مثال: کلاسی بنویسید که یک آرایه ۲۰ عنصری را از ورودی دریافت کرده و آنها را مرتب سازی نماید.

```
#include <iostream>
#include <conio.h>
const int n = 20;
class csort{
 int a[n];
public:
void read();
 void sort();
 void print();
};
void csort::read()
{
 for(int i=0; i<n; i++)
  cin>>a[i];
}
```

```
void csort::print()
{
 for(int i=0; i<n; i++)
  cout<<a[i]<<endl;
}
void csort::sort()
{
 int temp,i,j;
 for(i=n-1; i>0; i--)
  for(j=0; j<i-1; j++)
   if(a[j]<a[j+1])
   {
    temp = a[j];
    a[j]=a[j+1];
    a[j+1]=temp;
   }
}
void main(void)
{
 csort h;
 h.read();
 h.sort();
 h.print();
}
```

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 (حتیvoid)

سازنده‌های کلاس

اعضای داده ای کلاس را چگونه می توان مقدار دهی اولیه کرد

می توان یک تابع با نام دلخواه و بدون پارامتر ورودی تعریف کرد که متغیر ها را مقدار دهی اولیه کند (مثلا همه را صفر کند)

سازنده

- کلاس می تواند تابع عضو ویژه ای به نام سازنده داشته باشد
- تابع سازنده همنام کلاسی است که در آن تعریف شده است
- هنگام ایجاد اشیایی از کلاس به طور خودکار اجرا می شود

کلاسی بنویسید که دو متغیر a و b را از ورودی میخواند و دارای توابعی برای محاسبه توان و فاکتوریل می باشد

```
class cpowerfact
{
private:
 int a;
 int b;
public:
 cpowerfact();
 void read();
 int fact();
 int power();
private:
 int f;
 int p;
};
cpowerfact::cpowerfact()
{
 a=b=0;
 f=p=1;
}
void cpower::read()
{
 cout<<"please enter a,b=";
 cin>>a>>b;
}
int cpowerfact:fact()
{
 if(a==1||a==0)
  f=1;
 else
  for(int i=1; i<=a; i++)
   f=f*i;
 return f;
}
int cpowerfact::power()
{
 for(int i=1; i<=b; i++)
  p=p*a;
 return p;
}
void main(void)
{
 cpowerfact h;
 h.read();
 cout<<h.fact()<<h.power();
}
```

تعریف کلاسی برای نمایش یک مستطیل و عملیات مربوطه

```
#include<iostream.h>
class rectangle {
private:
int length,width;// اطلاعات حساس در بخش خصوصی ذکر میشوند
public:
rectangle(int l=10,int w=5) { length = l; width = w; }// متد سازنده
int get_length() { return length; }// متدی برای برگرداندن طول مستطیل
int get_width(){ return width; }// متدی برای برگرداندن عرض مستطیل
int area() { return length*width; } // متد محاسبه مساحت مستطیل
};
void main(){
rectangle r1(17,6); // ایجاد یک شئ مستطیل با طول ۱۷ و عرض ٦
rectangle r2 ;// ایجاد یک مستطیل با مقادیر طول و عرض پیش فرض
cout<<r1.area()<<r2.area;
}
```

در ابتدای برنامه فوق یک کلاس تعریف شده و در تابع اصلی از این کلاس نمونه برداری شده و شئ۲ از روی آن ساخته شده است. شئ r۱ یک مستطیل با طول و عرض ۱۷ و ٦ میباشد و شئ r۲ دارای طول و عرض پیش فرض کلاس خواهد بود(چون برنامه نویس در اینجا طول و عرض را قید نکرده است). بنابراین اگر شئ r۳ را بصورت rectangle r3(9) تعریف نماییم طول آن برابر ۹ میشود اما عرض آن مقدار پیش فرض ٥ خواهد بود. سپس تابع عضو یا همان متد از کلاس برای چاپ مشخصات مستطیل و متد area() برای محاسبه مساحت آن فراخوانی گردیده اند.

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 .

مخرب ها

- یک تابع عضو ویژه از کلاس است
- همنام با کلاس است و با یک کاراکتر ~ شروع می شود
- هنگامی که یک شی از بین می رود به صورت خودکار فراخوانی می شود
- هنگام پایان یافتن عمر متغیرهای محلی در انتهای تابع
- هنگام بکار گرفتن عملگر delete برای از بین بردن یک متغیر با فضای پویا
- نوشتن مخرب برای کلاس زمانی که برخی اعضای داده ای آن حافظه پویا دارند، ضروری است

افزودن یک نابودکننده به کلاس

```
class Ratio
{ public:
Ratio() { cout << "OBJECT IS BORN.\n"; }
~Ratio() { cout << "OBJECT DIES.\n"; }
private:
int num, den;
};
int main()
{  { Ratio x;   // beginning of scope for x
cout << "Now x is alive.\n";
}                  // end of scope for x
cout << "Now between blocks.\n";
{ Ratio y;
cout << "Now y is alive.\n";
}
}
```

خروجی

```
OBJECT IS BORN.
Now x is alive.
OBJECT DIES.
Now between blocks.
OBJECT IS BORN.
Now y is alive.
OBJECT DIES.
```

اشاره گر this (this pointer)

توابع عضو هر شی به یک اشاره گر جادویی به نام This دسترسی دارند که به خود شی اشاره میکند.

معنای وجود این متغیر در کلاس یعنی هر شی آدرس خود در حافظه را میداند و یا تعبیر دیگر اینکه هر شی خودش را میبیند و خودش را میشناسد.

یکی از پرکاربردترین کاربردها زمانی است که میخواهیم خروجی یک تابع عضو خود شی و یا آدرس خود شی باشد.

کاربرد دیگر زمانی است که میخواهیم خود شی را بعنوان آرگومان یک تابع ارسال نماییم.

فرض کنید تابعی داریم که یک شی را گرفته و اطلاعات آنرا نمایش دهد. ما میخواهیم همین تابع نمایش دادن بصورت یک تابع عضو کلاس نیز باشد. برای استفاده از تابع موجود میتوانیم از اشاره گر This استفاده نماییم.

```
class where
 {
   private:
     char charArr[10];
   public:
      void reveal()
      {
            cout<< "\nMy object's address is:"<<this;
      }
};
void main()
{
            where w1,w2;
            w1. reveal();
            w2. reveal();
            cout <<"w1 Address is "<<&w1;
            cout <<"w2 Address is "<<&w2;
}
```

```
My objects address is:0012FF74
My objects address is:0012FF68
w1 Address is 0012FF74
w2 Address is 0012FF68
```