فصل



سربارگذاری

اهداف آموزشی

- نحوه تعریف و کاربرد سربارگذاری توابع در زبان C++
- نحوه تعریف و کاربرد قالب (Template) در زبان C++
- نحوه سربارگذاری عملگرای دوتایی، رابطه‌ای و منطقی
- نحوه سربارگذاری عملگرای یکتایی
- آشنایی با توابع عملگردوست و نحوه کاربرد آنها

⊡ مقدمه

در این فصل به بحث و بررسی یکی از ویژگی‌های مهم پلی‌مورفیسم (چندریختی) یعنی **سربارگذاری**[1] می‌پردازیم. در زبان C++ سربارگذاری روی توابع و عملگرها صورت می‌گیرد. به کمک سربارگذاری برنامه‌ها و کدهای پیچیده راحت‌تر مدیریت می‌شوند. به‌طور مثال به کمک سربارگذاری در عملگرها به‌سادگی می‌توان عملگرهای ریاضی مانند + یا = یا * را با اشیای تعریف شده از یک کلاس جدید به کار برد یا به کمک سربارگذاری در توابع تنها با تعریف یک تابع کار چندین تابع را به‌طور همزمان انجام داد.

⊡ سربارگذاری توابع

در نسخه‌های قدیمی‌تر C++ از کلمه کلیدی overload جهت سربارگذاری توابع استفاده می‌گردید. شکل کلی دستور به فرمت زیر بوده است:

overload function_name;

البته درحال حاضر دیگر نیازی به استفاده از کلمه overload نمی‌باشد ولی باید توجه داشت که بایستی قبل از عمل سربارگذاری، تابع مفروض معرفی شده باشد.

ولی امروزه جهت سربارگذاری توابع در زبان C++ از یک نام مشترک با پارامترهای متفاوت استفاده می‌شود که اصطلاحاً به تابع مشترک، **تابع سربارگذاری شده**[2] و به خود این عمل **سربارگذاری تابع**[3] می‌گویند.

نکته: سربارگذاری تابع مخرب امکان‌پذیر نیست ولی سربارگذاری تابع سازنده برای یک کلاس متداول و امکان‌پذیر است.

---

1. Overloading
2. Overloaded Function
3. Function Overloading

مثال: می‌خواهیم برنامه‌ای بنویسیم که تحت آن تخصیص مقدار اولیه یا عدم تخصیص مقدار اولیه به یک شیء ممکن شود. (سربارگذاری توابع سازنده)

```
#include <iostream.h>
  class  coord {
      int  x,y ;
  public :
       coord( )  {x = 0;  y = 0;}                    // تابع سازنده با مقدار اولیه صفر
       coord(int i,  int j)  { x = i;  y = j;}       // تابع سازنده با مقدار اولیه دلخواه
       void get_xy(int &i , int &j)  {i = x;  j = y;}
  } ;
void  main( )
{
    coord   o1(10, 20) , o2;
     int    x,y ;
      o1.get_xy(x,y) ;
      cout<< "(o1)X : " <<x<<", Y : " <<y<<"\n";
      o2.get_xy(x,y) ;
      cout<<"(o2)X : "<<x<<", Y :"<<y;
}
```

خروجی:

```
(o1) X : 10 , Y : 20
(o2) X : 0  , Y : 0
```

کلاس coord مختصات دکارتی یک نقطه را نشان می‌دهد (x , y). به کمک سربارگذاری تابع سازنده شیء o1 دارای مقدار اولیه دلخواه و شیء o2 با مقدار اولیه صفر تعریف شده‌اند. دستور cout مختصات x و y هر یک از اشیاء را چاپ می‌کند.

۱ـ کدام یک از گزینه‌های زیر صحیح می‌باشد؟

۱) تابع سازنده (constructor) را می‌توان گرانبار (overload) نمود ولی تابع نابودکننده را نمی‌توان گرانبار نمود.

۲) هر دو تابع سازنده و نابودکننده را می‌توان گرانبار نمود.

۳) هیچ‌کدام از توابع سازنده و نابودکننده را نمی‌توان گرانبار نمود.

۴) تابع نابودکننده را می‌توان گرانبار نمود ولی تابع سازنده را نمی‌توان گرانبار نمود.

مثال: می‌خواهیم تابعی بنویسیم که مساحت مستطیل و مساحت مربع را به کمک بحث سربارگذاری توابع محاسبه کند.

```
#include <iostream.h>
double Area(double  length, double width);
double Area(double length);
 void  main( ) {
```

```
    cout<< "Area of a non-square box = "<<Area(20.0 , 7.5) << "\n";
    cout<<"Area of a square box ="<<Area(2.5);
  {
double   Area(double length, double width)
  {
    return   length*width;
   }
double   Area(double length)
  {
    return   length*length;
  }
```

خروجی:

```
Area of a non-square box = 150
Area of a square box = 6.25
```

با توجه به مثال بالا اگر می‌خواستید که در زبان C مساحت مستطیل و مربع را محاسبه کنید مجبور بودید دو تابع مستقل تعریف کنید در حالی که به کمک سربارگذاری توابع در C++ تنها با تعریف یک تابع که در تعداد پارامترها متفاوت هستند این عمل انجام شده است.

توجه کنید که کامپایلر با توجه به تعداد پارامترها هر بار یکی از توابع Area را فراخوانی می‌کند.

نکته: در زبان C++ سربارگذاری توابع با توجه به تعداد یا نوع پارامترها انجام می‌شود.

مثال: می‌خواهیم تابعی بنویسیم که بر اساس نوع پارامتر ورودی مجذور آن را محاسبه کند.

```
#include  <iostream.h>
  long  int  sqrnum(long  i);
  float  sqrnum(float f);
  double sqrnum(double  d);
void  main( )
{
   cout<< "Sqr Long Integer = " <<sqrnum(11L) << "\n";
   cout<<"Sqr Float = "<<sqrnum(10.0F) << "\n";
   cout<< "Sqr Double = "<<sqrnum(2.5);
}
long  int   sqrnum(long i) {
     return  i * i ;
}
float   sqrnum(float  f) {
    return  f * f ;
}
double   sqrnum(double d) {
   return  d*d;
}
```

خروجی:

```
Sqr Long Integer = 121
Sqr Float = 100
Sqr Double = 6.25
```

همان‌طور که مشخص است تابع sqrnum بر اساس نوع پارامترها فراخوانی می‌شود و دیگر نیازی به نوشتن سه تابع مجزا به ازای ورودی‌های مختلف نیست. توجه کنید که اعداد اعشاری به‌طور پیش‌فرض double در نظر گرفته می‌شوند، بنابراین برای آرگومان نوع float از حرف F به معنای float استفاده شده است و همچنین برای نوع long int هم باید از حرف L به همراه آرگومان آن استفاده کرد تا با نوع int اشتباه نشود.

۲ـ در برنامه روبه‌رو کدام خط در کلاس Stack قرار گیرد تا خطا تولید نشود؟

```
class stack {
public :
        stack(int size=10) : top(-1)    {a=new int[size]; }
~stack( )  {delete[ ] a;}
void push(const int& item) {a[++top] = item;}
void pop(int& item) {item=a[--top];}                    //1
void pop(const int& item) { item = a[--top];}           //2
void pop(void& item) {item=a[--top];}                   //3
void pop(int& item) {return a[--top];}                  //4
int isempty( ) const {return top== -1;}
int isfull( ) const {return top==size -1;}
private:
    int size, top, *a;    };
void main( ) {    }
```

۱) 1　　۲) 2　　۳) 3　　۴) 4

۳ـ خروجی برنامه زیر چیست؟

```
#include <iostream.h>
int f(int i);
long f(long l) ;
void main( ) {
     cout<< f(3L) << f(3);
}
int f(int i) {return i+i;}
long f(long l) {return l*l;}
```

۱) 96　　۲) 69　　۳) 99　　۴) 66

نکته: توابع را نمی‌توان بر اساس نام متفاوت پارامترهایشان سربارگذاری کرد.

مثال:

```
int F(int x) ;
int F(int y) ;
```

در هنگام فراخوانی خطا رخ می‌دهد.

نکته: توابع را می‌توان بر اساس علامت‌دار (signed) یا غیرعلامت‌دار (unsigned) بودن پارامترهایشان سربارگذاری کرد.

مثال:

```
char  F(signed char  x) ;
char  F(unsigned char  y) ;
```

سربارگذاری تابع F صحیح است.

۴ـ خروجی برنامه زیر چیست؟

```
#include <iostream.h>
void  fn(signed  a)   {cout<<"\n signed :  " << a;}
void fn(unsigned b)  {cout<<"\n unsigned : "<< b;}
void  main( ) {
   fn(5) ;
   fn(7u);
}
```

۱) unsigned : 5 ۲) signed : 5

unsigned:7u unsigned : 7

۳) پیام خطا صادر می‌شود چون نمی‌توان توابع را بر مبنای علامت‌دار بودن یا نبودن سربارگذاری کرد.

۴) به خاطر استفاده از حرف u جلوی عدد 7 پیام خطا صادر می‌شود.

نکته: توابع را نمی‌توان بر اساس نوع ایی که برمی‌گردانند، سربارگذاری کرد.

مثال:

```
int  F(int  x) ;
float  F(int  x) ;
```

در هنگام فراخوانی خطا رخ می‌دهد.

نکته: توابع را نمی‌توان بر اساس پارامترهای مقدار یا رفرنس سربارگذاری کرد، زیرا باعث بروز ابهام برای کامپایلر می‌شود.

مثال:

```
int  F(int  x,   int y) ;
int  F(int  x,  int &y);
```

در هنگام فراخوانی خطا رخ می‌دهد.

۵ـ خروجی برنامه زیر کدام است؟

```
void f(int a,int b){cout<<"Ok";}
void f(int a,int &b){cout<<"Yes";}
main()
 {
  f(2,3);  f(6,9);
 }
```

۱) OkYes

۲) YesOk

۳) OkOk

۴) ERROR

نکته: گاهی در سربارگذاری توابع ممکن است **ابهام**[1] رخ دهد که معمولاً بروز این ابهامات به‌خاطر نوع پارامترهای ورودی تابع سربارگذاری شده می‌باشد.

مثال:

```
#include <iostream.h>
 float   fn(float  f);
 double  fn(double d);
void  main( ) {
  cout<< "Float = "<< fn(5) << "\n"  ;      // باعث بروز خطا می‌شود.
  cout<< "Double = "<<fn(1.5);
}
float   fn(float  f)
{
return  f;
}
double fn(double d)
{
 return d;
}
```

دقت کنید هنگامی که عدد 5 را به‌عنوان آرگومان به تابع fn می‌فرستید کامپایلر دچار ابهام می‌شود زیرا نمی‌داند 5 را به‌عنوان float در نظر بگیرید یا double ولی عدد 1.5 را double در نظر گرفته و fn مربوط به آن را فراخوانی می‌کند (اعداد اعشاری به‌طور پیش فرض از نوع double در نظر گرفته می‌شوند) حال اگر می‌خواستید تابع fn مربوط به float فراخوانی شود بایستی آرگومان را به صورت 1.5F می‌نوشتید.

۶ـ در برنامه روبه‌رو، اگر خط یا خط‌هایی که دارای ERROR هستند Comment شوند. خروجی کدام است؟

```
int  add(int x, int y) { return x+y;   }
float add(int x , float y) {return  x+y;  }
float  add(float  x, int y) {return x+y;   }
float add(float x, float y) { return x+y;  }
void  main( )
{
cout<< add(2,3) << add(2,2.5);
cout<< add(2.5,2) << add(2.5, 2.5);
}
```

۱) 5444　　۲) 5556　　۳) 5555　　۴) 54.54.55

1. Ambiguity

❐ استفاده از آرگومان‌های پیش‌فرض جهت سربارگذاری

با استفاده از این ویژگی مقادیر پیش‌فرض به پارامترهای توابع سربارگذاری اختصاص می‌یابد. بدین‌ترتیب هرگاه تابعی فراخوانی شود که آرگومانی برای پارامتر متناظرش وجود نداشته باشد آن‌گاه مقدار پیش‌فرض پارامتر برای آرگومان در نظر گرفته می‌شود. بنابراین هنگام فراخوانی توابع دیگر لازم نیست تمامی آرگومان‌های تابع را ذکر کنید.

مثال: تابع زیر دارای دو پارامتر پیش فرض با مقدار یک است.

```
#include <iostream.h>
void  fun(int  x=1, int y=1);
void  main( ) {
    fun(8, 18);   //  X=8 , Y = 18   خروجی:
    fun(8) ;      //  X=8 , Y=1      خروجی:
    fun( );       // X = 1 , Y = 1   خروجی:
}
void  fun(int  x ,  int  y)
 {
   cout<< "X= "<<x  ;
   cout<<", Y="<<y << "\n";
 }
```

۷ـ در برنامه روبه‌رو، خروجی کدام است؟

```
int box(int l=1 , int w=2, int h=3)
{   return l*w*h;  }
void main( )
{
  cout<< box();
  cout<< box(10);
  cout<< box(10, 10);
  cout<< box(10,10,10);
}
```

۱) ERROR ۲) 6606006000 ۳) 1101001000 ۴) 6603001000

نکته: هنگام استفاده از آرگومان‌های پیش‌فرض باید مراقب ابهاماتی که ممکن است در سربارگذاری توابع ایجاد شود، بود.

مثال:

```
int   F(int  x, int y = 0) ;
int   F(int  x);
```

توجه کنید اگر به‌طور مثال تابع F(5) را فراخوانی کنید کامپایلر دچار ابهام می‌شود که کدام یک از دو تابع را باید فراخوانی کند، بنابراین خطای کامپایلری رخ می‌دهد.

نکته: در هنگام تعریف توابعی که از آرگومان‌های پیش‌فرض استفاده می‌کنند به این نکته مهم توجه کنید که بعد از پارامترهای پیش‌فرض مجاز به آوردن پارامتر عادی نیستید.

مثال:
```
int   F(int  a = 0 , int b = 0  , int c);
int   T(float   x,  float y = 2.5 , double z);
```
دستورات فوق باعث بروز خطای کامپایلری می‌شوند، زیرا پارامترهای عادی c و z بعد از پارامترهای پیش‌فرض آمده‌اند.

۸ـ در تعاریف روبه رو کدام خط(ها) خطا دارد؟

```
void f1(int a,int b,int c=4,int d=7,int e=3); //1
void f2(int a,int b=2,int c=4,int d,int e=7); //2
void f3(int,int,int=4,int=5);                 //3
void f4(int,int=2,int,int);                   //4
```

۱) 1,2,3,4

۲) 2,3,4

۳) 2,4

۴) 1

❐ یافتن آدرس تابع سربارگذاری شده

در زبان C منظور از آدرس یک تابع، آدرس اولین دستورالعمل اجرایی تابع می‌باشد. جهت به‌دست آوردن آدرس یک تابع کافی است نام تابع را بدون هیچ‌گونه پارامتر یا آرگومانی به یک متغیر اشاره‌گر نسبت دهید.

مثال: جهت به‌دست آوردن آدرس تابع fun به شکل زیر عمل می‌کنیم:
```
void   fun(int  a,  int b);
void   main( )
{
  int   *pr ;
  pr = fun ;
}
```
در زبان C++ قضیه یک مقدار متفاوت است و اگر بخواهید آدرس تابعی که سربارگذاری شده است را به‌دست آورید، نام آن به تنهایی کفایت نمی‌کند و بایستی نحوه معرفی اشاره‌گر بر اساس توابع سربارگذاری صورت گیرد. برای درک بهتر به مثال زیر توجه نماید.

مثال:
```
#include <iostream.h>
void  show(int  count);
void   show(int  count,  char ch);
void   main( )
{
  void (*fp1) (int);          // یک اشاره‌گر به تابعی که یک پارامتر دارد را درست می‌کند.
  void (*fp2) (int , char); // یک اشاره‌گر به تابعی که دو پارامتر دارد را درست می‌کند.
```

```
  fp1 = show ;          // آدرس شروع تابع سربارگذاری show با یک پارامتر در اشاره‌گر fp1 ریخته می‌شود.
  fp2 = show;           // آدرس شروع تابع سربارگذاری show با دو پارامتر در اشاره‌گر fp2 ریخته می‌شود.
  fp1(20) ;             // 20 عدد space در خروجی چاپ می‌شود.
  cout<< "\n" ;
  fp2(10, 'z') ;        // 10 عدد کاراکتر z در خروجی چاپ می‌شود.
  cout<< "\n";
  }
void   show(int count)
{
   for ( ; count ; count --)
        cout<< ' '  ;      // تابع show با یک پارامتر به تعداد count فضای خالی (Blank) در خروجی چاپ می‌کند.
}
void  show(int  count,  char  ch)
 {
  for ( ;count  ;  count -- )
         cout<< ch ;    // تابع show با دو پارامتر به تعداد count کاراکتر ch را در خروجی چاپ می‌کند.
 }
```

۹ـ خروجی برنامه روبه‌رو کدام است؟

```
void f(char);
void f(int , char);
main()
  {
   void (*fp1) (char)=f;
   void (*fp2) (int,char)=f;
   fp1('A');
   fp2(5,'B');
  }
void f(char ch){cout<<ch;}
void f(int a,char ch)
  {
    for(int i=0;i<a;cout<<ch,i++);
  }
```

۱) ABBBB

۲) AAAAAB

۳) ABBBBB

۴) ERROR

⊡ قالب[1]

یکی دیگر از جنبه‌های پلی‌مورفیسم ویژگی Template یا همان قابلیت **ژنریک**[2] می‌باشد. قالب مجموعه‌ای از کلاس‌ها یا توابع می‌باشد که با هر نوع داده‌ای می‌تواند کار کند. به‌عبارت دیگر به

1. Template
2. Generic

کمک Template و مفهوم ژنریک می‌توان کلاس‌ها و توابعی تعریف کرد که بـا سـایر داده‌هـا کـار کنند و دیگر نیازی به نوشتن مجدد کلاس یا تابع برای آنها نباشد.

❐ کلاس قالب[1]

جهت تعریف کلاس قالب یا همان کلاس ژنریک کافی است قبل از کلمه class عبارت template به صورت زیر آورده شود:

تعریف کلاس مفروض template <class T>

با توجه به تعریف بالا دیگر کلاس مفروض فقط محدود به انواع داده‌های تعریفی برنامه‌نویس نمـی‌باشـد و حتی می‌تواند نوع داده‌های دیگر مانند عدد صحیح، اعشاری و حرف .... را نیز بپذیرد. به‌عبارت دیگر با تغییر نوع داده T به‌راحتی می‌توان کلاس مفروض را با هر نوع داده دلخواهی بکار برد.

نکته: جهت تعـــریف توابع عضو خارج کلاس قالب بایسـتی از عبـارت template <class T> بـه همراه <T> نام کلاس مفروض، قبل از تعریف تابع عضو استفاده شود.

مثال: یک کلاس ژنریک به‌نام S تعریف کنید که عملیات درج و حذف در یک آرایه را با هر نـوع داده‌ای بتواند انجام دهد.

```
template   <class T>
class  S
{
T    Arr [25];
int  head ;
public :
   S( )  {head = 0;}
  void  Add(T data) ;        // تابع Add یک داده به آرایه اضافه می‌کند.
  T  Delete( ) ;             // تابع Delete یک داده از آرایه حذف می‌کند.
} ;

template  <class T>
void  S<T> :: Add(T data)
 {
    if (head == 25)  {cout<< "Array  is  Full! \n";
                      exit(1) ;  }
    Arr[head] = data;      // داده مفروض را در خانه head اُم آرایه Arr اضافه می‌کند.
    head ++;
}
template <class T>
T  S<T> :: Delete( )
  {
```

1. Template Class

```
    if (head == 0) {
    cout<< "Array  is Empty! \n";
    exit(1);
  }
   head --;
   return Arr[head] ;     // داده‌ای را که در خانه head اُم آرایه Arr قرار دارد را برمی‌گرداند.
}
```

نکته: برای تعریف اشیاء از کلاس‌های ژنریک لازم است نوع داده T از کلاس قالب حتماً مشخص شود.

مثال:

```
#include <iostream.h>
#include <stdlib.h>
void main( ) {
S<int> obl;
S<char> ob2;
ob1.Add(10);
ob1.Add(20);
cout<< "Class of integer: " <<ob1.Delete( ) << "and" << ob1.Delete( ) << "\n";
ob2.Add('A') ;
ob2.Add('B');
cout<<"Class of character: "<<ob2.Delete( ) << "and" << ob2.Delete( ) ;
}
```

خروجی:

```
Class of integer: 10 and 20
Class of character: A and B
```

با توجه به کد بالا عمل درج و حذف با دو نوع داده متفاوت integer و character بر روی کلاس ژنریک S انجام می‌گیرد و دیگر نیازی به تعریف دو کلاس مجزا برای هر یک از نوع داده‌ها نمی‌باشد.

۱۰ـ خروجی برنامه روبه رو کدام است؟

```
template <class X> class test{
  X data;
  public:
  test( ){data++;}
  setdata(X a){data=a;}
  void show( ){for(int i=0;i<=1;cout<<data<<i++);}
 };
main( )
  {
   test<char> t1;
   t1.setdata('S');
   t1.show( );
   test<int> t2;
   t2.setdata(9);
   t2.show( );
  }
```

۱) S0S19091

۲) S0S1290912

۳) 8408419091

۴) ERROR

❐ تابع قالب[1]

علاوه بر تعریف کلاس قالب در زبان C++ امکان تعریف تابع قالب (تابع ژنریک) نیز وجـود دارد. تابع قالب متشکل از مجموعه‌ای از توابع تحت یک نام مشترک می‌باشد به‌گونه‌ای کـه هـر یـک از توابع روی یک نوع داده پردازش انجام می‌دهند، همچنین برخلاف کلاس قالب نیازی بـه مشـخص کردن نوع داده T نمی‌باشد و نوع آرگومان‌های تابع در هنگام فراخوانی تابع تعیین‌کننده تابعی است که باید اجرا شود. فرم کلی تعریف یک تابع قالب به شکل زیر می‌باشد:

```
template <class T > T  نام تابع  (لیست پارامترهای تابع)
  {
    ⋮
  }
```

مثال:

```
#include <iostream.h>
template <class T> T Max(T x, T y)
  {
    if (x > y)
          return x;
    else
          return y;
  }
void main( )
  {
   int a=10 , b = 20;
   char ch1='a' , ch2 = 'A';
   cout<< "Max of number = "<<Max(a , b) << "\n";
   cout<< "Max of character ="<<Max(ch1 , ch2);
}
```

خروجی:

```
Max of number = 20
Max of character = a
```

توجه کنید که تابع Max یک تابع ژنریک بـوده کـه بـر اسـاس نـوع آرگومـان‌هـای ورودی خـود تشخیص می‌دهد که نوع داده T را char یا int در نظر بگیرد. به‌طور مثال در Max(a,b) بـه‌دلیـل اینکه متغیرهای a و b عدد صحیح هستند در نتیجه در تابع ژنریک Max به‌جای نوع داده T، نـوع داده int را جایگزین می‌کند.

---

1. Template Function

۱۱ـ در برنامه روبه‌رو اگر اعداد 1 , 2 , 3 وارد شوند خروجی کدام است؟

```
template<class  T>
T  maximum(T  value1,  T  value2 , T  value3)
{   T  max = value1;
if(value2>max)  max=value2;
if(value3>max)  max=value3;
return  max; }
void  main( )
{ int   int1,  int2,  int3;
  cin>>int1>>int2>>int3;
  cout<<maximum(int1, int2, int3);
}
```

۱) 1　　۲) 3　　۳) 1 2 3　　۴) 3 2 1

۱۲ـ در برنامه روبه‌رو، اگر از ورودی به‌ترتیب اعداد 8 6 4 وارد شود، خروجی کدام است؟

```
template <class T>
T  maximum(T value1, T  value2, T   value3)
{  T  max=value1;
if (value2>max)  max=value2;
            else   if(value3>max) max=value3;
 return  max;}
 main( )
 {
 int  int1,int2,int 3;
 cin>>int1>>int2>>int3;
 cout<<maximum(int1, int2, int3);
}
```

۱) 4　　۲) 6　　۳) 8　　۴) ERROR

۱۳ـ می‌خواهیم تابعی ژنریک به نام swap تعریف کنیم که بتوانـد محتویـات دو متغیـر ورودی خود را (از هر نوعی که باشند) با یکدیگر جابه‌جا کند. کدام تعریف درست است؟

۱)
```
template <class X> void swap(X  &a,  X &b)
{ X  t;   t=a;   a=b;   b=t;}
```

۲)
```
template  void  swap(X a,  X b)
{ X  t;   t=a;   a=b;   b=t;}
```

۳)
```
generic  void  swap(class X  &a ,  class  X  &b)
{ X  *t;   *t=a;   a=b;   b=*t;}
```

۴)
```
generic  <class X>  void  swap(&a ,  &b)
{ X  t;   t=a;   a=b;   b=t;}
```

نکته: در توابع قالب (ژنریک) به‌دلیل اینکه کامپایلر نوع داده به کار رفته در تابع ژنریک (نوع داده T) را بر اساس نوع آرگومان‌های تابع تعیین می‌کند بنابراین بایستی حتماً نوع آرگومان‌ها در هنگام فراخوانی توابع ژنریک با هم یکسان باشند در غیر این صورت باعث بروز خطای کامپایلری می‌شوند.

مثال:

```
Max(a,ch2) ;     // خطای کامپایلری
Max(b,ch1) ;     // خطای کامپایلری
```

بنابراین برای جلوگیری از خطای کامپایلری بایستی تابع ژنریک را طوری تعریف کنیم که بیش از یک نوع داده (T) را پشتیبانی کند.

مثال:

```
#include <iostream.h>
template <class T1 , class T2 > void show(T1 data1,T2 data2)
  {
   cout<< data1<< "and" << data2<< "\n";
  }
void main ( ) {
      int i = 10;
      char ch = 'A';
      float f = 5.7;
      char *str = "C++";
      show(ch , i);
      show(str , f);
      show(i , f);
}
```

خروجی:

```
A and 10
C++ and 5.7
10 and 5.7
```

تذکر: در زبان C++ توابع قالب (ژنریک) می‌توانند بیش از یک نوع داده داشته باشند مانند مثال بالا.

⊡ سربارگذاری عملگرها[1]

سربارگذاری یک عملگر شباهت زیادی به سربارگذاری یک تابع دارد با این تفاوت که سربارگذاری یک عملگر تنها مختص یک کلاس خاص می‌باشد. جهت سربارگذاری عملگرها نیاز به **تابع عملگر**[2]

1. Operators Overloading
2. Operator Function

می‌باشد. تابع عملگر می‌تواند تابع عضو یا تابع دوست کلاسی باشد که سربارگذاری عملگر برای آن کلاس انجام می‌گیرد. (توجه کنید تابع دوست اشاره‌گر this را ندارد)

فرم کلی یک تابع عملگر عضو به شکل زیر تعریف می‌شود:

(لیست آرگومان‌ها) عملگر سربارگذاری شده operator :: نام کلاس مقدار بازگشتی

{

⋮

}

مقدار بازگشتی تابع عملگر معمولاً از نوع کلاسی است که عملگر برای آن تعریف شده است البته می‌تواند از انواع داده‌ای دیگر نیز باشد. لیست آرگومان‌های تابع عملگر نیز برحسب نوع عملگر سربارگذاری شده تغییر می‌کند.

نکته: سربارگذاری عملگرها باید بر اساس موقعیت عملگر در جدول اولویت عملگرها و تعداد عملوندهای آن صورت گیرد. به‌طور مثال نمی‌توانید عملگر ++ را طوری سربارگذاری کنید که دو عملوند داشته باشد.

نکته: عملگرهای . ، *. ، : ? و :: در زبان C++ را نمی‌توانید سربارگذاری کنید.

نکته: تابع عملگر نمی‌تواند آرگومان پیش‌فرض داشته باشد.

❐ سربارگذاری عملگرهای دوتایی (باینری)[1]

جهت سربارگذاری عملگرهای دوتایی نیاز به تابع عملگریست که تنها یک پارامتر دارد. این پارامتر شیءای را که به عنوان عملوند سمت راست عملگر باینری می‌باشد را دریافت می‌کند و شیء سمت چپ عملگر نیز باعث فراخوانی تابع عملگر می‌شود و به‌طور ضمنی به وسیله اشاره‌گر this به آن ارسال می‌گردد.

مثال: می‌خواهیم با سربارگذاری عملگر + مختصات دو نقطه دکارتی را با هم جمع کنیم.

```
#include <iostream.h>
class  coord {
    int  x,y ;                                  // مقادیر مختصات
    public :
    coord( )  { x = 0 ;  y = 0; }               // تابع سازنده با مقادیر اولیه صفر
    coord(int i, int j) {x = i ; y = j;}        // تابع سازنده با مقادیر اولیه دلخواه
    void  get_xy(int  &i , int &j) {i = x ; j = y; }
    coord  operator + (coord ob);
```

1. Binary Operators

```
};
coord   coord  :: operator + (coord  ob)
   {
     coord  temp ;
      temp.x = x + ob.x ;
      temp.y = y + ob.y ;
      return temp ;
  }
void   main( )
{
   coord  o1(7, 7) , o2 (3 , 3) ,  o3;     // شیء o3 را با مقادیر اولیه صفر از کلاس coord تعریف می‌کند.
   int   x,y ;
    o3 = o1+o2;
    o3.get_xy(x,y);
    cout<<"(o1 + o2)X : "<<x<<" , Y: "<<y<<"\n";
 }
```

خروجی: `(o1 + o2) X : 10 , Y : 10`

تابع عملگر + operator به‌عنوان تابع عضو کلاس coord تعریف شده و مقدار برگشتی آن همان کلاس coord می‌باشد. بدین‌ترتیب عملگر + سربارگذاری شده و می‌تواند شیءهای o1 و o2 را با هم جمع کند. در نتیجه مختصات x ها و y های دو شیء با هم جمع می‌شوند. برای سربارگذاری عملگر ـ کافی است به‌جای عملگر + در هر جای کد مثال قبل عملگر ـ قرار دهید.

۱۴ـ در برنامه روبه‌رو، کلاس Rational ، شامل کدام نیست؟

```
class rational{
public:
  rational(int=0, int=1);
  rational(const rational&);
  void operator = (const rational&);
private:
int num, den;  };
void  rational  :: operator = (const rational& r)
{  num=r.num; den=r.num;   }
void  main( )
{   }
```

۱) عملگر دسترسی　　۲) سازنده کپی

۳) سازنده پیش‌فرض　　۴) عملگر سربارگذاری شده جایگزینی

نکته: سربارگذاری عملگرها بر روی یک کلاس می‌تواند طوری انجام شود که عملوند سمت راست یک نوع داده باشد نه یک شیء به‌طور مثال نوع int. دقت کنید حتماً نوع داده باید در سمت راست عملگر باشد.

مثال: می‌خواهیم برنامه‌ای بنویسیم که به کمک سربارگذاری عملگر + یک مقدار صحیح را با یک شیء از کلاس coord جمع کند.

```
#include <iostream.h>
class   coord {
      int  x,y;
public :
      coord ( )  {x=0 ; y = 0;}
      coord (int  i,  int  j)  {x=i ; y = j;}
      void  get_xy (int &i , int &j)  {i=x;  j=y;}
      coord  operator + (int i);
 };
coord   coord :: operator + (int  i)
{
    coord   temp;
    temp.x = x+i;
    temp.y =y+i;
     return temp;
  }
void   main( ) {
coord  o1(7, 7) ,  o2 (3, 3) , o3 ;
int  x,y ;
o3 = o1 + 100 ;          // عدد صحیح + شیء
o3.get_xy(x, y) ;        // عدد صحیح + شیء
cout<< "(o1 + 100) X : "<<x<< " , Y: " <<y<< "\n";}
```

خروجی:

```
(o1 + 100) X : 107 ,  Y : 107
```

توجه: در سربارگذاری عملگرهایی که روی یک شیء و یک نوع داده صورت می‌گیرد، نوع داده (در این مثال عدد صحیح 100) بایستی در سمت راست عملگر قرار گرفته و شیء ، سمت چپ عملگر سربارگذاری شده باشد. بنابراین کامپایلر عبارتی مانند o3=100+o1 را خطا می‌گیرد. برای رفع این خطا بایستی از تابع عملگر دوست استفاده شود. (به بحث تابع عملگر دوست در این فصل مراجعه شود)

نکته: در سربارگذاری عملگرها می‌توان از پارامتر رفرنس به عنوان پارامتر تابع عملگر استفاده نمود.

مثال:

```
coord  coord  :: operator + (coord &ob)
 {
   coord  temp ;
   temp.x = x + ob.x ;
   temp.y = y + ob.y;
   return   temp ;
}
```