    return temp;
}
```

چون این تابع عضو، یک شیء از نوع distance برمی گرداند نوع برگشتی این تابع distance می‌باشد واین تابع شی distance را به شی d2 اضافه می کند و جمع این دو شی را بر می گرداند

برگرداندن اشیاء از توابع

```
void distance::Display()
{
    cout<<"\n feet="<<feet;
    cout<<"\n inches="<<inches<<endl;
}
void main()
{
    distance d1(1,2),d2(5,6);
    distance d3,d4;
    d3. add_dist(d1,d2);
    d4=d1. add_dist2(d2);
    cout<<"d3 is:";
    d3. Display();
    cout<<"d4 is:";
    d4. Display();
    getch();
}
```

```
d3 is:
feet=6
inches =8
d4 is:
feet=6
inches =8
```

توابع عضو ثابت (const)

- تابع عضوثابت تضمین می کند که هیچ یک از داده عضو کلاس خود را تغییر نمی دهد.
- اگر بعد از اعلان تابع و قبل از بدنه آن کلمه کلیدی const قرار دهید آن تابع به یک تابع ثابت تبدیل می شود.
- توابع عضوی که کاری انجام نمی دهند اما داده ها را از شیء دریافت می کنند نامزدهای خوبی برای تابع ثابت شدن هستند زیرا نمی خواهند هیچ داده ای را تغییر دهند.
- ثابت ساختن یک تابع کمک می کند تا کامپایلر در صورت نیاز پیغام های خطا چاپ کند و به کاربر اطلاع می دهد که که تابع نمی خواهد چیزی را در داخل شیء آن تغییر دهد.

توابع ثابت- مثال

```
class aclass
{
     private:
          int alpha;
     public:
     void nonfunc()        //non-const member function
     {alpha=99;}           //ok
     void confunc() const    //const member function
     {alpha=99;}           //error: can't modify a member
};
```

اشیاء ثابت (const)

- هر گاه یک شیء را به صورت const اعلان کنیم دیگر نمی توانیم آن شیء را تغییر دهیم بنابراین تنها از توابع عضو const می توانیم استفاده کنیم زیرا تنها توابعی هستند که تضمین می کنند محتوای آن تغییر نمی کند.

اشیاء ثابت – مثال

```
1. class distance
2. {
3.    private:
4.          int feet;
5.          float inches;
6.    public:
7.          distance(int ft, float in) : feet(ft), inches(in)
8.          {  }
9.          void getdist()
10.         {
11.               cout<<"\n enter feet" ;  cin>>feet;
12.               cout<<" enter inchest"; cin>>inches;
13.         }
```

اشیاء ثابت – مثال

```
1.    void showdist() const
2.    {
3.    cout<<feet<<"\"<<inches<<"\";
4.    }
5. };
6. int main()
7. {
8.    const distance football(300,0);
9.    football.getdist();       //error
10.   cout<<"football=";
11.   football.showdist();    //ok
12.   cout<<endl;
13.   return 0;
14.}
```

اشیاء ثابت (const) – شرح مثال

- چون شیء football که از تعریف کردیم از نوع const می‌باشد خط ۹ باعث ایجاد خطا می شود زیرا تابع getdist یک تابع عضو از نوع const می‌باشد این در حالی است که چون تابع showdist() از نوع const می‌باشد خط ۱۱ باعث ایجاد خطا در برنامه نمی شود.