بسم الله الرحمن الرحیم

جزوه درس:

برنامه نویسی پیشرفته

نویسنده : علی نظری زاده

دانشگاه آزاد اسلامی واحد کرمانشاه

بهمن ماه سال 1394

علی نظری زاده

فصل اول: مقدمه ای بر برنامه نویسی

در تمامی زبان های برنامه نویسی باید نوع متغییرها را مشخص کرد و آن ها را با یکی از روش های زیر اعلان کرد

1- int : اگر بخواهیم در برنامه خود از اعداد صحیح مانند 26 یا 856 یا ... استفاده کنیم، باید متغییر خود را به این صورت اعلان کنیم. مانند روبه رو: ;int a یعنی این که متغییر a از نوع عدد صحیح می باشد و فقط می تواند اعداد صحیح را به خود اختصاص دهد. مثلا اگر مقدار a را مساوی با عدد 56.8 قرار بدهیم با خطا مواجه می شویم و برنامه اجرا نمی شود.

2- floa : اگر بخواهیم در برنامه خود از اعداد اعشاری مانند 83.6 یا 95.4 یا ... استفاده کنیم، باید متغییر خود را به این صورت اعلان کنیم. مانند روبه رو: ;float a یعنی این که متغییر a از نوع عدد اعشاری می باشد.

3- double : این نوع داده ای نیز برای اعداد اعشاری مورد استفاده قرار می گیرد با این تفاوت که دقت اندازه گیری آن بیش تر می باشد و اعداد بیش تری را بعد از ممیز نشان می دهد. مانند روبه رو: ;double a.

4- bool : این نوع داده ای فقط شامل دو حالت (صحیح : true و نادرست : false) می باشد و بیش تر برای زمانی از آن استفاده می کنیم که درستی یا نادرستی یک مقدار را بسنجیم. مانند روبه رو: ;bool a.

5- char : اگر بخواهیم در برنامه خود از کاراکتر (یک حرف یا یک علامت) استفاده کنیم، باید متغییر خود را به این صورت بیان کنیم. مانند روبه رو: ;char k یعنی این که متغییر k شامل یک کاراکتر یا یک حرف مانند: (a , #, Z, s) یا ... می باشد.

6- string : این نوع داده ای شامل مجموعه ای از کاراکترها که (رشته) نام دارد، می باشد. این روش بیش تر برای زمانی مورد استفاده قرار می گیرد که بخواهیم مثلا نام یا نام خانوادگی یا ... را از ورودی دریافت کنیم. به این نکته خوب دقت کنید که برای دریافت نام یا نام خانوادگی افراد به این صورت عمل می کنیم:

اعلان متغییر str از نوع رشته ای	String str;
دریافت رشته ی str با استفاده از دستور cin	cin>>str;
خروجی برنامه	cout<< " Ali ";

اما برای دریافت نام و نام خانوادگی با هم، از متدی به نام getline(cin, name) استفاده می کنیم. که به صورت زیر می باشد:

اعلان متغییر str از نوع رشته ای	string str;
دریافت رشته ی str با استفاده از دستور getline(cin, str)	getline(cin, str);
خروجی برنامه	cout<< " Ali Nazarizadeh "

همان طور که می بینید: cin فقط توانایی دریافت یک کلمه را دارد و هنگامی که به فضای خالی بین کلمات برسد، فقط اولین کلمه را از ورودی دریافت می کند و توانایی دریافت بقیه کلمات را ندارد. برای این حل این مشکل از تابع getline(cin, name) استفاده می کنیم که این تابع توانایی گرفتن رشته ها را از ورودی دارد. این تابع دارای دو پارامتر می باشد که پارامتر اول شی cin می باشد وآن را همان جوری می نویسم و در پارامتر دوم، اسم رشته مورد نظر را می نویسیم که در این جا اسم رشته، str می باشد.

نکته 1-1 : از دستور cin برای گرفتن داده ها و از دستور cout برای نمایش داده ها استفاده می کنیم.

نکته 1-2 : در ابتدای هر برنامه ای که به زبان C++ می نویسیم حتما باید این دو فایل کتابخانه ای را به اول برنامه اضافه کنیم:

```
#include<iostream.h>        فایل کتابخانه ای مربوط به cin و cout
#include<conio.h>           فایل کتابخانه ای مربوط به به تابع getch()
```


ساختار یک برنامه به زبان C++ به صورت زیر می باشد:

```
#include<iostream.h>
#include<conio.h>
int main()
{
        … … …;
        … … …;          پیاده سازی دستورات و نوشتن برنامه در داخل تابع main()
        … … …;
        getch();        این تابع، صفحه نمایش را نگه می دارد تا کاربر داده های خود را در خروجی مشاهده کند
        return 0;       این تابع مقدار صفر را به برنامه بازگشت می دهد تا مطمعن شود که برنامه درست کار می کند
}
```

1-1 حلقه های تکرار و دستورات شرطی

در برنامه نویسی از چند ترفند خاص برای نوشتن برنامه ها استفاده می کنیم که پایه و اساس آن ها، حلقه ها و دستورات شرطی می باشند. در زبان C++ چهار نوع حلقه تکرار یا شرطی وجود دارد که عبارت اند از:

1- دستور if : کلمه ی if به معنی (اگر) می باشد و بیش تر برای شرط های دستوری از آن استفاده می شود. بنابراین هرگاه که ما نیاز به برقراری یک شرط در برنامه داشتیم یا در صورت سوال کلمه ی (اگر) آمد، از این دستور استفاده می کنیم. این دستور به دو روش مورد استفاده قرار می گیرد.روش اول به صورت زیر می باشد:

```
if ( شرط برنامه )

{

        اگر شرط برنامه درست بود، دستورات این قسمت را انجام بده
}
else        در غیر این صورت
{
        دستورات مربوط به این قسمت را انجام بده
}
else
{
… … …;
}
```

روش دوم به صورت زیر می آید:

```
if( شرط اول برنامه )
{
        اگر شرط اول برنامه درست بود، دستورات این قسمت را انجام بده
}
else if ( شرط دوم برنامه )
{
        گر شرط دوم برنامه درست بود، دستورات این قسمت را انجام بده
}
else if ( شرط سوم برنامه )
{
        گر شرط سوم برنامه درست بود، دستورات این قسمت را انجام بده
}
… … …;
… … …;
```


مثال 1-1 : برنامه ای بنویسید که عددی را از ورودی دریافت کند و اگر یک رقمی بود در خروجی عدد 1 را چاپ کند، اگر دو رقمی بود عدد 2 را چاپ کند، اگر سه رقمی بود عدد 3 را چاپ کند و الی آخر.

```
#include<iostream.h>
#include<conio.h>
int main()
{
        int a;
        cout << "enter a number: ";
        cin >> a;
        if (a <= 9)
        {
                cout << "1";
        }
        else if (a <= 99)
        {
                cout << "2";
        }
        else if (a <= 999)
        {
                cout << "3";
        }
        else if (a <= 9999)
```

```
        {
                cout << "4";
        }
        getch();
        return 0;
}
```

توضیح برنامه : اعداد یک رقمی کوچک تر مساوی عدد 9 می باشند، بنابراین در دستور if اولی این شرط را قرار داده ایم که اگر عدد ورودی کوچک تر یا مساوی عدد 9 بود، پیغامی مبتنی بر این که عدد یک رقمی می باشد را چاپ کند ولی اگر اگر عدد ورودی کوچک تر یا مساوی عدد 9 نبود برنامه به دستور if بعدی می رود و در دستور بعدی چک می کند که اگر عدد ورودی کوچک تر مساوی عدد 99 بود، یک پیغام مبتی بر این که عدد دو رقمی می باشد را چاپ کند و به همین صورت برنامه شرط های بعدی را تک به تک چک می کند.

2- حلقه ی for : کلمه ی for به معنی (برای) می باشد و از این حلقه برای انجام عملیات شمارشی استفاده می کنیم. هرگاه در برنامه ای خواستیم مثلا اعداد بین 100 تا 200 یا ... شمارش کنیم و آن ها را در خروجی نمایش دهیم یا این که خواستیم از یک محل تا محل بعد را پیمایش کنیم، از این حلقه استفاده می کنیم که به صورت زیر نوشته می شود:

for (مبدا ; مقصد ; گام حرکت)

مثال 1-2 : برنامه ای بنویسید که اعداد بین 16 تا 180 را در خروجی نمایش دهد.

```
#include<iostream.h>
#include<conio.h>
int main()
{
        int i;                          اعلان متغییر i از نوع عدد صحیح
        for (i = 16; i <= 180; i++)     حلقه ی for
        {
                cout << i << " ";       متغییر i را در خروجی چاپ می کند
        }
        getch();
        return 0;
}
```

توضیح برنامه : همان طور که در قبل گفتیم، برای شمارش اعداد از حلقه ی for استفاده می کنیم. در این مثال گفته شده است که اعداد بین 16 تا 180 را چاپ کند، بنابراین یک حلقه ی for می نویسیم که مثلا یک متغییر مانند i را مساوی با عدد مبدا که همان عدد 16 می باشد قرار می دهیم و عدد مقصد را نیز مساوی با عدد 180 قرار می دهیم و مقدار i را یک واحد یک واحد اضافه می کنیم تا همه ی اعداد را پیمایش کند و هر بار هم که عددی را شمارش کند، آن را با استفاده از دستور خروجی cout در خروجی چاپ می کند.

3- حلقه ی while : کلمه ی while به معنی (تا زمانی که) می باشد و تقریبا اغلب قابلیت های دستور if و حلقه ی for را دارد که به صورت زیر نوشته می شود:

```
while ( تا زمانی که این شرط برقرار است )
{
        دستورات این قسمت را انجام بده
}
```


مثال 1-3 : برنامه ای بنویسید که تعدادی عدد را از ورودی دریافت کند و تا زمانی که عدد دریافتی بزرگ تر از صفر بود، حاصل ضرب عدد در خودش را در خروجی نمایش دهد و برنامه به کار خود ادامه دهد. اما اگر عدد دریافتی کوچک تر از صفر بود، برنامه به پایان برسد.

```
#include<iostream.h>
#include<conio.h>
int main()
{
        int i = 0;
        while (i >= 0)
        {
                cout << "enter a number: ";
                cin >> i;
                cout << i *i << endl;
        }
        getch();
        return 0;
}
```

توضیح برنامه: در ابتدا متغییر i را از نوع عدد صحیح معرفی کرده ایم و در ابتدا مقدار صفر را به آن نسبت داده ایم. دقت کنید که متغییر i همان عددی می باشد که قرار است از ورودی دریافت کنیم. در داخل حلقه ی while عبارت i>=0 نوشته شده است، این بدین معنی می باشد که تا زمانی که عدد دریافتی بزرگ تر مساوی صفر می باشد می توانیم عدد از ورودی دریافت کنیم و در آخرسر هم هر عددی را که از ورودی دریافت می کنیم ضرب در خودش می کنیم.

4- حلقه ی switch / case : کلمه switch به معنی (کلید یا کنترل کردن) و کلمه ی case به معنی (مورد) می باشد. این حلقه تقریبا توانایی انجام بیش تر قابلیت های دستور if را دارد. از این حلقه نیز می توان برای اینجام دستورات شرطی استفاده کرد. به این نکته دقت کنید که این حلقه فقط از نوع داده ای عدد صحیح (int) و کاراکتر (char) پشتیبانی می کند. یعنی این که برای کار با این دو نوع داده می توان از این حلقه ها استفاده کرد پس دیگر نمی توان مثلا رشته ها (string) یا اعداد اعشاری (float) را با استفاده از این حلقه انجام داد. که به صورت زیر نوشته می شود:

```
switch ( نوشتن متغییری که قرار است مورد بررسی قرار بگیرد )
{
case 1: cout << "one";          اگر شرط قسمت case درست بود، دستورات این خط را انجام بده
        break;                  در غیر این صورت
case 2: cout << "two";          اگر شرط قسمت case درست بود، دستورات ایت خط را انجام بده
        break;                  در غیر این صورت
```

```
        case 3: cout << "there";                    ... ... ...
                break;                              ... ... ...
        case 4: cout << "four";                     ... ... ...
                break;                              ... ... ...
        default: cout << "Erorr";          در صورت درست نبودن هیچ کدام از دستورات قسمت case
        }                                  دستور مربوط به قسمت default را انجام بده
```

مثال 1-4 : برنامه ای بنویسید که اعداد از یک تا چهار را از ورودی دریافت کند و معادل انگلیسی هر کدام را در خروجی نمایش دهد.

```
#include<iostream.h>
#include<conio.h>
int main()
{
        int n;
        cin >> n;
        switch (n)
        {
        case 1: cout << "one";
                break;
        case 2: cout << "two";
                break;
        case 3: cout << "there";
                break;
        case 4: cout << "four";
                break;
        default: cout << "Erorr";
        }
        getch();
        return 0;
}
```

توضیح برنامه : ابتدا متغییری را که قرار است در برنامه مورد بررسی قرار دهیم را در داخل دستور switch می نویسیم و با استفاده از دستورات case، هر قسمت را مورد بررسی قرار می دهیم. مثلا در اولین case عدد 1 را می نویسیم، یعنی این که اگر مورد نظر ما عدد 1 یا همان عددی را که وارد کرده ایم عدد 1 بود، در خروجی one را چاپ کن سپس با استفاده از دستور break برنامه را قطع می کنیم و اگر اولین case ما صحیح نبود به سراغ case بعدی برود و دستورات آن قسمت را انجام دهد. در قسمت بعدی نیز نوشته شده است که اگر عدد ورودی 2 بود، در خروجی two راچاپ کند. اما اگر هیچ کدام از case های ما درست نبود، دستور مربوط به قسمت پیش فرض برنامه یا همان default را انجام بده.

2-1 آرایه ها

در برنامه نویسی از آرایه ها استفاده های فراوانی می شود. در مثال های قبل ما فقط یک عدد را از ورودی دریافت کردیم اما اگر بخواهیم مثلا نمرات تعداد 100 دانشجو را دریافت کنیم چکار باید انجام داد؟ مسلما ما از 100 متغییر که هر کدام مربوط به یک عدد می باشد استفاده نمی کنیم، در این صورت است که از آرایه ها استفاده می کنیم.

نکته 1-3 : همیشه وقتی که از آرایه ها استفاده می کنیم، باید حتما یک حلقه ی for نیز در برنامه به کار ببریم. یعنی همیشه برای دریافت و نمایش داده ها از حلقه ی for استفاده می کنیم. مانند مثال زیر:

نکته 1-4 : برای نمایش داده ها در قالب یک آرایه، به صورت زیر عمل می کنیم:

نوع بازگشتی اسم آرایه [تعداد داده ها];

string	name	[3];	آرایه ای از نوع رشته ای که شامل 3 رشته می باشد
float	score	[3];	آرایه ای از نوع عدد صحیح که شامل 3 عدد یا خانه می باشد

مثال 1-5 : برنامه ای بنویسید که نام و شماره ی 3 دانش آموز را از ورودی دریافت کند و آن ها را در خروجی نمایش دهد.

```
#include<iostream.h>
#include<conio.h>
#include<string>
int main()
{
	string name[3];
	float score[3];
	int i;
	for (i = 0; i < 3; i++)
	{
		cout << "name " << i + 1 << ": ";
		cin >> name[i];
		cout << "score " << i + 1 << ": ";
		cin >> score[i];
	}
	for (i = 0; i < 3; i++)
	{
		cout << name[i] << " " << score[i] << endl;
	}
	getch();
	return 0;
}
```

توضیح برنامه : این برنامه یک آرایه به نام name دارد که از نوع رشته ای می باشد و تعداد خانه های آن 3 می باشد و هم چنین یک آرایه به نام score برای ذخیره نمره های دانشجویان دارد که به ترتیب نام و نمره هرکدام را از ورودی دریافت می کند و آن ها را در خروجی نشان می دهد.

3-1 تابع

در برنامه نویسی برای ساده تر کردن، خلاصه کردن و جلوگیری از نوشتن دستورات تکراری در برنامه و کاهش حجم برنامه از توابع استفاده می کنند. برای تعریف یک تابع نیز همانند تعریف متغییرها استفاده می کنیم. یعنی ابتدا نوع بازگشتی تابع و سپس نام تابع را می نویسیم. به صورت خیلی ساده تر می توان گفت که، راه تشخیص تابع این است که تابع همیشه دارای علامت پرانتز می باشد که در داخل آن یا پارامتر وجود دارد یا این که فاقد پارامتر می باشد. به طور کلی برای اعلان یک تابع، به صورت زیر عمل می کنیم:

نوع بازگشتی تابع نام تابع (پارامتر اول , پارامتر دوم , ...);

```
int        insert (int a, int b);
```

نکته 1-5 : علاوه بر 6 نوع بازگشتی که برای متغییرها داشتیم، یک نوع جدید بازگشتی به نام (void) نیز برای توابع به کار می بریم. این نوع داده ای هیچ مقداری را بازگشت نمی دهد.

نکته 1-6 : اگر نوع بازگشتی تابعی به صورت 6 حالت قبل باشد، یعنی این که غیر از void باشد، باید آن را با استفاده از دستور خروجی cout در خروجی بنویسیم. اما اگر نوع بازگشتی تابعی به صورت void باشد، نباید آن را در خروجی بنویسیم.

نکته 1-7 : برای توابعی که نوع بازگشتی آن ها جزء همان 6 نوعی که برای متغییرها بیان کردیم بود، باید در داخل متد از دستور بازگشتی return استفاده کنیم. اما اگر نوع بازگشتی ما void بود، دیگر نمی توانیم از دستور بازگشتی return استفاده کنیم و متغییری را بازگشت دهیم.

توابع نیز مانند سایر متغییرها، باید نوع آن را مشخص کرد. هر تابع یک نوع بازگشتی دارد که عبارت اند از:

1- یک مقدار صحیح (int) : در این حالت تابع اعداد صحیح را بازگشت می دهد.
2- یک مقدار اعشاری (float) : در این حالت تابع اعداد اعشاری را بازگشت می دهد.
3- یک مقدار اعشاری (double) : در این حالت تابع اعداد اعشاری را با دقت بالاتری بازگشت می دهد.
4- یک مقدار صحیح یا غلط (bool) : در این حالت تابع یا مقدار صحیح (true) یا این که مقدار غلط (false) را بازگشت می دهد.
5- یک مقدار کاراکتری (char) : در این حالت تابع کاراکترها را بازگشت می دهد.
6- یک مقدار رشته ای (string) : در این حالت تابع رشته ای را بازگشت می دهد.
7- بدون نوع بازگشتی (void) : در این حالت تابع هیچ مقداری را بازگشت نمی دهد.

مثال 1-6 : برنامه ای بنویسید که تاریخ تولد و تارخ امروز فردی را از ورودی دریافت کند و با استفاده از تابعی مشخص کند که چند روز از عمر این شخص گذشته است.

```
#include<iostream.h>
#include<conio.h>
int calculate(int year1, int month1, int day1, int year2, int month2, int day2)
{
        int y, m, d, result;
        y = year2 - year1;
        m = month2 - month1;
        if (m < 0)
        {
```

```
            m *= -1;
        }
        d = day2 - day1;
        if (d < 0)
        {
            d *= -1;
        }
        result = (y * 365) + (m * 30) + (d);
        return result;
}
int main()
{
        int year1, month1, day1, year2, month2, day2;
        cout << calculate(1374, 9, 15, 1394, 11, 21);
        getch()
        return 0;
}
```

توضیح برنامه : ما به 6 متغییر که سه تای آن ها نشان دهنده ی تاریخ تولد فرد براساس سال و ماه و روز و سه تای بعدی نشان دهنده ی تاریخ امروز بر اساس سال و ماه و روز می باشد نیاز داریم. برای محاسبه ی روزها، فقط کافی است که به ترتیب سال حال را از سال تولد و ماه حال را از ماه تولد و روز حال را از روز تولد کم کنیم. سپس تعدا سال هایی را که به دست می آید را در 365 و تعداد ماه هایی را که به دست می آید را در 30 و تعداد روز ها نیز با همان صورت می نویسیم و این سه متغییر را با هم جمع می کنیم تا تعداد روزهای عمر یک فرد را بدست آوریم. دقت کنید که برای متغییر m که نشان دهنده ی تعداد ماه ها و متغییر d که نشان دهنده ی تعداد روزها می باشد از یک دستور if استفاده کرده ایم. مثلا اگر ما ماه تولد را 8 و ماه حال را 3 وارد کنیم، طبق دستوری که نوشیتم
3-8=-5 می باشد بنابراین برای جلوگیری از این خطا چک می کنیم که اگر عدد به دست آمده از صفر کم تر باشد، آن را در 1- ضرب کنیم تا عدد را از منفی به مثبت تبدیل کنیم.

4-1 اشاره گرها

یکی از پرکاربرد ترین مطالب برنامه نویسی، اشاره گرها می باشد. اشاره گرها در واقع آدرس هایی از حافظه می باشند که به داده ها اشاره می کند. نشانه ی اشاره گره، کاراکتر (*) می باشد. مثلا یک اشاره گر را بدین صورت بیان می کنند: int *a;
از اشاره گرها استفاده های زیادی می شود که یکی از مهم ترین آن ها، استفاده از آرایه های پویا به وسیله ی اشاره گرها می باشد که نحوه استفاده از آن را در فصل بعدی خواهیم گفت.

فصل دوم: کلاس ها و اشیاء

در این فصل با اصول شی گرایی وبرنامه نویسی شی گرا آشنا خواهیم شد. تا الان و هر آن چه که در درس " مبانی کامپیوتر و برنامه سازی " خوانده ایم، مربوط به " برنامه نویسی ساخت یافته " می باشد. اما از این لحظه به بعد دیگر با این نوع برنامه نوشتن سروکار نداریم وکلا مباحث مربوط به شی گرایی و ارث بری را مورد بررسی قرار خواهیم داد. البته باید به این نکته دقت کرد که تمامی اصول برنامه نویسی و مقدمات پایه ای آن را در درس " مبانی کامپیوتر و برنامه نویسی " فرا گرفته ایم و در این قسمت با ساختار جدیدی از برنامه نویسی به نام " کلاس " آشنا خواهیم شد و تقریبا تمامی مطالبی که تا الان فرا گرفته ایم را به صورت ساختار و فرمی جدید در قالب یک " کلاس " پیاده سازی خواهیم کرد. دقت کنید که از این لحظه به بعد، به تابع، متد می گوییم. یعنی این که نام دیگر تابع، همان کلمه ی متد می باشد.

کلاس : به مجموعه ای ازمتغییرها و توابع که در یک فرم و ساختار کنار هم قرار می گیرند و با کمک هم دیگر یک برنامه بزرگ را طراحی می کنند کلاس می گویند.

شی : نمونه هایی از یک کلاس می باشد. یعنی این که ما ابتدا کلاس خود را طراحی و پیاده سازی می کنیم و سپس از آن کلاس، اشیایی را به وجود می آوریم.

هر کلاس دارای دو نوع داده ای می باشد که عبارت اند از:

1- عضو private : در این قسمت فقط متغییرها قرار می گیرند و هرمتغییری را که قرار باشد به برنامه اضافه کنیم باید آن را در قسمت private بنویسیم. تمامی متغییر هایی که در این قسمت قرار می گیرند به طور اتوماتیک به صورت عضو خصوصی در می آیند و این متغییرها فقط در داخل خود کلاس قابل دسترسی می باشند و خارج از کلاس، حق دسترسی به اعضای خصوصی private وجود ندارد.حتی در قسمت ()main.

2 -عضو public : تمام توابعی که قرار است در برنامه پیاده سازی کنیم باید آن ها را در این قسمت بنویسیم. بنابراین در این قسمت اکثرا توابع می آیند و به ندرت متغییری در این قسمت نوشته می شود. بنابراین تمامی توابع و متغییر هایی که در قسمت public می آیند، می توان آن ها را در قسمت ()main برنامه مورد استفاده قرار داد.

هر کلاس به طور کلی از سه قسمت اصلی تشگیل می شود که عبارت اند از:

1- قسمت مربوط به private و public که از این به بعد به آن " قسمت اول " می گوییم.
2- قسمت مربوط به پیاده سازی کلاس که در این مرحله فقط توابع موجود در قسمت public را پیاده سازی خواهیم کرد که از این به بعد به آن " قسمت دوم " می گوییم.
3- قسمت مربوط به ()main برنامه، که در این قسمت کلاسی را که طراحی کرده ایم مورد استفاده قرار می دهیم، بنابراین از این به بعد این مرحله را " قسمت سوم " می نامیم.

هر کلاس از یک قالب و فرم کلی پیروی می کند که به صورت زیر می باشد:

خط اول: مربوط به تابع cin وcout می باشد	#include<iostream.h>
خط دوم: مربوط به تابع ()getch می باشد	#include<conio.h>
خط سوم: اسم کلاس	class name
خط چهارم: آکولاد ({) باز می شود	{
خط پنجم: کلمه کلیدی private (قسمت اول)	private:
خط ششم: معرفی متغییرها	متغییر اول;
... ...	متغییر دوم;

```
public:                                  خط نهم: کلمه کلیدی public (قسمت اول)
متد اول;                                 خط هشتم: معرفی توابع
متد دوم;                                 ... ... ...
};                                       خط نهم: آکولاد به همراه یک سی می کالن بسته می شود. بدین صورت: ( ;{ )
متد اول                                  خط دهم: پیاده سازی توابع موجود در قسمت public (قسمت دوم)
{
 ... ... ...;
}
 متد دوم
{
 ... ... ...;
 }
int main()                               خط یازدهم: تابع ()main (قسمت سوم)
        ... ... ...;                     کدهای مربوط به تابع ()main
        ... ... ...;
        getch();                         خط دوازدهم: استفاده از تابع ()getch برای متوقف کردن صفحه خروجی
        return 0;                        خط سیزدهم: استفاده از تابع return 0 برای بازگشت دادن مقدار صفر
}
```

توضیح برنامه : در حالت کلی، ساختار اصلی یک کلاس به صورت کد قبل می باشد و تمامی کلاس هایی که قرار است طراحی کنیم، کلا از این قاعده پیروی می کنند.

خط اول: در این خط فایل کتابخانه ای مربوط به تابع cin و cout را می نویسیم.
خط دوم : در این خط فایل مربوط به تابع ()getch رامی نویسیم.
خط سوم : در این قسمت برنامه، اسم کلاس را مشخص می کنیم. برای این کار ابتدا کلمه ی کلیدی " class " را می نویسیم، سپس با یک فاصله، اسم کلاس خود را می نویسیم. مانند روبه رو: class book
خط چهارم : بعد از این که اسم کلاس خود را مشخص کردیم، یک آکولاد ({) باز می کنیم و بعد از این که آکولاد ({) را باز کردیم، شروع به نوشتن اولین قسمت برنامه که قبلا اسم آن را " قسمت اول " انتخاب کردیم، می کنیم.
خط پنجم : کلمه ی کلیدی private را می نویسیم.
خط ششم : در این قسمت، متغییرهایی که قرار است در برنامه مورد استفاده قرار دهیم را می نویسیم.
خط هفتم : کلمه ی کلیدی public را می نویسیم.
خط هشتم : در این قسمت توابعی را که قرا است طراحی کنیم را می نویسیم.
خط نهم: در این قسمت، آکولادی که قبلا باز کرده ایم را می بندیم، این بدین معنی می باشد که " قسمت اول " به پایان رسید.
خط دهم : توابع موجود در قسمت public را پیاده سازی می کنیم و کدهای مربوط به آن را در این قسمت می نویسیم.
خط یازدهم : تابع ()main را می نویسیم و یک آکولاد ({) باز می کنیم و در این قسمت کلاسی را که پیاده سازی کرده ایم مورد استفاده قرار می دهیم.
خط دوازدهم : در این قسمت تابع ()getch را می نویسیم که وظیفه این تابع این می باشد که وقتی برنامه را اجرا کردیم، صفحه خروجی را متوقف می کند تا ما از برنامه خود استفاده کنیم.
خط سیزدهم : و در مرحله ی آخر نیز تابع return 0 را می نویسیم. در مطالب قبل هم گفتیم که اگر تابعی نوع بازگشتی آن غیر از void باشد، باید با استفاده ز دستور بازگشتی return یک مقدار را بازگشت دهد. در این جا نیز چون تابع ()main دارای نوع بازگشتی int می باشد، بنابراین باید یک مقدار را بازگشت دهد. که همیشه در این تابع باید مقدار صفر بازگشت داده شود.

نتیجه گیری : هر کلاس از سه قسمت اصلی تشکیل شده است که آن ها را با نام های (قسمت اول و قسمت دوم و قسمت سوم) نام گذاری کردیم و مراحل نوشتن آن ثابت و به صورتی که در مثال قبل توضیح دادیم می باشد.

نکته 2-1 : همیشه در پایان private و public یا همان " قسمت اول " یک آکولاد بسته به همراه یک سی می کالن (;) بدین صورت می آید (;{). بنابراین با دیدن این شکل می فهمیم که قسمت اول برنامه به پایان رسیده است.

نکته 2-2 : در قسمت مربوط به پیاده سازی کلاس یا همان " قسمت دوم " تمامی توابعی را که در قسمت public نوشته ایم پیاده سازی می کنیم و کدها و دستورات هر کدام از آن ها را در داخل خود همان تابع می نویسیم.

نکته 2-3 : دستور int n; بدین معنی می باشد که متغییر n از نوع عدد صحیح یا همان int می باشد. اما در برنامه نویسی شی گرا در داخل ()main یا همان " قسمت سوم " تا حد امکان چنین کدی نمی نویسیم بلکه در این حالت از کلاس خود یک شی ایجاد می کنیم. اگر اسم کلاس خو را number گذاشته باشیم، در این حالت خواهیم داشت:

number ob1;

این دستور بدین معنی می باشد که ob1 از نوع number می باشد بنابراین چنین مفهومی را بیان می کند که : کلاس number یک شی از نوع ob1 دارد.

نکته 2-4 : همیشه در ()main یا همان " قسمت سوم " نحوه کد نویسی به صورت زیر می باشد:

1- در ابتدا تابع ()main را می نویسیم.
2- یک آکولاد ({) باز می کنیم.
3- اسم کلاس را می نوییسیم.
4- بعد از اسم کلاس، یک شی ایجاد می کنیم.
5- سپس باید با استفاده از شی ایجاد شده به تک تک متدهای قسمت public دسترسی داشته باشیم. بدین صورت که ابتدا شی را می نویسیم، سپس کاراکتر نقطه (.) را جلوی آن قرار می دهیم و در آخر اسم متد را می نویسیم. و به همین ترتیب نحوه ی دسترسی را برای هر متد به طور جداگانه و در یک خط مجزا می نویسیم.

ابتدا با یک مثال ساده شروع می کنیم. قبلا در درس " مبانی کامپیوتر و برنامه سازی " همچین برنامه ای داشتیم:

مثال 2-1 : برنامه ای بنویسید که عددی را از ورودی دریافت کرده و آن را در خروجی نمایش دهد.

روش اول (بدون پیاده سازی کلاس):

```
#include<iostream.h>
#include<conio.h>
int main()
{
        int n;                          اعلان متغییر n از نوع عدد صحیح
        cout << "enter n: ";            یک پیغام چاپ می کند
        cin >> n;                       دریافت متغییر n
        cout << n;                      متغییر n را به خروجی می برد
        getch();
        return 0;
}
```

حال می خواهیم مثال قبل را با استفاده از کلاس ها پیاده سازی کنیم. در این حالت باید کلاس را با استفاده از ساختار و قالبی که قبلا برای آن مشخص کردیم پیاده سازی کنیم ودر نهایت کلاس ما به صورت زیر طراحی می شود:


```
#include<iostream.h>
#include<conio.h>
class number
{
private:
        int n;                          یک متغییر به نام n
public:
        void insert();          یک تابع برای دریافت داده ها
        void print();           یک تابع برای نمایش داده ها
};
void number::insert()           پیاده سازی متد insert()
{
        cout << "enter a number: ";
        cin >> n;
}
void number::print()            پیاده سازی متد print()
{
        cout << n;
}
int main()                                      ابتدا تابع main() نوشته می شود
{                                               یک آکولاد ( { ) باز می کنیم
        number ob1;     اسم کلاس را می نویسیم و سپس یک شی به نام ob1 به وجود می آوریم
        ob1.insert();           شی ob1 باید به همه ی متدهای قسمت public  با استفاده از
        ob1.print();            کاراکتر نقطه ( . ) دسترسی داشته باشد
        getch();
        return 0;
}
```

توضیح کلاس : در برنامه ی بالا کلاسی را طراحی کرد ه ایم که عددی را از ورودی دریافت می کند و در خروجی آن را نشان می دهد. در صفحه قبل این برنامه را بدون کلاس ها وبه صورت ساده طراحی کردیم.
ساختار این کلاس همانند توضیحلات قبل از یک قاعده ی خاص پیروی می کند که اگر به یاد داشته باشیم هر کلاسی از سه قسمت تشکیل شده است که کلاس بالا نیز از این قاعده پیروی می کند.
همیشه برای انجام دادن هر عملی در برنامه، ابتدا یک تابعی که کار مورد نظر را انجام دهد به وجود می آوریم و سپس کدهای مربوط به هر تابع را در قسمت پیاده سازی کلاس که همان " قسمت دوم " می باشد پیاده سازی می کنیم.
در این برنامه ابتدا یک متغییر را به وجود می آوریم و چون متغییرها همیشه در قسمت private می باشند، بنابراین متغییر n را نیز در قسمت private می نویسیم. در این برنامه گفته شده است که " عددی را از ورودی دریافت و آن را در خروجی نشان دهید " بنابراین نیاز به یک تابع داریم که عدد مورد نظر را از ورودی دریافت کند و تابع دیگری را نیز نیاز داریم که عدد مورد نظر را به خروجی ببرد. در این مثال تابع insert() وظیفه دریافت عدد و تابع

()print وظیفه چاپ عدد را بر عهده دارد. در داخل متد ()insert با استفاده از دستورcin متغییر n را از ورودی دریافت می کند و در داخل متد ()print با استفاده از دستور cout، متغییر n را به خروجی می برد.

نکته 2-5 : همیشه در " قسمت دوم "، پیاده سازی توابع به صورت کد بالا می باشد که این خط از این قاعده پیروی می کند:

1- ابتدا نوع بازگشتی تابع موجود را می نویسیم.
2- سپس اسم کلاس رامی نویسیم.
3- عملگر حوضه دید (::) را قرار می دهیم.
4- و در آخر خود تابع مورد نظر را که قرار است پیاده سازی کنیم، می نویسیم.
که به طور مثال تابع ()insert در قسمت پیاده سازی کلاس بدین صورت نوشته می شود: void number::insert()
که این تابع نیز مراحل چهارگانه بالا را به ترتیب رعایت کرده است.

نکته 2-6 : در همه ی کلاس ها چندین متد ثابت وجود دارد که تقریبا در همه ی کلاس هایی که قرار است پیاده سازی کنیم، این متدها وجود دارند و تقریبا می توان گفت که در همه ی کلاس ها این توابع ثابت می باشند.
بعضی از متدهایی که بطور رایج در اکثر برنامه ها با آن ها سروکار داریم و در پیاده سازی اغلب کلاس ها، این متدها نیز در کلاس ما وجود دارند عبارت اند از:

1- متد دریافت داده ها : وظیفه این متد، دریافت داده ها می باشد که اغلب این متدها با نام هایی از قبیل: ()set , ()insert , ()input و ... نام گذاری می شوند. این نوع متدها همیشه دارای نوع بازگشتی void می باشند و دارای دو نوع (پارامتر دار و بدون پارامتر) می باشند.
2- متد مربوط به انجام محاسبات : وظیفه این متد، انجام محاسبات بر روی داده ها می باشد. به طور مثال برای محاسبه ی مساحت یا محاسبه ی یک فرمول ریاضی یا ... مورد استفاده قرار می گیرد که اغلب این متدها با نام هایی از قبیل: ()convert , ()calculate و ... نام گذاری می شوند. این نوع متدها در اکثر برنامه ها دارای نوع بازگشتی void می باشند و فقط در بعضی مواقع خاص نوع بازگشتی آن ها int یا float یا ... می باشد و دارای دو نوع (پارامتر دار و بدون پارامتر) می باشند.
3- متد چاپ داده ها : این متد وظیفه نمایش داده ها در صفحه نمایش را برعهده دارد که اغلب این متدها با نام هایی از قبیل: ()get , ()show , ()print و ... نام گذاری می شوند. این نوع متدها اغلب دارای نوع بازگشتی void می باشند و فقط در بعضی از برنامه ها، نوع بازگشتی آن ها int یا float یا ... می باشد و همیشه این نوع متدها فاقد پارامتر می باشند.

نکته 2-7 : اگر هر متدی نوع بازگشتی آن int یا float یا ...یعنی نوع بازگشتی آن به جز (void) باشد، باید در داخل ()main آن متد را با استفاده از دستور cout در خروجی بنویسیم. مانند زیر:

```
int main()
{
number ob1;
cout << ob1.show();
}
```

نکته 2-8 : اگر هر متدی نوع بازگشتی آن int یا float یا ...یعنی نوع بازگشتی آن به جز void باشد، در داخل متد باید از دستور بازگشتی return استفاده کنیم و با استفاده از return یک مقدار را بازگشت دهیم.

مثال 2-2 : کلاسی طراحی کنید که طول و عرض مستطیلی را از ورودی دریافت کرده و محیط و مساحت آن را محاسبه کند و در خروجی نمایش دهد.

روش اول (بدون پیاده سازی کلاس):


```
#include<iostream.h>
#include<conio.h>
int main()
{
        float lenght, width, area, perime;
        cout << "enter lenght: ";
        cin >> lenght;
        cout << "enter width: ";
        cin >> width;
        area = lenght*width;
        perime = 2 * (lenght + width);
        cout << "area: " << area << endl;
        cout << "perime: " << perime;
        getch();
        return 0;
}
```

اما حال میخواهیم برنامه بالا را با استفاده از کلاس ها پیاده سازی کنیم که به صورت زیر طراحی می شود:

```
#include<iostream.h>
#include<conio.h>
class rectangle
{
private:
        float lenght, width, area, perime;
public:
        void insert();                  متد دریافت داده ها که insert() نام دارد
        void calculate();               متد مربوط به عملیات محاسباتی که calculate() نام دارد
        void print();                   متد مربوط به چاپ داده ها که print() نام دارد
};
void rectangle::insert()                پیاده سازی متد insert()
{
        cout << "enter lenght: ";
        cin >> lenght;
        cout << "enter width: ";
        cin >> width;
}
void rectangle::calculate()             پیاده سازی متد calculate()
{
```

```
        area = lenght*width;
        perime = 2 * (lenght + width);
}
void rectangle::print()                                  پیاده سازی متد ()print
{
        cout << "area: " << area << endl;
        cout << "perime: " << perime;
}
int main()                                               تابع ()main
{
        rectangle r1;
        r1.insert();
        r1.calculate();
        r1.print();
        getch();
        return 0;
}
```

توضیح کلاس : در این برنامه : طول = length و عرض = width و محیط = area و مساحت = perime می باشد. در قبل هم اشاره کردیم که هر کلاس دارای چند متد ثابت می باشد و در اکثر برنامه ها این متدها وجود دارند مانند : ()insert یا ()calculate یا ()print یا ... . ابتدا در برنامه نویسی هر مثالی که به ما داده شد، ابتدا باید متغییرها و متدهای آن را مشخص کرد. چون در این مثال گفته شده است که : طول و عرض مستطیل را از ورودی دریافت کند، بنابراین ابتدا باید دو متغییر به نام های length و width که نشان دهنده ی طول و عرض مستطیل هستند به وجود آوریم و در مطالب قبل هم گفتیم که : تمامی متغییرها در قسمت private قرار می گیرند، بنابراین باید این دو متغییر را در قسمت private قرار داد. علاوه بر این، در ادامه سوال گفته شده است که : طول و عرض مستطیل را محاسبه کند، بنابراین دو متغییر به نام های area و perime نیز به وجود می آوریم و آن ها را نیز در قسمت private می نویسیم و حاصل محیط و مساحت را در آن ها می ریزیم و در نهایت این دو متغییر را به خروجی می بریم. بعد از این که تعداد متغییرها در برنامه را مشخص کردیم و آن ها را در قسمت private قرار دادیم، حال باید بفهمیم که در این برنامه به چند متد نیاز داریم و قبلا هم اشاره کردیم که تمامی متدها باید در قسمت public نوشته شوند، بنابراین به هر اندازه که متد لازم داشته باشیم، آن ها را به قسمت public اضافه می کنیم و در قسمت پیاده سازی یا همان " قسمت دوم " تک تک متدها را تعریف و کدهای مربوط به هرکدام را می نویسیم و در مرحله ی آخر که تابع ()main یا همان " قسمت سوم " می باشد، کلاس خود را مورداستفاده قرار می دهیم وبا استفاده از شی که درست کردیم، تک تک متدهای قسمت public را مورد دسترسی قرار می دهیم. در این مثال گفته شده است که طول و عرض مستطیلی را از ورودی دریافت کند، بنابراین باید یک متد دریافت داده ها به نام ()insert برای آن طراحی کنیم که طول و عرض مستطیل را که به ترتیب length و width می باشد را از ورودی دریافت کند. در ادامه گفته شده است که محیط و مساحت آن را محاسبه کند، بنابراین ما نیاز به یک متد محاسبه به نام ()calculate داریم که ابتدا محیط را محاسبه و در متغییر area ریخته و سپس مساحت آن را نیز محاسبه و در متغییر perime بریزد. در پایان نیز گفته شده است که محیط و مساحت را در خروجی نمایش دهد، بنابراین یک متد نمایش داده ها به نام ()print طراحی می کنیم که محیط و مساحت را که به ترتیب در داخل متغییرهای area و perime هستند را در خروجی نمایش دهد. بنابراین در این برنامه ما به سه متد نیاز داریم.

نکته 2-9 : بررسی سه متد اصلی (متد دریافت و متد محاسبه و متد چاپ داده ها) :

1- اگر متد (دریافت داده ها) که همان set() , insert() , input() یا ...، به صورت بدون پارامتر تعریف شود در این صورت در قسمت پیاده سازی یا همان " قسمت دوم " در داخل خود متد از دستور cin و cout استفاده می کنیم. و همیشه در این نوع متدها باید متغییرها را با استفاده از دستور cin از صفحه کلید گرفت. همیشه نوع بازگشتی متد دریافت داده ها به صورت void می باشد.

2- اما اگر این متد به صورت پارامتر دار مشخص شد دیگر حق استفاده از دستور cin و cout را در این متد نداریم و باید تک تک متغییرهای قسمت private را با تک تک پارامترهای این متد با استفاده از کاراکتر (=) مساوی هم قرار داد. به عنوان مثال اگر در برنامه قبل متد insert() به صورت پارامتر دار بود، برنامه به صورت زیر تغییر می کرد. همیشه نوع بازگشتی متد دریافت داده ها به صورت void می باشد.

```
#include<iostream.h>
#include<conio.h>
class rectangle
{
private:
        float lenght, width, area, perime;
public:
        void insert(float a, float b);
};
void rectangle::insert(float a, float b)
{
        length=a;          به ترتیب تک تک متغییرهای قسمت private را مساوی با پارامترهای
        width=b;           داخل متد insert قرار می دهیم
}
… … …;
… … …;
… … …;
int main()
{
        rectangle r1;
        r1.insert(2,3);  چون متد insert دارای دو پارامتر می باشد بنابراین دو عدد را داخل آن قرار می دهیم
        r1.calculate();
        r1.print();
        getch();
        return 0;
}
```

توضیح کلاس : چون متد دریافت داده ها پارامتر دار می باشد، بنابراین دیگر در داخل خود متد insert() نمی توان از دستور cin و cout استفاده کرد فقط می توان عملیات نسبت دهی را انجام داد که به صورت مثال قبل می باشد. به هر تعداد که متد ما پارامتر داشته باشد، به همان تعداد نیز در قسمت main() در داخل متد insert عدد قرار می دهیم. مثلا در مثال قبل تابع insert دارای دو پارامتر از نوع عدد اعشاری بود، بنابراین در داخل main() نیز به همان تعداد پارامتر، یعنی 2 عدد را برای تابع insert() ارسال می کنیم که بدین صورت می باشد:

```
r1.insert(2,3);
```

در واقع این دو پارامتر به ترتیب نماینده متغییرهای length و width می باشند و فقط فرق این روش با روش قبل در این است که، به جای این که یک پیغام در صفحه نمایش چاپ شود و سپس ما داده ها را که در این جا طول و عرض می باشند با استفاده از صفحه کلید وارد کنیم، تنها با قرار دادن آن مقادیر به صورت پارامتر در داخل خود تابع، متغییرهای خود را مقدار دهی می کنیم.

3- متد (محسبه) یا همان ()calculate , ()convert یا ... اگر بدون پارامتر تعریف شود در داخل متد فقط محاسبات مورد نظر را انجام می دهیم و در این قسمت هیچ عملیات نسبت دهی انجام نمی دهیم مانند مثال قبل. این متد هم می تواند دارای نوع بازگشتی void و هم غیر void باشد اما خیلی کم پیش می آید که نوع بازگشتی آن غیر از void باشد، بنابراین اغلب نوع بازگشتی آن void می باشد.

4- اگر متد محاسبه، به صورت پارامتر دار تعریف کنیم، علاوه بر انجام محاسبات، عملیات نسبت دهی هم در آخر سر انجام می دهیم.

5- متد (چاپ داده ها) یا همان ()get , ()show , ()print یا ... هیچ گاه به صورت پارامتر دار نمی آید و تنها بدون پارامتر می باشد. این متد قانون و دستور خاصی ندارد و فقط جهت نمایش داده ها می باشد. اگر نوع بازگشتی این متد به صورت int یا float یا ... یعنی نوع بازگشتی آن به جز (void) باشد، باید در قسمت ()main آن را با استفاده از دستور cout در خروجی نوشت.

نکته 2-10 : اگر متدی نوع بازگشتی آن به صورت int یا float یا... یعنی نوع بازگشتی آن به جز (void) باشد، در قسمت پیاده سازی توابع که همان " قسمت دوم " می باشد همیشه با استفاده از دستور return یک مقدار را برگشت می دهد ولی اگر متد ما دارای نوع بازگشتی void بود دیگر هیچ مقداری را برگشت نمی دهد.

مثال 2-3 : کلاس rectangle که در زیر آمده است خلاصه ای از مطالبی که تا الان گفته ایم را در خود جای داده است بنابراین به آن دقت کنید.

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>                          کتابخانه مربوط به تابع exit(1) می باشد
class rectangle                             اسم کلاس rectangle می باشد
{
private:                                    متغییرهای قسمت private
        float lenght, width, area, perime;
public:                                     متدهای قسمت public
        void insert();
        void calculate();
        void print();
        void set_lenght(float a);
        void set_width(float b);
        float get_lenght();
        float get_width();
};
void rectangle::insert()                    پیاده سازی متد ()insert
{
        cout << "enter lenght: ";
```

```
        cin >> lenght;
        cout << "enter width: ";
        cin >> width;
}
void rectangle::calculate()                    پیاده سازی متد calculate()
{
        area = lenght*width;
        perime = 2 * (lenght + width);
}
void rectangle::print()                        پیاده سازی متد print()
{
        cout << "area: " << area << endl;
        cout << "perime: " << perime << endl;
}
void rectangle::set_lenght(float a)            پیاده سازی متد set_lenght(float a)
{
        if (a < 0)
        {
                cout << "Erorr!";
                exit(1);
        }
        lenght = a;
}
void rectangle::set_width(float b)             پیاده سازی متد set_width(float b)
{
        if (b < 0)
        {
                cout << "Erorr!";
                exit(1);
        }
        width = b;
}
float rectangle::get_lenght()                  پیاده سازی متد get_lenght()
{
        return lenght;
}
float rectangle::get_width()                   پیاده سازی متد get_width()
{
        return width;
}
int main()                                     پیاده سازی تابع main()
```

```
{
        rectangle r1;                                   شی r1 را به وجود می آوریم
        r1.insert();                    با استفاده از شی r1 باید به تک تک متد های قسمت public
        r1.calculate();                         با استفاده از کاراکتر نقطه ( . ) دسترسی پیدا کنیم
        r1.print();
        r1.set_lenght(6);
        r1.set_width(2);
        cout << r1.get_lenght() << endl;        چون نوع بازگشتی توابع get_lenght() و
        cout << r1.get_width();         get_width() به صورت float می باشد، بنابراین
        getch();                        آن ها را با استفاده از دستور cout در خروجی می نویسیم
        return 0;
}
```

توضیح کلاس : در این برنامه سعی کرده ایم که خلاصه ای از مطالبی را که تا الان خوانده ایم پیاده سازی کنیم. این برنامه، همان کلاس rectangle مثال قبل می باشد فقط در این مرحله آن را کمی پیشرفته تر کرده ایم و چند متد دیگر نیز به آن اضافه کرده ایم. چون در این مثال ازتابع exit(1) استفاده می کنیم بنابراین باید فایل کتابخانه ای آن که #include<stdlib.h> می باشد را به برنامه نیز اضافه کنیم. در این مثال علاوه بر سه متد قبلی که همان insert() و calculate() و print() می باشند، چهار متد دیگر نیز به برنامه اضافه شده اند که این چهار متد هم، جز همان متدهایی است که قبلا در آن ها را پیاده سازی کردیم. وظیفه این متدها به صورت زیر می باشد:

1- متد set_lenght(float a) : اگر به خاطر داشته باشید، این متد یکی از انواع (متدهای دریافت کننده داده ها) می باشد که کار همان متد insert() را انجام می دهد اما در این جا به صورت پارامتر دار آمده است و قبلا هم گفتیم که اگر (متدهای دریافت داده ها) به صورت پارامتر دار بیایند، در این صورت دیگر از دستور cin و cout در داخل آن ها استفاده نمی کنیم وفقط تک تک متغییرهای قسمت private را با استفاده از کاراکتر مساوی (=)، مساوی پارامترهای خود تابع قرار می دهیم و همیشه نوع بازگشتی این متدها به صورت void می باشد. معمولا در جلوی متد set، مقداری که قرا است برای آن فرستاده شود را مینویسیم که در اینجا در جلوی متد set، کلمه ی lenght نوشته شده است. این بدین معنی می باشد که متد set قرار است متغییر length که همان طول مستطیل می باشد را دریافت کند. اما علاوه برعملیات نسبت دهی، چند خط دیگر نیز به برنامه اضافه شده است که به صورت زیر می باشد:

```
if (a < 0)
        {
                cout << "Erorr";
                exit(1);
        }
```

این کد فقط چک می کند که اگر طول دریافتی کوچک تر از صفر بود، ابتدا یک پیغام " Erorr! " چاپ کند و سپس با استفاده از تابع exit(1) از برنامه خارج شود و یک کار دل به خواهی می باشد که اگر هم این کد را ننویسیم هیچ مشکلی پیش نمی آید و فقط برای بهتر شدن برنامه، این کد را نیز به آن اضافه کرده ایم.

2- متد set_width(float b) : این متد نیز دقیقا مانند متد set_lenght(float a) می باشد فقط با این تفاوت که این متد وظیفه دریافت طول مستطیل که همان متغییر width می باشد را دارد.

3- متد get_lenght() : این متد هم یکی از انواع (متدهای چاپ داده ها) می باشد که کار همان متد print() را انجام می دهد اما در این جا نوع بازگشتی آن به صورت float می باشد و هیچ گاه هم این متدها به صورت پارامتر دار

نمی باشند. معمولا در جلوی متد get، مقداری که قرار است بازگشت دهد را می نویسم که در این جا جلوی متد get، کلمه ی length نوشته شده است. این بدین معنی می باشد که متد get قرار است متغییر length را که همان طول مستطیل می باشد را در خروجی نمایش دهد و در مطالب قبل هم گفتیم که اگر نوع بازگشتی تابعی به صورت int یا float یا ... یعنی به جز void باشد، همیشه یک مقدار را با استفاده از دستور return برگشت می دهد و همیشه این نوع متدها، متغییرهایی که جلوی آن ها نوشته می شود را بازگشت می دهند که در این جا چون جلوی متد get کلمه ی length نوشته شده است بنابراین این متد باید با استفاده از دستور return مقدار length را برگشت دهد.
4- متد ()get_width : این متد نیز دقیقا مانند متد get_lenght می باشد فقط با این تفاوت که این متد وظیفه چاپ متغییر width را دارد و با استفاده از دستور return، مقدار width را برگشت می دهد.

1-2 سازنده ها و مخرب ها

گاهی اوقات شرایطی پیش می آید که ما می خواهیم یک متغییر در ابتدا، دارای یک مقدار ثابت باشد و اگر در برنامه آن را با استفاده از توابع مربوطه مقدار دهی نکنیم، هم چنان دارای یک مقدار باشد. از سازنده ها استفاده های زیادی می شود که در مثال های بعدی آن ها را در برنامه های خود مورد استفاده قرار می دهیم.

نکته 2-11 : راه تشخیص سازنده در تابع ()main این است که، سازنده همیشه دارای پرانتز () و پارامتر می باشد. در واقع سازنده یک شی پرانتز دار می باشد. بنابراین اگر در داخل تابع ()main یک شی داشتیم که دارای پرانتز بود یا در داخل پرانتز، پارامتر وجود داشت نتیجه می گیریم که آن شی، سازنده می باشد. به صورت زیر:

```
Int main()
{
Rectangle r1(3), r2(4,6);          شی r1 و r2 سازنده می باشند
... ... ...;                       شی r1 سازنده تک پارامتری و شی r2 سازنده دو پارامتری می باشد
... ... ...;
}
```

توضیح برنامه : در مطالب قبل گفتیم که همیشه در داخل تابع ()main ابتدا اسم کلاس را می نویسیم و جلوی اسم کلاس یک شی ایجاد می کنیم. اما در اینجا مشاهده می کنیم که اشیاء ما دارای پرانتز و چندین پارامتر می باشند، بنابراین با توجه به نکته قبل، نتیجه می گیریم که این اشیاء، سازنده هستند و اگر یک پارامتر داشتیم می گوییم که سازنده ما تک پارامتری می باشد و اگر دو پارامتر داشتیم می گوییم که سازنده ما دو پارامتری می باشد و الی آخر. البته به این نکته نیز توجه کنید که تمام اشیایی که ما در تابع ()main به وجود می آوریم، به طور پیش فرض خود برنامه یک سازنده بدون پارامتر برای آن در نظر می گیرد. بنابراین دیگر نیاری به نوشتن سازنده بدون پارامتر در برنامه نیست وخود C++ به طور پیش فرض این کار را انجام می دهد.

نکته 2-12 : هنگامی که در برنامه خود از سازنده استفاده کردیم باید یک تابع مخرب نیز به برنامه اضافه کنیم. تابع مخرب تابعی است که وظیفه پس دادن حافظه ای را دارد که توسط متد سازنده ایجاد شده است. بنابراین اگر متد سازنده ای را به کلاس اضافه کردیم، حتما باید یک متد مخرب را نیز به کلاس خود اضافه کنیم.
از متد سازنده برای مشخص کردن تعداد بعدهای یک بردار، تعداد ضلع ها ی یک مثلث چند وجهی، تعداد ضرایب یک معادله درجه دو و ... استفاده می شود.

نکته 2-13 : هم متد سازنده و هم متد مخرب هیچ نوع بازگشتی ندارند، حتی فاقد نوع بازگشتی void هم می باشند. و هر دو این متدها، هم نام با کلاس می باشند. مثلا اگر اسم کلاس ما vector باشد، متد سازنده این کلاس به صورت زیر در " قسمت اول " نوشته می شود : (با فرض این که سازنده ما تک پارامتری باشد).


```
public:
        vector(float a);
};
```

نکته 2-14 : متد سازنده و مخرب هم نام با کلاس می باشند ولی با این تفاوت که همیشه قبل از متد مخرب، کاراکتر (~) قرار می گیرد و متد مخرب هیچ گاه به صورت پارامتر دار نمی آید و همیشه فاقد پارامتر می باشد. همیشه برای چنین کلاس هایی، داخل متد مخرب در قسمت پیاده سازی کلاس یا همان " قسمت دوم " یا یک پیغام چاپ می کنیم که مثلا شی ما از بین رفته است یا این که مانند مثال زیر هیچ کدی در داخل آن نمی نویسیم و آن را خالی می گذاریم. مثلا اگر اسم کلاس ما vector باشد، متد مخرب این کلاس به صورت زیر در " قسمت اول " نوشته می شود:

```
public:
        ~vector();
};
```

نکته 2-15 : تا الان ما وارد مبحث اشاره گرها نشده ایم و کلاس هایی را که تا الان پیاده سازی کرده ایم، نسبتا ساده بوده اند. اگر برنامه ما بدون اشاره گر بود، در آن صورت کدهای مربوط به متدهای سازنده و مخرب بسیار ساده می باشند و در همه ی برنامه ها این کدها ثابت می باشند. کدهای مربوط به توابع سازنده و مخرب در قسمت پیاده سازی کلاس یا همان " قسمت دوم " به صورت زیر می باشد:

متد سازنده : در داخل این متد فقط عملیات نسبت دهی را با استفاده از کاراکتر(=) انجام می دهیم. به صورت زیر:
1- متد سازنده بدون پارامتر : طبق مطالب گفته شده قبلی، لازم نیست که متد سازنده بدون پارامتر را برای کلاس بنویسیم چون خود C++ به طور پیش فرض این کار را انجام می دهد ولی برای این که با کدهای مربوط به آن آشنا شویم، در مثال زیر از آن استفاده کرده ایم. در داخل این متد تمام متغییرهای قسمت private را مساوی با یک عدد دل خواه قرار می دهیم. مثلا به صورت زیر:

```
#include<iostream.h>
#include<conio.h>
class vector
{
private:
        float x, y;
public:
        vector();
};
vector::vector()
{
        x = y = 0;          تمام متغییرهای قسمت private را برابر با یک مقدار اولیه قرار می دهیم
}                           که در این جا برابر با صفر قرار داده شده است
```

2- متد سازنده تک پارامتری : در داخل این متد نیز عملیات نسبت دهی را انجام می دهیم که به دو صورت می باشد : یا این که مثل سازنده بدون پارامتر، تک تک متغییرها را برابر یک عدد دل خواه قرار می دهیم، یا این که تک تک این متغییرها را برابر پارامترهای خود متد سازنده قرار می دهیم. مثلا به صورت زیر:


```
#include<iostream.h>
#include<conio.h>
class vector
{
private:
        int x, y;
public:
        vector(float a);
};
vector::vector(float a)
{
        x = y = a;          تک تک متغییرهای قسمت private را برابر با پارامتر سازنده قرار می دهیم
}
```

در این صورت، اگر در داخل تابع main() هر مقداری را که برای سازنده بفرستیم، به متغییرهای x و y نیز نسبت داده می شود و این متغییرها نیز، همان مقداری را می گیرند که برای متد سازنده فرستاده ایم.

3- متد سازنده دو پارامتری : نحوه پیاده سازی این متد نیز دقیقا مانند متد تک پارامتر می باشد. مثلا به صورت زیر:

```
#include<iostream.h>
#include<conio.h>
class vector
{
private:
        int x, y;
public:
        vector(float a, float b);
};
vector::vector(float a, float b)
{
        x = a;
        y = b;
}
```

4- متد مخرب : همان طور که در مطالب قبل گفتیم، اگر اشاره گرنداشتیم پیاده سازی توابع ساده می باشند و در داخل متد مخرب نیز یا بک پیغام چاپ می کنیم یا این که این متد را خالی می گذاریم و در داخل آن چیزی نمی نویسیم.

نکته 2-16 : قبلا گفتیم که در تابع ()main باید به تک تک متدهای قسمت public با استفاده ازکاراکتر نقطه (.) دسترسی پیدا کنیم. اما در مطالب قبل راه تشخیص متد سازنده را یاد گرفتیم و در داخل متد ()main نیز، نیازی به دسترسی با استفاده از کاراکتر نقطه (.) نمی باشد. متد مخرب هم به هیچ عنوان در تابع ()main نمی تواند بیاید.

مثال 2-4 : کلاسی به نام vector برای کار با بردارها طراحی کنید که در این کلاس از سازنده ها و مخرب ها نیز استفاده شده باشد.

```
#include<iostream.h>
#include<conio.h>
class vector
{
private:
        float x, y;
public:
        vector();                       سازنده بدون پارامتر
        vector(float a);                سازنده تک پارامتری
        vector(float b, float c);       سازنده دو پارامتری
        ~vector();                      متد مخرب
        void insert();
        void calculate();
        void print();
};
void vector::insert()
{
        ... ... ... ;
}
void vector::calculate()
{
        ... ... ... ;
}
void vector::print()
{
        ... ... ... ;
}
vector::vector()                        پیاده سازی سازنده بدون پارامتر
{
        x = y = 0;
}
vector::vector(float a)                 پیاده سازی سازنده تک پارامتری
{
        x = y = a;
}
```

```
vector::vector(float b, float c)        پیاده سازی سازنده دو پارامتری
{
        x = 28;
        y = 12;
}
vector::~vector()                       پیاده سازی متد مخرب
{

}
int main()                              تابع main()
{
        vector v0, v1(2), v2(3, 6);
        v1.insert();
        v1.calculate();
        v1.print();
        getch();
        return 0;
}
```

توضیح کلاس : در این کلاس سه نوع سازنده (بدون پارامتر و تک پارامتری و دو پارامتری) به همراه یک متد مخرب را پیاده سازی کرده ایم. علاوه بر این، توابع دیگری مانند insert() و calculate() و print() نیز در برنامه وجود دارند. چون در صورت سوال گفته شده است که از سازنده ها استفاده کنید، بنابراین چند نوع سازنده برای آن می نویسیم که در این کلاس از سه سازنده (بدون پارامتر و تک پارامتر و دو پارامتر) استفاده کرده ایم. برای این که با هر سه نوع سازنده و روش های مربوط به پیاده سازی آن ها آشنا شوید، در این کلاس ما از چندین نوع سازنده استفاده کرده ایم. دستورات هر کدام از متدهای سازنده و متد مخرب و قواعد مربوط به هرکدام را نیز با استفاده از نکته های قبل (نکته 15 و 16) پیاده سازی کرده ایم.

2-2 اشاره گرها

از این مرحله به بعد کمی روش برنامه نویسی خود را تغییر می دهیم و با استفاده از اشاره گرها، کلاس های پیشرفته تری را پیاده سازی می کنیم. در کلاس های قبل، روش کار با متدهای سازنده و مخرب و ... را یاد گرفتیم اما چون از این مرحله به بعد، اشاره گرها نیز به برنامه ما اضافه خواهند شد، بنابراین کمی روش کار نسبت به قبل فرق می کند. به این نکته دقت کنید که اشاره گرها هم مانند سایر متغییرها، در قسمت private نوشته می شوند.

نکته 2-17 : در کل، تمامی متدهای سازنده و مخرب به 5 قسمت تقسیم می شوند که عبارت اند از:

1- متد سازنده (بدون اشاره گر) : این نوع متد را در مثال قبل مورد بررسی قرار دادیم و تمامی مطالب و نکته های آن را نوشتیم.
2- متد سازنده (با اشاره گر) : اگر در برنامه خود از اشاره گرها استفاده کردیم، بنابراین باید قواعد و دستورات مربوط به این نوع متد را پیاده سازی کنیم.
3- متد مخرب (بدون اشاره گر) : این نوع متد را در مثال قبل مورد بررسی قرار دادیم و تمامی مطالب و نکته های

آن را نوشتیم.
4- متد مخرب (با اشاره گر) : اگر در برنامه خود از اشاره گرها استفاده کردیم، بنابراین باید قواعد و دستورات مربوط به این نوع متد را پیاده سازی کنیم
5- متد سازنده کپی : این نوع سازنده نیز، همانند سازنده های دیگر، و تقریبا نحوه پیاده سازی آن مشابه سازنده های دیگر می باشد.

نکته 2-18 : به این نکته خوب دقت کنید که : هرگاه در برنامه خود از اشاره گر استفاده کردیم، باید یک متد سازنده کپی را نیز به کلاس خود اضافه کنیم ولی اگر اشاره گر نداشتیم، دیگر نیازی به پیاده سازی متد سازنده کپی نمی باشد.

نکته 2-19 : هرگاه در یک کلاس اشاره گر داشتیم، نحوه پیاده سازی متد سازنده و متد مخرب نیز تغییر خواهد کرد. البته در مثال های قبل هم مشاهده کردیم که، پیاده سازی متد سارنده و مخرب از یک چارچوب خاص پیروی می کند و در تمام برنامه ها به یک صورت پیاده سازی می شود. بنابراین هنگامی که اشاره گر داشتیم، پیاده سازی متد سازنده ومخرب نیز از یک قاعده خاص پیروی می کنند و در تمامی برنامه ها به این روش پیاده سازی می شوند.

نکته 2-20 : هنگامی که در برنامه خود از اشاره گر استفاده کردیم، بنابراین تمام داده های ما دیگر در داخل اشاره گر قرار می گیرند و هرگاه خواستیم که داده های خود را از ورودی دریافت یا آن ها را در خروجی نمایش یا این که بر روی آن ها عملیات محاسباتی انجام دهیم، بنابراین باید این عملیات را روی اشاره گرها انجام دهیم. بنابراین وقتی که در برنامه اشاره گر وجود داشت، دیگر با اشاره گرها سروکار داریم و تمام عملیات خود را بر روی اشاره گرها انجام می دهیم.

نکته 2-21 : پیاده سازی متد سازنده برای وقتی که اشاره گر داشتیم:

```
#include<iostream.h>
#include<conio.h>
class vector
{
private:
        int n;                          یک متغییر به نام n از نوع int
        float a;                        یک اشاره گر به نام a از نوع float
public:
        vector(int n1);                 متد سازنده تک پارامتری
}:
vector::vector(int n1)                  پیاده سازی متد سازنده
{
        a = new float[n1];              مرحله اول: استفاده از آرایه ی پویا
        if (!a)                         مرحله دوم: یک شرط if قرار می دهیم و در داخل آن
        {
                cout << "Erorr!";       یک پیغام "Erorr!" چاپ می کنیم و با استفاده از
                exit(1);                دستور exit(1) از برنامه خارج می شویم
        }
        n = n1;         مرحله سوم: متغییر داخل قسمت private را مساوی با پارامتر سازنده قرار می دهیم
        for (int i = 0; i < n; i++)     مرحله چهارم: یک حلقه for می نویسیم
```

```
        {
                a[i] = 0;                    مرحله پنجم: تمام داده ها را مساوی صفر قرار می دهیم
        }
}
```


در همه ی برنامه هایی که اشاره گر داریم، همیشه متد سازنده به صورت بالا می باشد و از همین قاعده پیروی می کند.

مرحله اول : در ابتدای کار از آرایه های پویا استفاده می کنیم که روش این کار به صورت زیر می باشد:

```
اسم اشاره گر   =   new کلمه ی کلیدی   نوع بازگشتی اشاره گر   [ پارامتر سازنده ];
a              =   new                float                   [n1];
```

مرحله دوم : در این مرحله یک شرط if قرار می دهیم و در داخل آن این کد را می نویسیم : (!a) این دستور بدین معنی می باشد که چک می کند اگر اشاره گر وجود نداشت، یک پیغام "Erorr!" چاپ کند و از برنامه خارج شود
مرحله سوم : متغییر داخل قسمت private را مساوی با پارامتر سازنده قرار می دهیم. n = n1;
مرحله چهارم : یک حلقه for می آوریم در داخل آن از صفر تا n-1 را می شماریم.
مرحله پنجم : تمام داده ها را مساوی با صفر قرار می دهیم. در نکته های قبل هم گفتیم که، تمامی داده ها در داخل اشاره گر قرار می گیرند.

3-2 متد سازنده کپی

هنگامی که در برنامه خود از اشاره گرها استفاده کردیم بنابراین باید یک متد سازنده کپی نیز برای کلاس خود طراحی کنیم. استفاده از سازنده کپی در کلاس ها به خاطر این می باشد که چون وقتی از اشاره گرها استفاده می کنیم به ناچار باید در متد سازنده از آرایه های پویا نیز استفاده کنیم و وظیفه آرایه های پویا این می باشد که به اندازه پارامتر داخل متد سازنده حافظه به وجود می آورند و این به وجود آوردن حافظه نیازمند کپی گرفتن بیت به بیت از تمامی داده ها می باشد اما در C++ به طور پیش فرض این نحوه کپی گرفتن وجود دارد اما به طور دقیق این کار را برای ما انجام نمی دهد بنابراین باید خود برنامه نویس یک متد کپی را به برنامه اضافه کند که نحوه کپی گرفتن پیش فرض C++ را لغو کند و یک روش کپی گرفتن جدید برای کلاس پیاده سازی کند که این متد نیز همانند متدهای سازنده در تمامی برنامه ها از یک قاعده خاص پیروی می کند و همیشه ساختار و چارچوب آن در تمامی برنامه ها یکسان می باشد. به صورت زیر پیاده سازی می شود:

برای اعلان سازنده کپی آن را بدین صورت می نویسیم:

```
اسم کلاس ( const کلمه ی کلیدی   اسم کلاس   & کاراکتر   یک شی )
vector   ( const                vector      &           ob )
```

```
#include<iostream.h>
#include<conio.h>
class vector
{
private:
```

```
        int n;
        float *a;
public:
        vector(const vector & ob);                          متد سازنده کپی
};
vector::vector(const vector & ob)                           پیاده سازی متد سازنده کپی
{
        a = new float[cob.n];                               مرحله اول: استفاده از آرایه پویا
        if (!a)                                             مرحله دوم: یک شرط if قرار می دهیم و در داخل آن
        {
                cout << "Erorr!";                           یک پیغام "Erorr!" چاپ می کنیم و با استفاده از دستور
                exit(1);                                    exit(1) از برنامه خارج می شویم
        }
        n = cob.n;      مرحله سوم: متغییر داخل private را مساوی با پارامتر داخل سازنده کپی قرار می دهیم
        for (int i = 0; i < ob.n; i++)                      مرحله چهارم: یک حلقه ی for می نویسیم
        {
                a[i] = cob.a[i];        مرحله پنجم: تمام داده ها را مساوی پارامتر داخل سازنده کپی قرار می دهیم
        }
}
```

مرحله اول : در ابتدای کار از آرایه های پویا استفاده می کنیم که روش این کار به صورت زیر می باشد:

[متغییرداخل private . پارامتر داخل سازنده کپی] نوع بازگشتی اشاره گر کلمه ی کلیدی new = اسم اشاره گر

```
a = new float [ob.n];
```

به این نکته دقت کنید که در پیاده سازی متد سازنده کپی، هر وقت که پارامتر داخل متد سازنده کپی را نوشتیم، باید بعد از آن کاراکتر نقطه (.) و سپس متغییر قسمت private را قرار بدهیم. مانند دستور بالا.

مرحله دوم : دقیقا مانند سازنده قبلی می باشد

مرحله سوم : متغییر داخل private را مساوی با پارامتر داخل سازنده کپی قرار می دهیم. البته به این نکته خوب دقت کنید که سازنده کپی یک تفاوت جزعی با سازنده معمولی دارد وآن این است که در داخل سازنده کپی همیشه در مرحله سوم بعد از پارمتر، دوباره با استفاده از کاراکتر نقطه (.) اسم متغییر داخل private می آید.به صورت زیر:

برای سازنده معمولی: n=n1;

برای سازنده کپی: n=cob.n;

مرحله چهارم : دقیقا مانند سازنده قبلی می باشد

مرحله پنجم : تمام داده ها را مساوی با پارامتر داخل سازنده قرار می دهیم. البته به این نکته خوب دقت کنید که سازنده کپی یک تفاوت جزعی با سازنده معمولی دارد وآن این است که در داخل سازنده کپی همیشه در مرحله چهارم بعد از اشاره گر، دوباره با استفاده از علامت نقطه (.) خود اشاره گر می آید.به صورت زیر:

برای سازنده معمولی: n=n1;

برای سازنده کپی: a[i]=ob.a[i];

نکته 22-2 : پیاده سازی متد مخرب برای وقتی که اشاره گر داشتیم:

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

```
#include<conio.h>
class vector
{
private:
        int n;                          یک متغییر به نام n
        float *a;           یک اشاره گر به نام a از نوع float
public:
        ~vector();                      متد مخرب
}:
vector::~vector()                       پیاده سازی متد مخرب
{
        delete[]a;          مرحله اول: از بین بردن اشاره گر a  با استفاده از دستور delete

 }
```

مرحله اول : از بین بردن اشاره گر a با استفاده از دستور delete. که به صورت زیر آن را پیاده سازی می کنیم:

```
delete کلمه ی کلیدی    []    اسم اشاره گر;
delete                 []    a;
```

نکته 2-23 : همیشه وقتی اشاره گر داشتیم، در داخل متدهای (دریافت داده ها و نمایش داده ها) از حلقه ی for استفاده می کنیم. این کار به خاطر این می باشد که چون تعداد داده های ما متفاوت می باشد، بنابراین در داخل حلقه ی for از صفر تا آخرین اندیس را به ترتیب شمارش می کنیم و داده ها را یا دریافت، یا نمایش می دهیم.

مثال 2-5 : کلاسی برای کار با بردارها در فضای n بعدی پیاده سازی کنید که تعداد بعدهای این بردار با استفاده از یک متد سازنده کنترل شود. (با استفاده از اشاره گرها)

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
class vector
{
private:
        int n;                          متغییر n همان تعداد بعدهای بردار ما می باشد
        float *a;                       یک اشاره گر به نام a از نوع float
public:
        vector(const vector & ob);      متد سازنده کپی
        vector(int n1);                 متد سازنده تک پارامتری
        ~vector();                      متد مخرب
        void set(float x, int index);   متد دریافت داده ها
        void show();                    متد نمایش داده ها
};
vector::vector(const vector & ob)       پیاده سازی متد سازنده کپی
```

```
{
        a = new float[ob.n];
        if (!a)
        {
                cout << "Erorr!";
                exit(1);
        }
        n = ob.n;
        for (int i = 0; i < ob.n; i++)
        {
                a[i] = ob.a[i];
        }
}
vector::vector(int n1)
```

پیاده سازی متد سازنده

```
{
        a = new float[n1];
        if (!a)
        {
                cout << "Erorr!";
                exit(1);
        }
        n = n1;
        for (int i = 0; i < n; i++)
        {
                a[i] = 0;
        }
}
vector::~vector()
```

پیاده سازی متد مخرب

```
{
        delete[]a;
}
void vector::show()
```

پیاده سازی متد show()

```
{
        for (int i = 0; i < n; i++)
        {
                cout << a[i] << " " << endl;
        }
}
void vector::set(float x, int index)
```

پیاده سازی متد set

```
{
        if (index < 0 || index >= n)
```

```
        {
                cout << "Erorr!";
                exit(1);
        }
        a[index] = x;
}
int main()                                              تابع main()
{
        vector v1(3);       یک سازنده تک پارامتری که پارامتر آن، نشان دهنده تعداد بعدهای یک بردار می باشد
        v1.set(3, 0);                                   قرار دادن مقدار 3 در اندیس شماره 0 با استفاده از متد set
        v1.set(8, 1);                                   قرار دادن مقدار 8 در اندیس شماره 1 با استفاده از متد set
        v1.set(5, 2);                                   قرار دادن مقدار 5 در اندیس شماره 2 با استفاده از متد set
        v1.show();                                      نشان دادن داده ها با استفاده از متد show()
        getch();
        return 0;
};
```

توضیح کلاس : این کلاس تقریبا همان کلاس مثال قبل می باشد با این تفاوت که در این جا از اشاره گرها استفاده کرده ایم. چون در این کلاس از اشاره گرها استفاده کرده ایم، بنابراین باید سازنده ها و مخرب های مربوط به اشاره گرها را پیاده سازی کنیم وهم چنین باید یک متد سازنده کپی نیز پیاده سازی کنیم. قواعد مربوط به هرکدام از آن ها را در نکته های قبل مورد بررسی قرار دادیم و طبق همان قواعد و دستورها، کدها را پیاده سازی می کنیم.
علاوه بر متدهای سازنده و مخرب، چندین متد دیگر هم در کلاس وجود دارند که عبارت اند از:
متد set(float x, int index) : این متد جز متدهای دریافت داده ها می باشد و فقط وظیفه دریافت داده ها را برعهده دارد.طبق مطالب قبل، چون این متد دارای پارامتر می باشد، بنابراین در داخل متد، از دستور cin و cout استفاده نمی کنیم، یعنی این که داده ها را با استفاده از صفحه کلید وارد نمی کنیم بلکه در قسمت main() داده ها را برای این متد می فرستیم. متد set بدین صورت عمل می کند:
این متد دارای دو پارامتر می باشد. پارامتر اول که x نام دادر وظیفه گرفتن اعداد را دارد و پارامتر دوم که index نام دارد متغییرهای x را در خود جای می دهد. بنابراین متغییر index تعداد اندیس های یک آرایه را در خود دارد و با استفاده از متغییر x به تک تک این متغییرها مقدار می دهیم. در صورت سوال گفته شده است که پارمتر سازنده، تعداد بعدهای این بردار می باشد، بنابراین وقتی در قسمت main()، عدد 3 را برای این سازنده می فرستیم بدین معنی می باشد که بردار ما 3 بعدی می باشد و وتعداد بعدهای آن از اندیس 0 تا n-1 یعنی دارای اندیس 0 و 1 و 2 می باشد، بنابراین با استفاده از پارامتر x که همان داده ها می باشند، این داده ها را در پارامتر index که همان شماره اندیس ها می باشد قرار می دهیم. شکل این برنامه به صورت زیر می باشد:

مقدار	شماره اندیس
3	0
8	1
5	2

متد show() : این متد جز متدهای نمایش داده ها و اطلاعات می باشد و در قبل هم گفتیم که اگر اشاره گر داشتیم، تمام داده های ما در داخل اشاره گر قرار می گیرند و هم در قسمت cin و هم در قسمت cout و هم برای انجام

محاسبات، از اشاره گرها استفاده می کنیم. بنابراین در این متد هم برای نمایش اطلاعات طبق نکته های قبلی، ابتدا یک حلقه ی for می نویسیم، سپس تمام اشاره گرها را با استفاده از دستور cout در خروجی نشان می دهیم.

مثال 2-6 : کلاسی برای کار با " زمان " در نظر بگیرید. این کلاس را طوری طراحی کنید که یک سازنده داشته باشد که مقدار پیش فرض 00:00:00 (یعنی هم ساعت و هم دقیقه و هم ثانیه در ابتدای کار دارای مقدار صفر باشند) را برای زمان آغازین در نظر بگیرد. سازنده کلاس را طوری طراحی کنید که پارامتر ساعت بین اعداد 23-0 و پارامتر دقیقه بین اعداد 59-0 و پارامتر ثانیه بین اعداد 59-0 باشد، در غیر این صورت اگرهرکدام از این سه پارامتر خارج از این مقادیر بودند، مقدار صفر برای آن در نظر گرفته شود و در نهایت زمان را در خروجی چاپ کند.


```
#include<iostream.h>
#include<conio.h>
class time
{
private:
        int hour, minute, second;
public:
        time(int a, int b, int c);          سازنده سه پارامتری برای ساعت و دقیقه و ثانیه
        void print();                       متد نمایش داده ها
};
time::time(int a, int b, int c)             پیاده سازی متد سازنده
{
        if (a<0 || a>23)
        {
                a = 0;
        }
        if (b<0 || b>59)
        {
                b = 0;
        }
        if (c<0 || c>59)
        {
                c = 0;
        }
        hour = a;
        minute = b;
        second = c;
        a = 0;
        b = 0;
        c = 0;
}
void time::print()                          پیاده سازی متد print
```

```
{
        cout << hour << ":" << minute << ":" << second;
}
int main()                                              تابع main()
{
        time t1(21, 43, 28);
        t1.print();
        getch();
        return 0;
}
```


توضیح کلاس : چون در این کلاس ما قصد داریم با زمان کار کنیم بنابراین باید متد سازنده ما دارای سه پارامتر باشد که پارامتر اول مشخص کننده ی ساعت (houre) و پارامتر دوم مشخص کننده ی دقیقه (minute) وپارامتر سوم مشخص کننده ی ثانیه (second) باشد. دقت کنید که در صورت سوال نگفته است که متد دریافت داده ها را نیز به کلاس اضافه کنید، بنابراین ما هم فقط یک متد سازنده برای مقدار دادن به پارامترها و یک متد print() برای نمایش داده ها برای این کلاس پیاده سازی می کنیم. چون در صورت سوال گفته شده است که محدوده اعداد را کنترل کنید بنابراین از سه دستور if استفاده شده است که اولی کنترل کننده ی ساعت و دومی کنترل کننده ی دقیقه و سومی کنترل کننده ی ثانیه می باشد. در هر سه این ها چک می شود که اگر اعداد دریافتی خارج از محدوده مورد نظر بود، آن را به صفر تبدیل کند. قبلا هم گفتیم که در داخل متد سازنده، تک تک متغییرهای قسمت private را با استفاده از کاراکتر (=) مساوی پارامترهای خود سازنده قرار می دهیم. علاوه بر این گفته شده است که در صورت مقدار ندادن به این سه پارامتر،مقدار پیش فرض صفر را برای هر سه پارامتر در نظر بگیرید که همه ی این دستورات را در داخل متد سازنده پیاده سازی کرده ایم.

مثال 7-2 : کلاسی برای کار با چند جمله ای ها طراحی کنید. سازنده ای برای کلاس طراحی کنید که درجه ی چند جمله ای را مشخص کند. یک چند جمله ای بدین صورت نوشته می شود: $20x^3+16x^2+18x+24$

```
#include<iostream.h>
#include<conio.h>
class polynomial
{
        int *p;                 اشاره گر p که ضرایب چند جمله ای را در خود نگه می دارد
        int d;                  متغییر d که نشان دهنده ی درجه ی چند جمله ای می باشد
public:
        polynomial(const polynomial & ob);          متد سازنده کپی
        polynomial(int  d1);                        متد سازنده
        ~polynomial();                              متد مخرب
        void insert(int i, int a);                  متد دریافت داده ها
        void print();                               متد نمایش داده ها
};
polynomial::polynomial(const polynomial & ob)       پیاده سازی متد سازنده کپی
{
        p = new int[d + 1];
```

```
        if (!p)
        {
        cout >> "Erorr!";
        exit(1);
        }
         d = ob.d;
        for (int i = 0; i <= d; i++)
        {
                p[i] = ob.p[i];
        }
}
polynomial::polynomial(int  d1)
{
        p = new int[d1 + 1];
        if (!p)
        {
                cout << "Error";
                exit(1);
        }
        d = d1;
        for (int i = 0; i <= d; i++)
                p[i] = 0;
}
polynomial::~polynomial()
{
        delete[]p;
}
void polynomial::insert(int i, int a)
{
        if (i<0 || i>d)
        {
                cout << "out of index";
                exit(1);
        }
        p[i] = a;
}
void polynomial::print()
{
        for (int i = d; i>0; i--)
        {
                if (p[i] != 0)
```

پیاده سازی متد سازنده

پیاده سازی متد مخرب

پیاده سازی متد insert

پیاده سازی متد print()

```
            {
                cout << p[i] << "x^" << i;
            }
        }
        cout << p[0];
}

int main()                                          تابع main()
{
        polynomial  ob1(3);                         این چند  جمله ای حداکثر دارای 1+3 درجه می باشد
        ob1.insert(0, 2);                           قرار دادن مقدار 2 برای ضریب x به توان 0
        ob1.insert(1, 8);                           قرار دادن مقدار 8 برای ضریب x به توان 1
        ob1.insert(2, 5);                           قرار دادن مقدار 5 برای ضریب x به توان 2
        ob1.insert(3, 12);                          قرار دادن مقدار 12 برای ضریب x به توان 3
        ob1.print();
        getch();
        return 0;
}
```

توضیح کلاس : در ابتدا یک اشاره گر به نام p که ضرایب چند جمله ای را در خود نگه می دارد و یک متغییر به نام d که درجه ی چندجمله ای می باشد را در قسمت private قرار می دهیم سپس یک سازنده که درجه ی چندجمله ای را کنترل می کند پیاده سازی می کنیم. در مرحله بعد یک متد دریافت کننده که دارای دو پارامتر می باشد (پارامتر اول: محل قرار گرفتن ضرایب چند جمله ای و پارامتر دوم: مقدار ضرایب می باشد) را به کلاس خود اضافه می کنیم و در آخر نیز از یک متد ()print برای نمایش چندجمله ای استفاده می کنیم. دقت کنید که چون در برنامه خود از اشاره گر استفاده می کنیم، بنابراین باید یک متد سازنده کپی نیز به کلاس خود اضافه کنیم. علاوه بر این باید یک متد مخرب هم بنویسیم تا حافظه ی ایجاد شده را دوباره به سیستم برگرداند. یک چند جمله ای بدین صورت نوشته می شود: $20x^3+16x^2+18x+24$ همان طور که می بینید، بالاترین درجه ی چند جمله ای، 3 می باشد. اما تعداد درجه های چند جمله ای، همیشه یک واحد بیش تر از بالاترین درجه ی چند جمله ای می باشد، به خاطر همین است که در داخل متد سازنده مقدار d که همان درجه ی چند جمله ای می باشد را به علاوه ی عدد 1 کرده ایم.

مثال 2-8 : پشته ای را با استفاده از کلاس ها پیاده سازی کنید. یک پشته عبارت است از : یک آرایه که تعدادی داده را در خود نگه می دارد و در خروجی از آخر به اول آن ها را نمایش می دهد. (بدون اشاره گر)

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
class stackint
{
        int stack[100];                             یک آرایه 100 عضوی که 100 عدد صحیح را در خود نگه می دارد
        int tos;                                    متغییر tos اندیس عنصر خالی بالای پشته می باشد
public:
        stackint();                                 متد سازنده
```

```
        void push(int x);        متد دریافت داده ها که پارامتر دار می باشد و مقدار x را در پشته قرار می دهد
        int pop();               متد چاپ داده ها
};
stackint::stackint()             پیاده سازی متد سازنده
{
        tos = 0;
}
void stackint::push(int x)       پیاده سازی متد دریافت داده ها که push نام دارد
{
        if (tos >= 100)
        {
                cout << "stack is full";
                exit(1);
        }
        stack[tos] = x;
        tos++;
}
int stackint::pop()              پیاده سازی متد نمایش داده ها که pop نام دارد
{
        if (tos <= 0)
        {
                cout << "stack is empty";
                exit(1);
        }
        tos--;
        return stack[tos];
}
int main()
{
        stackint s1, s2;
        for (int i = 0; i <= 10; i++)
        {
                s1.push(i);
        }
        for (int j = 0; j <= 10; j++)
        {
                cout << s1.pop() << " ";
        }
        cin.get();
        cin.get();
        return 0;
```

}

توضیح کلاس : پشته ها یکی از ساختمان داده هایی هستند که در علم کامپیوتر از آن ها استفاده های زیادی می شود. پشته یک آرایه ای می باشد که تعدادی داده را در خود نگه می دارد و در خروجی از آخر به اول آن ها را نمایش می دهد. بنابراین برای ذخیره این مقادیر نیاز به یک آرایه به نام stack داریم که همه ی این داده ها رادر خود نگه داری کند. تعداد عنصرهای این آرایه هم برابر با 100 قرار داده ایم یعنی این که پشته ما تا 100 عدد را می تواند در خود نگه داری کند. متغییر tos عنصر خالی بالای آرایه می باشد و ما با استفاده از این متغییر چک می کنیم که آیا پشته ما پر است یا نه. در ابتدا از یک سازنده برای این کلاس استفاده می کنیم که مقدار tos را در ابتدا برابر صفر قرار می دهد و این بدین معنی می باشد که در ابتدا پشته ما خالی است و هیچ مقداری را در خود جای نداده است. متد push هم یک متد دریافت داده ها می باشد که دارای یک پارامتر به نام x می باشد و این پارامتر همان مقداری است که باید در پشته قرار دهد. اگر به خاطر داشته باشید در مطالب قبل هم گفتیم که متد دریافت داده ها همیشه دارای نوع بازگشتی void می باشد. و در آخر هم از یک متد چاپ داده ها به نام pop برای نمایش داده ا استفاده می کنیم.
به این نکته دقت کنید که چون از آرایه ها استفاده می کنیم و طول این آرایه 100 عنصر می باشد، بنابراین در قسمت main() با استفاده ز متد push، باید بیش از یک مقدار را وارد پشته کنیم. مثلا در این مثال ما از یک حلقه for استفاده کرده ایم که ازعدد 0 تا 10 را شمارش می کند و با استفاده از متد push که در داخل این حلقه قرار دارد تک تک این مقادیر را (یعنی اعداد 0 تا 10) را در داخل پشته قرار می دهد. و برای نمایش این اعداد هم از یک حلقه for استفاده می کنیم و تک تک این داده ها را با استفاده از متد pop در خروجی نمایش می دهیم.
دقت کنید که متد push را با استفاده از دستور cout در خروجی قرار نداده ایم ولی متد pop را با استفاده از دستور cout در خروجی نوشته ایم. دلیل این کار این است که چون متد دریافت داده ها همیشه نوع بازگشتی آن void می باشد و نباید در خروجی نوشته شود اما متد چاپ داده ها در این کلاس به صورت نوع بازگشتی int آمده است، بنابر این باید این متد را با استفاده از دستور cout در خروجی بنویسیم. در داخل متد push کدهایی نوشته شده است که بدین صورت بیان می شود: در ابتدا چک می کند که اگر اندیس خالی بالای پشته که همان tos می باشد، بزرگتر از طول پشته ما، که در این جا 100 در نظر گرفته شده است، باشد یک پیغام مبتنی بر این که پشته پر است را چاپ کند و با استفاده از دستور exit(1) از برنامه خارج شود. اما اگر پشته پر نبود، مقدار دریافتی x را در داخل آرایه مورد نظر قرار دهد و به خانه شماره بعد برود که برای این کار یک واحد به متغییر tos اضافه می کنیم.
کدهای مربوط به متد pop نیز بدین صورت بیان می شوند: ابتدا می آید چک می کند که اگر اندیس خالی بالای آرایه یا همان tos مساوی با عدد صفر بود، یک پیغام مبتنی بر این که پشته خالی است را چاپ کند و با استفاده از دستور exit(1) از برنامه خارج شود. اما اگر پشته خالی نبود، به ترتیب از خانه ی بالای پشته به سمت پایین پشته بیاید و تک تک این داده ها را نمایش دهد برای این کار از دستور --tos استفاده کرده ایم که از آخر پشته به سمت اول حرکت کند و چون نوع بازگشتی متد pop از نوع int می باشد، بنابراین تک تک داده ها را که در آرایه stack ذخیره شده اند با استفاده از دستور return برگشت می دهد.

مثال 2-9 : پشته مثال قبل را طوری پیاده سازی کنید که دارای اشاره گر هم باشد و با استفاده از آرایه های پویا، تعداد خانه های پشته را هر اندازه که می خواهیم تعیین کنیم و دیگر لازم نباشد که دارای یک مقدار ثابت مثلا 100 باشد بلکه طول پشته را به هر مقدار که خودمان می خواهیم تعیین کنیم.

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
class stackint
{
        float *p;
        int size;
```

```
        int tos;
public:
        stackint(int a);
        ~stackint();
        void push(int x);
        int pop();
};
stackint::stackint(int a)
{
        p = new float[a];
        if (!p)
        {
                cout << "Erorr!";
                exit(1);
        }
        size = a;
}
Stackint::~stackint()
{
        delete[]p;
}
void stackint::push(int x)
{
        if (tos >= size)
        {
                cout << "dtack is full";
                exit(1);
        }
        p[tos] = x;
        tos++;
}
int stackint::pop()
{
        if (tos <= 0)
        {
                cout << "stack is empty";
                exit(1);
        }
        tos--;
        return p[tos];
}
```

علی نظری زاده

```
int main()
{
	stackint s1(20);
	for (int i = 0; i <= 10; i++)
	{
		s1.push(i);
	}
	for (int j = 0; j <= 10; j++)
	{
		cout << s1.pop() << " ";
	}
	getch();
	return 0;
}
```

توضیح کلاس : این کلاس تقریبا مانند مثال قبل می باشد با این تفاوت که در این جا از اشاره گرها استفاده کرده ایم و کمی برنامه خود را پیشرفته تر پیاده سازی کرده ایم. در این جا چون اشاره گر داریم، دیگر نیازی نیست که طول آرایه خود را در ابتدا مشخص کنیم بلکه از یک اشاره گر استفاده می کیم که داده های ما را در خود ذخیره کند و در خروجی آن ها را نمایش دهد. علاوه براین از یک متغییر جدید به نام size که محدوده آرایه را کنترل می کند نیز استفاده کرده ایم. دقت کنید که ما در قسمت ()main مقدار 20 را برای سازنده فرستاده ایم، این بدین معنی می باشدکه پشته ما حداکثر تا سقف 20 عدد را می تواند در خود نگه دارد و اگر در داخل حلقه for عدد پایانی خود را بیش تر از 20 نوشتیم برنامه یک پیغام خطا می دهد. تمامی متد های دیگر نیز با استفاده از مطالب قبلی پیاده سازی کرده ایم.

گاهی اوقات شرایط تغییر می کند و " قسمت سوم کلاس " که همان تابع ()main می باشد را به ما می دهند و می گویند با توجه به ()main داده شده متدها های لازم را پیاده سازی کنید. یعنی " قسمت سوم " را به ما می دهند و ما باید " قسمت اول " و " قسمت دوم " را پیاده سازی کنیم. مانند مثال زیر:

مثال 2-10 : با توجه به ()main داده شده متدهای لازم را پیاده سازی کنید. (کلاس data برای کار با تاریخ در نظر گرفته شده است و تاریخ را براساس سال و ماه به صورت شمسی دریافت می کند و معادل آن را به میلادی تبدیل می کند).

```
#include<iostream.h>
#include<conio.h>
class data
{
private:
	int year, month;
public:
… … … ;
… … … ;
};
int main()
{
```

```
        data d1;
        d1.insert(1394, 11);
        d1.calculate();
        d1.show();
        cin.get();
        cin.get();

}
```

این کلاس به صورت زیر پیاده سازی می شود:

```
#include<iostream.h>
#include<conio.h>
class data
{
private:
        int year, month;
public:
        void insert(int year, int month);
        void calculate();
        void show();
};
void data::insert(int year1, int month1)
{
        year = year1;
        month = month1;
}
void data::calculate()
{
        if (month <= 10)
        {
                year += 621;
                month += 2;
        }
        else
        if (month > 10)
        {
                year += 622;
                month -= 10;
        }
}
void data::show()
{
```

```
        cout << year << "/" << month;
}
int main()
{
        data d1;
        d1.insert(1394, 11);
        d1.calculate();
        d1.show();
        getch();
        return 0;
}
```

توضیح کلاس : کلاس بالا مربوط به تاریخ می باشد و در صورت سوال هم گفته شده است که با دو نوع داده ای سال و ماه کار می کنیم. اول از همه به قسمت ()main نگاه می کنیم، در این قسمت سه متد وجود دارد که متد ()insert همان متد دریافت کننده می باشد و دارای دو پارامتر است که پارامتر اولی مربوط به سال و پارامتر دومی مربوط به ماه می باشد و یک متد محاسبه کننده به نام calculate دارد که عملیات تبدیل تاریخ از شمسی به میلادی را انجام می دهد و در آخر هم یک متد چاپ داده دارد که تاریخ تغییر یافته را به خروجی می برد.
با توجه به مطالب قبل، چون متد insert پارامتر دار می باشد، بنابراین از دستور cin و cout در داخل این متد استفاده نمی کنیم و فقط تک تک متغییرهای قسمت private را مساوی با پارامتر های آن قرار می دهیم.
به این نکته هم خوب دقت کنید، چون در قسمت private اشاره گر نیاورده است بنابراین برنامه ما نه به متد مخرب و نه به متد سازنده کپی نیاز دارد. و چون اشیاء ما هم درقسمت ()main فاقد پارامتر می باشند، بنابراین نتیجه می گیریم که به متد سازنده هم نیازی نیست.

مثال 2-11 : با توجه به ()main داده شده متدهای لازم را پیاده سازی کنید. (کلاس student برای کار با نمرات دانش آموزان طراحی شده است و اسم و نمره دانش آموزان را دریافت و هم آن ها را در خروجی نشان می دهد و هم میانگین نمرات کل دانش آموزان را محاسبه می کند.

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include<string>
class student
{
private:
        string *p;                      اشاره گر p از نوع رشته ای
        string str;                     رشته ای که نام دانش آموزان را ذخیره می کند
        float score[100];               یک آرایه که نمره ی دانش آموزان را ذخیره می کند
        float avrage;                   یک متغییر که معدل کل دانش آموزان را ذخیره می کند
        int x;                          متغییری که نشان دهنده ی تعداد دانش آموزان می باشد
public:
… … …;
… … …;
```

```
};
int main()
{
        student s1(5);
        s1.input();
        s1.calculate();
        s1.show();
        getch();
        return 0;
}
```

این کلاس به صورت زیر پیاده سازی می شود:

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include<string>
class student
{
private:
        string *p;                          اشاره گر p از نوع رشته ای
        string str;                         رشته ای که نام دانش آموزان را ذخیره می کند
        float score[100];                   یک آرایه که نمره ی دانش آموزان را ذخیره می کند
        float avrage;                       یک متغییر که معدل کل دانش آموزان را ذخیره می کند
        int x;                              متغییری که نشان دهنده ی تعداد دانش آموزان می باشد
public:
        student(int a);                     متد سازنده تک پارامتری
        ~student();                         متد مخرب
        student(const student & ob);        متد سازنده کپی
        void input();                       متد دریافت داده ها
        void calculate();                   متد محاسبه
        void show();                        متد نمایش داده ها
};
student::student(const student &ob)         پیاده سازی متد سازنده کپی
{
        p = new string[ob.x];
        if (!p)
        {
                cout << "Erorr!";
                exit(1);
        }
        x = ob.x;
```

```
        for (int i = 0; i < ob.x; i++)
        {
                p[i] = ob.p[i];
        }
}
student::student(int a)                                  پیاده سازی متد سازنده
{
        p = new string[a];
        if (!p)
        {
                cout << "Erorr!";
                exit(1);
        }
        x = a;
}
student::~student()                                      پیاده سازی متد مخرب
{
delete[]p;
}
void student::input()                                    پیاده سازی متدinput
{
        for (int i = 0; i < x; i++)
        {
                cout << "enter name student " << i + 1 << ": ";
                cin >> p[i];
                cout << "enter score student " << i + 1 << ": ";
                cin >> score[i];
        }
}
void student::calculate()                                پیاده سازی متد calculate
{
float avg = 0, avrage1;
int count = 0;
for (int i = 0; i < x; i++)
{
avg += score[i];
count += 1;
}
avrage1 = avg / count;
avrage = avrage1;
}
```

پیاده سازی متد show()

```
void student::show()
{
	cout << endl;
	for (int i = 0; i < x; i++)
	{
		cout << "student " << i+1 << ": " << p[i] << ": " << score[i] << endl;
	}
	cout << endl;
	cout << "avrage student: " << avrage;
}
int main()
{
	student s1(5);
	s1.input();
	s1.calculate();
	s1.show();
	getch();
	return 0;
}
```

توضیح کلاس : این کلاس اسامی تعدادی از دانش آموزان به همراه نمره آن ها را دریافت می کند و در خروجی اسامی دانش آموزان به همرا نمره آن ها نمایش داده می شود. علاوه بر این، میانگین نمرات کل دانش آموزان را نیز محاسبه می کند. قبلا هم گفتیم که از سازنده ها برای مشخص کردن تعداد افراد یا تعداد بعدهای یک بردار استفاده می شود. در این جا هم برای مشخص کردن تعداد دانش آموزان استفاده می کنیم و اسامی دانش آموزان را هم در اشاره گر p ذخیره می کنیم. آرایه ی score نیز نمرات تک تک دانش آموزان را در خود جای میدهد و متغییر average نیز میانگین نمرات کل کلاس را در خود ذخیره می کند. از یک متغییر کمکی به نام x که نشان دهنده ی تعداد دانش آموزان می باشد نیز استفاده می کنیم.از متغییر avg برای جمع همه ی نمرات دانش آموزان و از متغییر count نیز برای شمارش تعداد دانش آموزان استفده می کنیم. دقت کنید که چون از اشاره گرها استفاده می کنیم باید متد سازنده کپی و متد مخرب نیز به کلاس اضافه کنیم. در قسمت ()main، مقدار پارامتر سازنده برابر با عدد 5 می باشد، این بدین معنی می باشد که تعداد دانش آموزان 5 نفر می باشند.

مثال 12-2 : کلاسی به نام calculator طراحی کنید که قابلیت های یک ماشین حساب معمولی را داشته باشد.

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
#include<string>
class calculator
{
private:
	string str;
	char ch;
```

```
        float a, b;
        float n;
public:
        void math1();
        void math2();
};
void calculator::math1()
{
        cin >> a >> ch >> b;
        if (ch == '+')
                cout << a + b;
        else
        if (ch == '-')
                cout << a - b;
        else
        if (ch == '*')
                cout << a*b;
        else
        if (ch == '/')
                cout << a / b;
}
void calculator::math2()
{
        cin >> str >> n;
        if (str == "sin")
        {
                cout << sin(n);
        }
        else if (str == "cos")
        {
                cout << cos(n);
        }
        else if (str == "tan")
        {
                cout << tan(n);
        }
        else if (str == "sqrt")
        {
                cout << sqrt(n);
        }
        else if (str == "log")
```

```
        {
                cout << log(n);
        }
        else if (str == "asin")
        {
                cout << asin(n);
        }
        else if (str == "acos")
        {
                cout << acos(n);
        }
        else if (str == "atan")
        {
                cout << atan(n);
        }
        else if (str == "sinh")
        {
                cout << sinh(n);
        }
        else if (str == "cosh")
        {
                cout << cosh(n);
        }
        else if (str == "tanh")
        {
                cout << tanh(n);
        }
}
int main()
{
        calculator ob1;
        ob1.math1();
        cout << endl;
        ob1.math2();
        getch();
        return 0;
}
```

توضیح کلاس: در C++ یک کتابخانه ی ریاضی به نام math وجود دارد که توابع ریاضی زیادی مانند sin و cos و log و ... در آن وجود دارد. برای استفاده از این توابع ریاضی باید فایل کتابخانه math را به صورت #include<math.h> به برنامه اضافه کرد. مثلا برای دریافت مقدار: sin 90 ما به دو متغییر نیاز داریم که اولی از

نوع رشته ای که مثلا عبارت sin یا tan یا ... را از ورودی دریافت کند و متغییر دومی باید از یک نوع اعشاری باشد که یک مقدار را دریافت می کند. بنابراین ما این عملیات را در قالب یک متد به نام math2() پیاده سازی کرده ایم. علاوه براین، چهار عملیات اصلی (+ و – و * و /) نیز با استفاده از یک متد به نام math1 طراحی کرده ایم که دو عدد و یک کاراکتر را از ورودی دریافت می کند و با استفاده از کاراکتر ورودی عملیات مناسب را بر روی اعداد انجام می دهد. مثلا اگر کاراکتر ورودی '+' بود، دو عدد را با هم جمع کند و الی آخر. دقت کنید که دو متد math1() و math2() همان متدهای دریافت داده ها می باشند که ما در این جا آن ها را این چنین نام گذاری کرده ایم. دقت کنید که هر این دو متد بدون پارامتر می باشند، بنابراین در داخل آن ها از دستور cin و cout استفاده کرده ایم، یعنی این که داده ها را با استفاده از صفحه کلید وارد می کنیم.

فصل سوم: سربارگزاری عملگرها

سربارگزاری عملگرها یکی از قدرتمند ترین بخش های C++ می باشد که با استفاده از این روش می توان عملیات محاسباتی را در قسمت main() بر روی اشیاء انجام داد. مانند عملیات جمع یا تفریق یا ضرب یا ... و یا حتی اگر اشیاء را با استفاده از دستور cin از ورودی دریافت یا با استفاده از دستور cout در خروجی نمایش داد، باید از سربارگزاری عملگرها استفاده کرد. اگر در داخل main() چنین کدی بنویسیم با خطا مواجه می شویم و برنامه اجرا نمی شود.

```
#include<iostream.h>
#include<conio.h>
class vector
{
        ... ... ...;
        ... ... ...;
};
int main()
{
        vector v1, v2, v3;
        v3 = v1 + v2;
        cout << v3;
        getch();
        return 0;
}
```

توضیح کلاس : در قسمت main() برنامه بالا، 3 شی به نام های v1 و v2 و v3 وجود دارند که حاصل جمع شی v1 و شی v2 در شی v3 ریخته می شود و در آخر سر هم شی v3 در خروجی نشان داده شده است. اگر برنامه بالا را اجرا کنیم با خطا مواجه می شویم، چون دراین قسمت دو شی با هم جمع شده اند و شی نهایی نیز با استفاده از دستور cout در خروجی نمایش داده شده است. طبق مطالب قبلی گفتیم که هر گاه عملیات محاسباتی بر روی اشیاء انجام

دادیم یا آن ها را با استفاده از دستور cin از ورودی گرفتیم یا با استفاده از دستور cout در خروجی نمایش دادیم، باید از روش سربارگزاری عملگرها استفاده کنیم.

نکته 3-1 : با استفاده از عملیاتی که بر روی اشیاء انجام می شود باید دقیقا همان عملگر را سربارگزاری کرد مثلا اگر چنین کدی نوشته شده بود باید تک تک این عملگرها را سربار گزاری کرد:


```
class vector
{
        ... ... ...;
        ... ... ...;
};
int main()
{
        vector v1, v2, v3;          خط اول: ایجاد سه شی به نام های v1 و v2 و v3
        cin >> v1 >> v2 >> v3;      خط دوم: باید عملگر cin را سربارگزاری کرد
        v3 = v1 + v2;               خط سوم: باید عملگر جمع ( + ) را سربارگزاری کرد
        v2 = v1 / v3;               خط چهارم: باید عملگر تقسیم ( / ) را سربارگزاری کرد
        cout << v3;                 خط پنجم: باید عملگر cout را سربارگزاری کرد
        cout << v2;                 خط ششم: باید عملگر cout را سربارگزاری کرد
        getch();
        return 0;
}
```

توضیح کلاس : طبق مطالب قبل گفتیم اگر در قسمت ()main هر عملیات محاسباتی بر روی اشیاء انجام شد باید دقیقا همان عملگر را سربار گزاری کنیم، که در کلاس بالا در قسمت ()main کدهایی نوشته شده است که به صورت زیر تعریف می شوند:

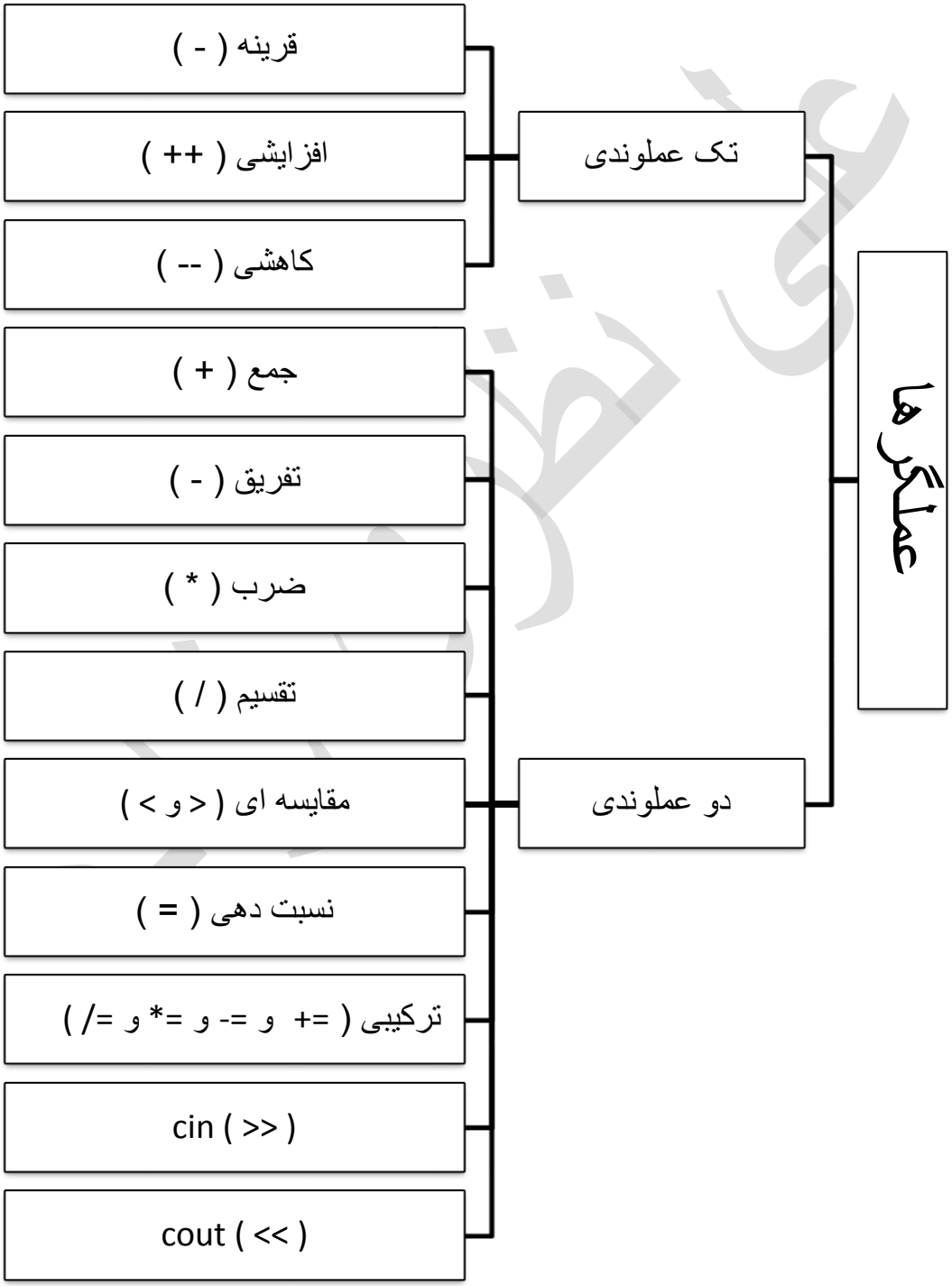
خط اول : در این جا سه شی با نام های v1 و v2 و v3 به وجود آمده است.
خط دوم : در این جا مشاهده می کنیم که هر سه شی توسط دستور ورودی cin از ورودی گرفته می شوند، بنابراین باید عملگر cin را سربارگزاری کرد.
خط سوم : در این قسمت حاصل جمع دو شی v1 و v2 در شی v3 ریخته می شود، بنابراین باید عملگر جمع (+) را سربارگزاری کنیم.
خط چهارم : در این قسمت نیز شی v1 را تقسیم بر شی v3 کرده ایم و حاصل آن را در شی v2 ریخته ایم، بنابراین باید عملگر تقسیم (/) را سربارگزاری کرد.
خط پنجم : در این مرحله نیز شی v3 با استفاده از دستور cout در خروجی نمایش داده شده است، بنابراین باید عملگر cout را سربارگزاری کرد.
خط ششم : در این قسمت نیز همانند خط پنجم، باید عملگر cout را سربارگزاری کرد.

نکته 3-2 : دقت کنید که هر عملگر فقط یک بار سربارگزاری می شود. مثلا در خط پنجم و ششم عملگر cout نوشته شده است، اما فقط یک بار این عملگر را سربار گزاری می کنیم و به هر تعداد که در قسمت ()main مورد استفاده قرار بگیرد، مشکلی پیش نمی آید چون ما آن را سربارگزاری کرده ایم. ما در این فصل اکثر عملگرهایی که زیاد مورد استفاده قرار می گیرند و با آن ها سروکار داریم را سربارگزاری می کنیم.

عملگرهایی که قرار است در این فصل آن ها را سربارگزاری کنیم عبارت اند از:

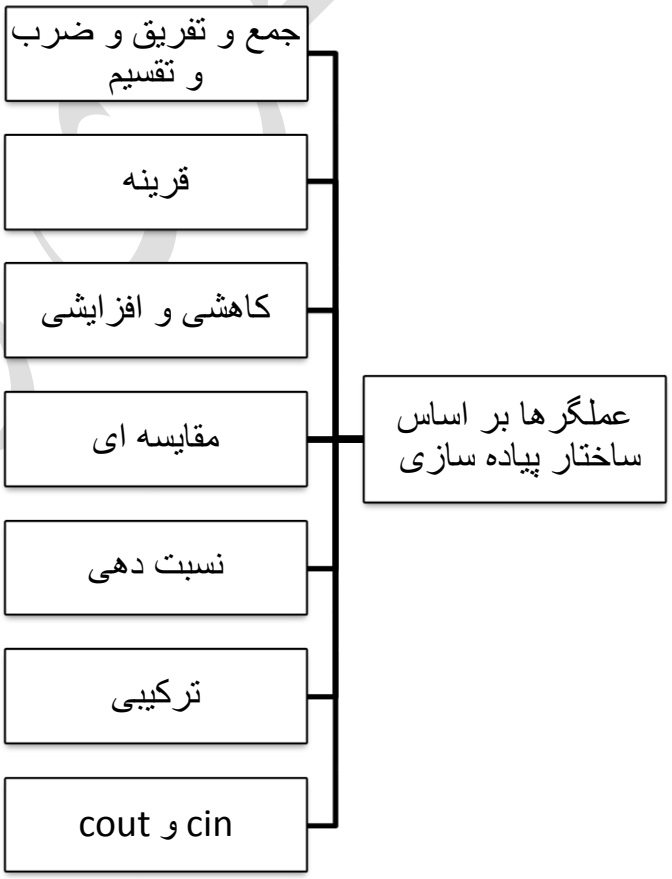
1- عملگر جمع (+) 2- عملگر تفریق (-) 3- عملگر ضرب (*) 4- عملگر تقسیم (/) 5- عملگر قرینه (-)
6- عملگر افزایشی (++) 7- عملگر کاهشی (--) 8- عملگرهای مقایسه ای (< و >) 9- عملگر نسبت دهی (=)
10- عملگرهای ترکیبی (=+ و =* و =- و =/) 11- عملگر cin (<<) 12- عملگر cout (>>).

این عملگرها را می توان به صورت کلی در دو گروه به صورت زیر دسته بندی کرد:



طبق جدول قبل، عملگرها به دو دسته اصلی تک عملوندی و دو عملوندی تقسیم می شوند. عملگرهای تک عملوندی بدین معنی می باشد که فقط بر روی یک عملوند عملیات محاسباتی انجام می دهند مانند عملگر قرینه که بدین صورت بیان می شود: a=28; b=-a; ابتدا متغییر a را مساوی عدد 28 قرار داده ایم و قرینه آن را در متغییر دیگری به نام b ریخته ایم. همان طور که مشاهده می کنید عملگر قرینه فقط بر روی یک عملوند که a نام دارد، عمل می کند و مقدار آن را به 28- تبدیل می کند یا مثلا عملگر افزایشی یا کاهشی نیز جز عملگرهای تک عملوندی می باشند که مثلا عملگر افزایشی بدین صورت بیان می شود a=32; a++; در این جا نیز، ابتدا متغییر a را مساوی عدد 32 قرار داده ایم و در آخر هم یک واحد به آن اضافه کرده ایم که جواب پایانی برابر عدد 33 خواهد شد.
اما عملگرهای دو عملوندی بر روی دو عملوند عملیات محاسباتی را انجام می دهند مانند عملگر جمع یا ضرب یا ... مثلا عملگر جمع بدین صورت عمل می کند: a=12, b=6; c=a+b; ابتدا دو متغییر به نام های a که برابر با عدد 12 و b که برابر با عدد 6 می باشند را به وجود آورده ایم، و در آخرسر هم این دو مقدار را با هم جمع کرده ایم و جواب نهایی را در متغییر c ریخته ایم. بنابراین عملگر جمع بر روی دو متغییر به نام های a و b عملیات محاسباتی را انجام می دهند، بنابراین این عملگر دو عملوندی می باشد.

عملگرها با توجه به نوع ساختار و فرم پیاده سازی به صورت زیر دسته بندی می شوند:



همان طور که در این جدول مشاهده می کنید، هر کدام از عملگرها در یک گروه قرار می گیرند و ساختار پیاده سازی آن ها به یک فرم می باشد. مثلا اگر سربارگزاری عملگر جمع را یاد گرفته باشیم، می توانیم عملگرهای تفریق و ضرب وتقسیم را نیز به راحتی پیاده سازی کنیم چون ساختار این چهار عملگر به یک صورت می باشد.

نکته 3-3 : همیشه برای سربارگزاری هر عملگری، از کلمه ی کلیدی operator استفاده می کنیم. برای این کار ابتدا در " قسمت اول " تابع مربوط به سربارگزاری آن عملگر را پیاده ساری می کنیم، که به صورت زیر می باشد:
(شی اسم کلاس) نوع عملگر کلمه کلیدی operator اسم کلاس

مثلا برای سربارگزاری عملگر جمع، در " قسمت اول " بدین صورت عمل می کنیم: (با فرض این که اسم کلاس vector باشد).

```
vector operator+(vector ob);
```

که در این جا اسم کلاس vector می باشد، آن را می نویسیم سپس کلمه ی کلیدی operator را می نویسیم بعد عملگری که قرار است سربارگزاری کنیم را می نویسیم بعد داخل پرانتزباز اسم کلاس و در آخر سر هم یک شی به وجود می آوریم. بنابراین همیشه برای سربارگزاری هر عملگری، بعد از کلمه ی کلیدی operartor باید آن عملگر را بنویسیم.

نکته 4-3 : دقت کنید که در داخل پرانتز چنین کدی نوشته شده است: ;(vector ob) این بدین معنی می باشد که این تابع دارای یک پارامتر به نام ob می باشد که نوع بازگشتی آن، از نوع خود کلاس یعنی vector می باشد.

نکته 5-3 : در مطالب قبلی هم گفتیم که عملگر جمع دو عملوندی می باشد بنابراین برای انجام محاسبات جمع، ما به دو شی نیاز داریم که با هم جمع شوند و چون نوع بازگشتی کلاس، vector می باشد، بنابراین باید با استفاده از دستور return یک شی را بازگشت دهیم. ترفندی که قرار است برای انجام عملیات جمع انجام دهیم بدین صورت می باشد که، ابتدا باید دو شی را به وجود آوریم و سپس آن ها را با هم جمع کنیم. برای به وجود آوردن این دو شی، ابتدا یک شی را به صورت پارامتر می فرستیم و یک شی را نیز در داخل خود تابع به وجود می آوریم و آن ها را با هم جمع می کنیم. دقت کنید که شی بازگشت داده شده، همان شی که در داخل تابع به وجود آورده ایم می باشد. پس دقت کنید که شی اصلی ما همان شی است که در داخل تابع به وجود آورده ایم، پس باید این شی را با استفاده از دستور بازگشتی return بازگشت دهیم.

بعد از این که در " قسمت اول " تابع عملگر مورد نظر را نوشتیم، سپس باید در " قسمت دوم " کد های مربوط به آن را پیاده سازی کنیم.

نکته 6-3 : به این نکته ی بسیار مهم خوب توجه کنید که سربارگزاری عملگرهایی که به صورت یک گروه در جدول قبل نوشتیم، به یک صورت پیاده سازی می شوند و با توجه به این که در قسمت private یا همان " قسمت اول " اشاره گر وجود داشته باشد یا این که بدون اشاره گر باشد، ساختار مربوط به هر عملگر را پیاده سازی می کنیم. که در مطالب بعدی به ترتیب در صورت وجود اشاره گر یا فاقد اشاره گر در برنامه، ساختار مربوط به هر کدام را پیاده سازی می کنیم.

نکته 7-3 : عملگرهای تک عملوندی، بدون پارامتر می باشند ولی عملگرهای دو عملوندی همیشه دارای یک پارامتر از نوع خود کلاس می باشند. فقط عملگرهای افزایشی و کاهشی در بعضی مواقع خاص یک پارامتر از نوع عدد صحیح می گیرند که در ادامه آن را توضیح خواهیم داد.

1-3 سربارگزاری عملگرهای جمع و تفریق و ضرب و تقسیم

این چهار عملگر، از نوع دو عملوندی می باشند، بنابراین دارای یک پارمتر از نوع خود کلاس می باشند. اگر در قسمت private اشاره گر وجود نداشت بدین صورت عمل می کنیم:


```
class vector
{
        float x, y, z;
public:
vector operator+(vector ob);          نوشتن تابع مربوط به سر بارگزاری عملگر در قسمت public
};
vector vector::operator+(vector ob)      مرحله اول: پیاده سازی متد مربوط به سربارگزاری جمع
{
        vector r;                                   مرحله دوم: ایجاد یک شی به نام r
        r.x = ob.x + x;                             مرحله سوم: انجام عملیات جمع
        r.y = ob.y + y;                             ... ... ...
        r.z = ob.z + z;                             ... ... ...
    return r;                                       مرحله چهارم: بازگشت شی ایجاد شده
}
```

طبق نکته ی قبل، عملگرهای دو عملوندی همیشه یک پارامتر دارند که نوع باز گشتی آن از نوع خود کلاس می باشد. اگر در برنامه ما اشاره گر وجود نداشت، بنابراین عملگر جمع را برای همه ی کلاس ها به صورت بالا پیاده سازی می کنیم. طبق مطالب قبل، شی که به صورت پارامتر فرستاده می شود ob نام دارد و شی دوم که باید ایجاد کنیم، r نام دارد ودر قبل هم گفتیم که باید شی که به وجود آورده ایم را بازگشت دهیم، که در این جا نیز ما شی r را با استفاده از دستور بازگشتی return بازگشت می دهیم. توجه کنید که برای سربارگزاری عملگرهای تفریق و ضرب و تقسیم، نیز به صورت بالا عمل می کنیم. فقط با این تفاوت که هر جا عملگر جمع دیدیم، به جای آن مثلا تفریق یا ضرب یا تقسیم می نویسیم. مثلا کلاس زیر عملگر تفریق را سربار گزاری می کند.

```
class vector
{
        float x, y, z;
public:
vector operator-(vector ob);          نوشتن تابع مربوط به سر بارگزاری عملگر در قسمت public
};
vector vector::operator-(vector ob)      مرحله اول: پیاده سازی متد مربوط به سربارگزاری تفریق
{
        vector r;                                   مرحله دوم: ایجاد یک شی به نام r
        r.x = ob.x - x;                             مرحله سوم: انجام عملیات تفریق
        r.y = ob.y - y;                             ... ... ...
        r.z = ob.z - z;                             ... ... ...
    return r;                                       مرحله چهارم: بازگشت شی ایجاد شده
}
```

مرحله اول : ابتدا عملگری را که قرار است سربارگزاری کنیم در این قسمت یعنی در " قسمت دوم " بدین صورت می نویسیم. دقیقا آن را مانند قسمت public می نویسیم فقط با این تفاوت که قبل از آن، نوع بازگشتی این تابع را باید بنویسیم که نوع بازگشتی آن از نوع خود کلاس یعنی vector می باشد.

مرحله دوم : ابتدا یک شی به وجود می آوریم. همان طور که در مطالب قبل در قسمت ()main نیز داشتیم، برای ایجاد یک شی ابتدا نام کلاس سپس نام آن شی را می نویسیم. که در این جا اسم شی t می باشد.

مرحله سوم : در این مرحله باید دستورات مربوط به عملیات تفریق را پیاده سازی کنیم. برای این کار ابتدا باید با استفاده از شی ایجاد شده به تک تک متغییرهای قسمت private با استفاده از کاراکتر نقطه (.) دسترسی داشته باشیم. سپس یک کاراکتر مساوی (=) قرار می دهیم و در طرف دیگر مساوی، این دفعه نیز باید شی ای را که به صورت پارامتر فرستاده ایم را به تک تک متغییرهای قسمت private با استفاده از کاراکتر نقطه (.) در دسترس خود قرار بدهیم و در آخر نیز کاراکتری که قرار است سربارگزاری کنیم را می نویسیم و سپس متغییر قسمت private را می نویسیم. سپس در خط بعد باز همین عملیات را انجام می دهیم فقط مثلا به جای متغییر x از متغییر بعدی که y نام دارد استفاده می کنیم. یعنی این که باید به هر تعداد که متغییر داشته باشیم، به ترتیب چنین عملیاتی را انجام دهیم.

مرحله چهارم : در قسمت آخر نیز شی اصلی ما، یعنی همان شی ای که ایجاد کرده ایم را با استفاده از دستور بازگشتی return بازگشت می دهیم.

همان طور که مشاهده می کنید، برای سربارگزاری عملگر تفریق نیز همانند عملگر جمع عمل می کنیم با این تفاوت که فقط هرجا عملگر جمع دیدیم به جای آن عملگر تفریق را قرار می دهیم. بنابراین اگر در برنامه اشاره گر وجود نداشت، سربارگزاری این چهار عملگر (جمع و تفریق و ضرب و تقسیم) به صورت سربارگزاری قبل می باشد و برای هر زمانی که بخواهیم این چهار عملگر را سربارگزاری کنیم، از روش بالا استفاده می کنیم و برای همه ی کلاس ها این روند ثابت می باشد.

اما اگر در قسمت private اشاره گر وجود داشت، برای سربار گزاری این چهار عملگر به صورت زیر عمل می کنیم:

```
class vector
{
        float n;                    متغییر n، تعداد بعدهای بردار می باشد
        flo at *p;                  یک اشاره گر به نام p
public:
vector operator*vector ob);         نوشتن تابع مربوط به سر بارگزاری عملگر در قسمت public
};
vector vector::operator*(vector ob)     مرحله اول : پیاده سازی متد مربوط به سربارگزاری ضرب

{
        vector r(n);                    مرحله دوم: ایجاد یک شی به نام r
        if (n != ob.n)                  مرحله سوم: ایجاد یک دستور if
        {
                cout << "Error!";       مرحله چهارم: ایجاد یک پیغام خطا
        }
        for (int i = 0; i<n; i++)       مرحله پنجم: نوشتن یک حلقه for
        {
                r.a[i] = a[i] * ob.a[i];   مرحله ششم: انجام عملیات ضرب
```

مرحله هفتم: بازگشت شی ایجاد شده با استفاده از دستور return return r;

}

نکته 3-8 : دقت کنید وقتی اشاره گر داشتیم، شی ای را که به وجود می آوریم باید متغییر قسمت private را در خل آن بنویسیم. به صورت زیر:

وقتی اشاره گر نبود: vector r;

وقتی اشاره گر داشتیم: vector r(n);

که در قسمت بالا، r شی ایجاد شده می باشد که در جلوی آن متغییر قسمت private را آورده ایم. این بدین معنی می باشد که شی r دارای n بعد می باشد.

مرحله اول : ابتدا عملگری را که قرار است سربارگزاری کنیم در این قسمت یعنی در " قسمت دوم " بدین صورت می نویسیم. دقیقا آن را مانند قسمت public می نویسیم فقط با این تفاوت که قبل از آن، نوع بازگشتی این تابع را باید بنویسیم که نوع بازگشتی آن از نوع خود کلاس یعنی vector می باشد.
مرحله دوم : ابتدا یک شی به وجود می آوریم. همان طور که در مطالب قبل در قسمت ()main نیز داشتیم، برای ایجاد یک شی ابتدا نام کلاس سپس نام آن شی را می نویسیم. که در این جا اسم شی r می باشد.طبق مطالب قبل، اگر اشاره گر داشتیم، شی ای که ایجاد می کنیم باید متغییر داخل قسمت private را در داخل یک پرانتز جلوی آن بنویسیم.
مرحله سوم : در این مرحله یک دستور if می آوریم و در داخل آن شرط می گذاریم که اگر تعداد بعدهای بردار یا همان متغییر قسمت private که n می باشد، مخالف با پارامتری فرستاده شده باشد (این بدین معنی می باشد که تعداد بعد های بردار با شی ای که به صورت پارامتر فرستاده ایم برابر نیست)، در مرحله بعد یک پیغام مبتنی بر خطا چاپ کند.
مرحله چهارم : در این قسمت یک پیغام ;" Error " چاپ می کنیم.
مرحله پنجم : قبلا هم گفتیم که اگر در برنامه ما اشاره گر وجود داشت، باید از حلقه ی for استفاده کنیم. در این مرحله یک حلقه for می نویسیم و از عدد صفر تا n-1 که همان تعداد بعدهای بردار می باشد را شمارش می کنیم. منظور از n-1 همان n> می باشد.
مرحله ششم : در مطالب قبل هم گفتیم که اگر در برنامه اشاره گر وجود داشت، تمام داده های ما در داخل اشاره گر قرار می گیرند و هر عملیات محاسباتی را که بخواهیم انجام دهیم، باید بر روی اشاره گرها انجام دهیم. در این مرحله برای انجام دادن عملیات ضرب، مانند حالت بدون اشاره گر استفاده می کنیم فقط با این تفاوت که دیگر در داخل حلقه for فقط یک خط کد می نویسیم. چون در این قسمت در داخل private متغییر نداریم، پس به جای این متغییرها از اشاره گر موجود در قسمت private استفاده می کنیم.

وقتی که اشاره گر نداشتیم: r.x = ob.x * x; r.y = ob.y * y; r.z = ob.z * z

وقتی که اشاره گر داشتیم: r.a[i] = a[i] * ob.a[i];

همان طور که در بالا می بینید، برای این چهار عملگر (جمع و تفریق و ضرب و تقسیم) در صورت بودن اشاره گر یا نبود اشاره گر، به صورت بالا عمل می کنیم. اگر اشاره گر داشتیم، فقط به جای متغییرهای قسمت private، اشاره گر موجود در برنامه را می نویسیم.

3-2 سربارگزاری عملگر قرینه

این عملگر نیز در صورت وجود اشاره گر یا نبود اشاره گر، پیاده سازی آن کمی فرق می کند. طبق مطالب قبل، عملگرهایی مانند عملگر قرینه که تک عملوندی می باشند، فاقد پارامتر هستند.

اگر در برنامه اشاره گر وجود نداشت، سربارگزاری این عملگر به صورت زیر می باشد:


```
class vector
{
private:
        float x, y, z;
public:
        vector operator-();          نوشتن تابع مربوط به سربارگزاری عملگر در قسمت public
};
vector vector::operator-()           مرحله اول: پیاده سازی متد مربوط به سربارگزاری قرینه
{
        vector r;                    مرحله دوم: ایجاد یک شی به نام r
        r.x = -x;                    مرحله سوم: انجام عملیات قرینه
        r.y = -y;                    ... ... ...
        r.z = -z;                    ... ... ...
    return r;                        مرحله چهارم: بازگشت شی ایجاد شده
}
```

این عملگر نیز تقریبا مانند عملگر جمع سربارگزاری می شود فقط با این تفاوت که در این جا پارامتر نداریم و فقط شی ای را که به وجود آورده ایم را قرینه می کنیم و آن را نیز با استفاده از دستور بازگشتی return بازگشت می دهیم. بنابراین چون در این جا اشاره گر نداریم، باز شی ایجاد شده را با استفاده از کاراکتر نقطه (.) به تک تک متغییر های قسمت private در دسترس خود قرار می دهیم و کاراکتر مساوی قرار می دهیم و تمام این متغییرهار را قرینه می کنیم.

مرحله اول : ابتدا عملگری را که قرار است سربارگزاری کنیم در این قسمت یعنی در " قسمت دوم " بدین صورت می نویسیم. دقیقا آن را مانند قسمت public می نویسیم فقط با این تفاوت که قبل از آن، نوع بازگشتی این تابع را باید بنویسیم که نوع بازگشتی آن از نوع خود کلاس یعنی vector می باشد.
مرحله دوم : ابتدا یک شی به وجود می آوریم. همان طور که در مطالب قبل در قسمت ()main نیز داشتیم، برای ایجاد یک شی ابتدا نام کلاس سپس نام آن شی را می نویسیم. که در این جا اسم شی r می باشد.
مرحله سوم : در این مرحله، عملیات مربوط به قرینه را انجام می دهیم. برای قرینه فقط باید تمام داده ها را در یک منفی (-) ضرب کنیم، که در اینجا داده های ما سه متغییر به نام های x و y و z می باشد که تک تک آن ها را در یک منفی ضرب کرده ایم.
مرحله چهارم : در این مرحله نیز شی ای که به جود آورده ایم را با استفاده از دستور بازگشتی return بازگشت می دهیم.

اگر در برنامه اشاره گر وجود داشت به صورت زیر عمل می کنیم:

```
class vector
{
private:
```

```
        float n;                    متغییر n، تعداد بعدهای بردار می باشد
        flo at *p;                  یک اشاره گر به نام p
public:
        vector operator-();         نوشتن تابع مربوط به سر بارگزاری عملگر در قسمت public
};
vector vector operator-()           مرحله اول: پیاده سازی متد مربوط به عملگر قرینه
{
        vector r(n);                مرحله دوم: ایجاد یک شی به نام r
        for (int i = 0; i < n; i++) مرحله سوم: نوشتن یک حلقه for
        {
                r.p[i] = -p[i]      مرحله چهارم: انجام عملیات قرینه
        }
        return r;                   مرحله پنجم: بازگشت شی ایجاد شده
}
```


مرحله اول : ابتدا عملگری را که قرار است سربارگزاری کنیم در این قسمت یعنی در " قسمت دوم " بدین صورت می نویسیم. دقیقا آن را مانند قسمت public می نویسیم فقط با این تفاوت که قبل از آن، نوع بازگشتی این تابع را باید بنویسیم که نوع بازگشتی آن از نوع خود کلاس یعنی vector می باشد.
مرحله دوم : ابتدا یک شی به وجود می آوریم. همان طور که در مطالب قبل در قسمت ()main نیز داشتیم، برای ایجاد یک شی ابتدا نام کلاس سپس نام آن شی را می نویسیم. که در این جا اسم شی r می باشد. طبق مطالب قبل، اگر اشاره گر داشتیم، شی ای که ایجاد می کنیم باید متغییر داخل قسمت private را در داخل یک پرانتز جلوی آن بنویسیم.
مرحله سوم : قبلا هم گفتیم که اگر در برنامه ما اشاره گر وجود داشت، باید از حلقه ی for استفاده کنیم. در این مرحله یک حلقه for می نویسیم و از عدد صفر تا n-1 که همان تعداد بعدهای بردار می باشد را شمارش می کنیم. منظور از n-1 همان n> می باشد.
مرحله چهارم : در این مرحله عملیات مربوط به عملگر قرینه را انجام می دهیم. برای این کار شی ایجاد شده را با استفاده از کاراکتر نقطه (.) به اشاره گر برنامه نسبت می دهیم و سپس یک مساوی قرار می دهیم و اشاره گر را در یک منفی ضرب میکنیم. دقت کنید که تمام داده های ما در داخل این اشاره گر می باشند بنابراین باید بر روی این اشاره گر عملیات قرینه سازی را انجام دهیم.
مرحله پنجم : در این مرحله نیز شی ایجاد شده را با استفاده از دستور بازگشتی return بازگشت می دهیم.

3-3 سربارگزاری عملگرهای افزایشی و کاهشی

چون این دو عملگر را در یک گروه جای داده ایم، بنابراین اگر سربارگزاری یکی از آن ها را بلد باشیم، سربارگزاری عملگر دیگری نیز به راحتی انجام می گیرد. این عملگر هم تحت وجود اشاره گر در برنامه یا نبود اشاره گر در برنامه، مورد بررسی قرار می گیرد وبه صورت زیر سربارگزاری می شود. ابتدا به این نکته ها در مورد این عملگر دقت کنید:
نکته 3-9 : عملگرهای افزایشی و کاهشی به دو صورت در برنامه نویسی مورد استفاده قرار می گیرند که عبارت اند از: پیشوندی و پسوندی.

1- پیشوندی : مثلا برای عملگر افزایشی (++)، نوع پیشوندی به این صورت می باشد:

```
int a=23;      ++a;
```

همان طور که مشاهده می کنید، عملگر افزایشی (++) قبل از متغییر a بدین صورت می آید: ;++a که این بدین معنی می باشد که ، هنگامی که به متغییر a مثلا مقدار 23 نسبت داده می شود، بلافاصله یک واحد به a اضافه می شود و مقدار آن به عدد 24 تغییر می کند. به این نوع، عملگرهای افزایشی، پیشوندی می گویند.

2- پسوندی : مثلا برای عملگر افزایشی (++)، نوع پسوندی به این صورت می باشد:

```
int a=23;      a++;
```

همان طور که مشاهده می کنید، عملگر افزایشی (++) بعد از متغییر a بدین صورت می آید: ;a++ که این بدین معنی می باشد که ، هنگامی که به متغییر a مثلا مقدار 23 نسبت داده شده است، بلافاصله یک واحد به a اضافه نمی شود، بلکه ابتدا مورد محاسبه قرار می گیرد سپس در مرحله بعد یک واحد به آن اضافه می شود. به این نوع، عملگرهای افزایشی، پسوندی می گویند.

نکته 3-10 : دقت کنید که عملگرهای افزایشی و کاهشی، عملگرهای تک عملوندی می باشند، پس بدون پارامتر می باشند. اما به این نکته خوب دقت کنید که اگر این عملگرها به صورت پیشوندی بیایند، بدون پارامتر می باشند. ولی اگر به صورت پسوندی بیایند، یک پارامتر از نوع عدد صحیح (int) می گیرند. به صورت زیر:

به صورت پیشوندی: vector operator++();

به صورت پسوندی: vector operator++(int x);

نکته 3-11 : سربارگزاری عملگرهای افزایشی و کاهشی به صورت پیشوندی و پسوندی، کمی متفاوت می باشند. طبق مطالبی که در قبل گفتیم، برای سربارگزاری عملگرهای افزایشی و کاهشی به صورت پیشوندی، ابتدا باید یک واحد به تک تک متغییرهای قسمت private اضافه کنیم، سپس آن ها را با استفاده از شی ای که ایجاد کرده ایم به تمام متغییرهای قسمت private در دسترس خود قرار بدهیم. بنابراین برای حالت پیشوندی بدین صورت عمل می کنیم:

```
x=x+1;    r.x=x;
```

طبق دستور بالا، ابتدا یک واحد به متغییر x اضافه شده، سپس آن را در دسترس شی ای که ایجاد شده قرار داده شده است. بنابراین برای حالت پیشوندی به صورت بالا عمل می کنیم.

اما اگر بخواهیم این عملگرها را به صورت پسوندی پیاده سازی کنیم، فقط کافی است که جای این دو کد را باهم عوض کنیم یعنی این که ابتدا شی ایجاد شده را به تک تک متغییرهای قسمت private در دسترس قرار بدهیم سپس در مرحله بعد یک واحد به همه ی آن ها اضافه کنیم. به صورت زیر:

```
r.x=x;    x=x+1;
```

سربار گزاری عملگر افزایشی به صورت پیشوندی برای زمانی که اشاره گر نداریم:

```
class vector
{
private:
        float x, y, z;
public:
        vector operator++();            نوشتن تابع مربوط به سر بارگزاری عملگر در قسمت public
};
vector vector::operator++()             مرحله اول: نوشتن متد مربوط به سربارگزاری عملگر افزایشی
{
        vector r;                       مرحله دوم: ایجاد یک شی با نام r
```

```
        x = x+1;                                    مرحله سوم: انجام عملیلت افزایشی
        y = y+1;                                    ... ... ...
        z = z+1;                                    ... ... ...
        r.x = x;                                    ... ... ...
        r.y = y;                                    ... ... ...
        r.z = z;                                    ... ... ...
        return r;                                   مرحله چهارم: بازگشت شی ایجاد شده:
}
```

مرحله اول : ابتدا عملگری را که قرار است سربارگزاری کنیم در این قسمت می نویسیم.
مرحله دوم : یک شی به نام r به وجود می آوریم.
مرحله سوم : در این مرحله عملیات مربوط به عملگر افزایشی را انجام می دهیم. برای این کار بدین صورت عمل می کنیم: ابتدا متغییر اول قسمت private که در این جا x می باشد را می نویسیم سپس آن را مساوی با خودش قرار می دهیم، با این تفاوت که باید یک واحد به آن اضافه شود. سپس متغییرهای بعدی که y و z می باشند را نیز به ترتیب بدین صورت می نویسیم. وقتی که به تمام متغییرهای قسمت private با این روش یک واحد اضافه کردیم، سپس باید با استفاده از شی ای که درست کردیم و با استفاده از کاراکتر نقطه (.) به تک تک متغییرهای قسمت private دسترسی داشته باشیم.
مرحله چهارم : در این مرحله نیز شی ای را که ایجاد کرده ایم را با استفاده از دستور بازگشتی return بازگشت می دهیم.

پیاده سازی عملگر افزایشی به صورت پسوندی برای زمانی که اشاره گر نداریم:

```
class vector
{
private:
        float x, y, z;
public:
        vector operator++();                        نوشتن تابع مربوط به سر بارگزاری عملگر در قسمت public
};
vector vector::operator++(int x)                    مرحله اول: نوشتن متد مربوط به سربارگزاری عملگر افزایشی
{
        vector r;                                   مرحله دوم: ایجاد یک شی با نام r
        r.x = x;                                    مرحله سوم: انجام عملیلت افزایشی
        r.y = y;                                    ... ... ...
        r.z = z;                                    ... ... ...
        x = x+1;                                    ... ... ...
        y = y+1;                                    ... ... ...
        z = z+1;                                    ... ... ...
        return r;                                   مرحله چهارم: بازگشت شی ایجاد شده:
}
```

مرحله اول : ابتدا تابع مربوط به سربارگزاری این عملگر را می نویسیم.
مرحله دوم : یک شی ایجاد می کنیم.
مرحله سوم : عملیات مربوط به سربارگزاری عملگرافزایشی را پیاده سازی می کنیم که در قسمت های قبل آن را توضیح داده ایم.
مرحله چهارم : در این مرحله نیز، شی ای که ایجاد کرده ایم را با استفاده از دستور بازگشتی return بازگشت می دهیم.

سربار گزاری عملگر کاهشی به صورت پیشوندی برای زمانی که اشاره گر داریم:

```
class vector
{
private:
        float *p;
         float n;
public:
        vector operator--();            نوشتن تابع مربوط به سر بارگزاری عملگر در قسمت public
};
vector vector--()                       مرحله اول: نوشت متد مربوط به سربارگزاری عملگر کاهشی
{
        vector r(n);                    مرحله دوم: ایجاد یک شی به نام r
        for (int i = 0; i < n; i++)     مرحله سوم: نوشتن یک حلقه for
        {
                p[i] = p[i] - 1;        مرحله چهارم: انجام عملیات کاهشی
                r.p[i] = p[i];          ... ... ...
        }
        return r;                       مرحله پنجم: بازگشت شی ایجاد شده
}
```

همان طور که در قسمت بالا می بینید، وقتی که در برنامه اشاره گر وجود داشته باشد، از حلقه ی for استفاده می کنیم و دستورات داخل این قسمت، کوتاه تر می باشد. بنابراین تمام عملیات این قسمت نیز همانند قسمت قبلی می باشد فقط با این تفاوت که به جای متغییرهای قسمت private، از اشاره گر استفاده می کنیم. این نکته را هم نیز به یاد داشته باشید، وقتی که در برنامه اشاره گر داشته باشیم، شی ای را که درست می کنیم باید با استفاده از پرانتز در داخل آن، متغییر قسمت private که همان n (تعداد بعدهای بردار) را بنویسیم.

سربارگزاری عملگر کاهشی به صورت پسوندی برای زمانی که اشاره گر داریم:

```
class vector
{
private:
        float *p;
         float n;
public:
```

```
        vector operator--(int x);          نوشتن تابع مربوط به سر بارگزاری عملگر در قسمت public
};
vector vector--()                          مرحله اول: نوشتن متد مربوط به سربارگزاری عملگر کاهشی
{
        vector r(n);                       مرحله دوم: ایجاد یک شی به نام r
        for (int i = 0; i < n; i++)        مرحله سوم: نوشتن یک حلقه for
        {
                r.p[i] = p[i];             مرحله چهارم: انجام عملیات کاهشی
                p[i] = p[i] - 1;           ... ... ...
        }
        return r;                          مرحله پنجم: بازگشت شی ایجاد شده
}
```


سربارگزاری این نوع نیز، همانند پیشوندی می باشد فقط با این تفاوت که در این جا یک پارامتر از نوع عدد صحیح (int) داریم و فقط کدهای داخل حلقه for را جا به جا می نویسیم. بنابراین دقت کنید که ما این دو عملگر را هم به صورت پیشوندی و هم پسوندی و بدون اشاره گر یا با اشاره گر مورد بررسی قرار دایم، پس برای تمامی کلاس ها نوع سربارگزاری این دو عملگر بدین صورت بود که آن را پیاده سازی کردیم، بنابراین با توجه به این که در برنامه اشاره گر وجود داشت یا نه، دستورات و ساختار هرکدارم را طبق این مطالبی که گفتیم، پیاده سازی می کنیم.

4-3 سربارگزاری عملگرهای مقایسه ای

این عملگر از نوع دو عملوندی می باشد، بنابراین دارای پارامتر می باشد. عملگر مقایسه ای بر خلاف بقیه عملگرها، دارای نوع بازگشتی int می باشد. برای انجام مقایسه بر روی دو شی ابتدا باید عملیات محاسباتی بر روی آن ها انجام دهیم، سپس آن ها را با هم مورد مقایسه قرار دهیم. بنابراین برای استفاده از این عملگر به یک تابع محاسباتی به نام calculate نیاز داریم که با توجه به نیاز خود، محاسبات مناسب را بر روی اشیاء انجام داده و سپس در مرحله دوم، متد مربوط به سربارگزاری عملگر مقایسه ای را می نویسیم و در داخل این متد، اشیایی را که در داخل متد calculate مورد محاسبه قرار دادیم را با هم مقایسه می کنیم. دقت کنید که برای انجام عملیلت مقایسه بین دو شی، در صورت بزرگ بودن شی اول نسبت به شی دوم، مقدار 1، در غیر این صورت مقدار 0 بازگشت داده می شود. به خاطر همین است که نوع بازگشتی این متد به صورت int می باشد. پس برای سربارگزاری این عملگر، به یک تابع محاسباتی به نام calculate نیز نیاز داریم. دقت کنید که در داخل متد calculate، بستگی به عملیاتی که قرار است انجام بدهیم، دستورات را پیاده سازی می کنیم. مثلا در این قسمت می خواهیم در داخل متد calculate، اندازه ی بردارها را بدست آوریم. برای بدست آوردن اندازه هر بردار، کافی است طبق فرول زیر عمل کنیم:

$$\sqrt{x * x + y * y + z * z + \cdots}$$

بنابراین باید تک تک بعدهای هر بردار را که در این جا x و y و z می باشند را بدین صورت در زیر رادیکال نوشت.

نوشتن یک متد calculate برای استفاده در سربارگزاری عملگر مقایسه ای: (بدون اشاره گر)

```
class vector
```

```
{
private:
        float x, y, z;
public:
        float calculate();                          نوشتن یک تابع محاسباتی به نام: calculate
};
float vector::calculate()                           مرحله اول: نوشتن متد مربوط به انجام عملیات محاسباتی
{
        float s;                                    مرحله دوم: ایجاد یک متغییر به نام s
        s = (x*x)+(y*y) +(z*z);                     مرحله سوم: محاسبه ی اندازه بردارها
        return sqrt(s);                             مرحله چهارم: بازگشت دادن اندازه بردارها
}
```

مرحله اول : در این قسمت، متد مورد نظر را بدین صورت مینویسیم و در قسمت های بعدی کدهای مربوط به آن را پیاده سازی می کنیم.

مرحله دوم : برای ذخیر کردن مقدار بردارها، ابتدا باید یک متغییر به وجود آوریم و حاصل فرمول قبل را در آن بریزیم.

مرحله سوم : در این قسمت حاصل فرمول قبل را بدست می آوریم. دقت کنید که ما در قسمت private سه متغییر داریم، بنابراین باید طبق فرمول قبل ابتدا تک تک این متغییرها را در هم ضرب کنیم، سپس حاصل جمع آن ها را در متغییر s بریزیم.

مرحله چهارم : در این مرحله نیز از حاصل جمع همه ی متغییرهایی که در متغییر s ریخته شده اند با استفاده از تابع sqrt() جذر می گیریم و با استفاده از دستور بازگشتی return این مقدار رادیکالی را بازگشت می دهیم.

نکته 3-12 : دقت کنید که برای جذر گرفتن یک مقدار، از تابع sqrt() موجود در فایل کتابخانه ای math.h استفاده می کنیم. بدین صورت:

```
#include<math.h>
```

بنابراین باید این فایل کتابخانه ای را نیز به قسمت اول برنامه اضافه کنیم.

سربارگزاری عملگر مقایسه ای برای زمانی که اشاره گر نداشتیم:

```
class vector
{
private:
        float x, y, z;
public:
        int operator<(vector rop);                  نوشتن متد مربوط به سربارگزاری عملگر در قسمت public
};
int vector::operator<(vector rop)                   مرحله اول: نوشتن متد مربوط به سربارگزاری عملگر
{
        if (calculate() < rop.calculate)            مرحله دوم: انجام عملیات مقایسه ای
        return 1;       مرحله سوم: در صورت درست بودن عملیات مقایسه ای، عدد 1 بازگشت داده می شود
        return 0;       مرحله چهارم: در صورت اشتباه بودن عملیات مقایسه ای، عدد 0 بازگشت داده می شود ذ
}
```

مرحله اول : ابتدا متد مربوط به عملگر مقایسه ای را بدین صورت می نویسیم.
مرحله دوم : در این قسمت چک می کنیم که اگر شی اول کوچکتر از شی دوم بود، مقدار 1، در غیر این صورت مقدار 0 بازگشت داده شود. برای این کار ما باید متد محاسباتی یا همان calculate را مورد مقایسه قرار دهیم، چون اندازه بردار را در داخل این متد محاسبه کرده ایم. برای این کار از یک دستور شرطی if استفاده می کنیم و در داخل آن، ابتدا متد calculate را می نویسیم سپس با استفاده از شی ای که به صورت پارامتر فرستاده ایم، متد calculate را با استفاده از کاراکتر نقطه (.) در دسترس خود قرار می دهیم و این دو را با هم مورد مقایسه قرار می دهیم.
دقت کنید که ما در این جا عملگر (>) را نوشته ایم که این عملگر به معنی (کوچک تر) می باشد. عملگر بعدی، <) (به معنای (بزرگ تر) می باشد. بنابراین باید در هنگام پیاده سازی آن ها را اشتباه ننویسیم.
مرحله سوم : چون در این جا ما از عملگر مقایسه ای کوچکتر (>) استفاده می کنیم، بنابراین اگر شی اول از شی دوم کوچک تر بود، مقدار 1 بازگشت داده می شود.
مرحله چهارم : اما اگر شی اول از شی دوم کوچک تر نبود، مقدار 0 بازگشت داده می شود.

نکته 13-3 : تقریبا در سربارگزاری اکثر عملگرها، شی ای را که در داخل تابع به وجود می آوریم را با استفاده ازدستور بارگشتی return بازگشت می دهیم. اما برای سربارگزاری برخی از عملگرها، شی به وجود نمی آوریم و به جای آن، اشاره گر this را بدین صورت بازگشت می دهیم: return *this;

اشاره گر this : این اشاره گر در زمان فراخوانی اشیاء به طور اتوماتیک به آن ها فرستاده می شود و حاوی اطلاعات شی فراخواننده می باشد. بنابراین در سربارگزاری بعضی از عملگرهایی که به دلایلی قادر به ایجاد شی نیستم، می توانیم از این اشاره گر استفاده کنیم و آن را با استفاده از دستور بازگشتی return بازگشت دهیم. مثلا برای سربارگزاری عملگر نسبت دهی، از این روش استفاده می کنیم.

5-3 سربارگزاری عملگر نسبت دهی

دقت کنید که خود C++ به طور پیش فرض عملیات نسبت دهی را برای ما انجام داده است، اما در مواقی که اشاره گر نداشتیم، لازم نیست که این عملگر را سربارگزاری کنیم و از همان تعریف پیش فرض C++ استفاده می کنیم. اما اگر در برنامه ما اشاره گر وجود داشت، باید سربارگزای عملگر نسبت دهی را حتما انجام بدهیم، یعنی باید تعریف پیش فرض C++ را لغو کنیم و خود ما این عملیات را با استفاده از سربارگزاری این عملگر انجام دهیم. عملگر نسبت دهی نیز دو عملوندی می باشد، اما در رابطه با این عملگر باید به چند نکته زیر دقت کرد:

نکته 14-3 : این عملگر برای سربارگزاری، در متد خود نیاز به یک کاراکتر به نام refrence (&) دارد که به صورت زیر سربارگزاری می شود:

سربارگزاری عملگر نسبت دهی برای وقتی که اشاره گر نداشتیم:

```
class vector
{
private:
        float x, y, z;
public:
        vector & operator=(vector & ob);
```

نوشتن متد مربوط به سربارگزاری عملگر در قسمت: public

```
};
polynomial::polynomial & operator=(polynomial & ob)        مرحله اول: نوشتن متد مربوط به
{                                                          سربارگزاری عملگر نسبت دهی
        x = ob.x;                                          مرحله دوم: انجام عملیات نسبت دهی
        y = ob.y;                                          ... ... ...
        z = ob.z;                                          ... ... ...
        return *this;                                      مرحله سوم: بازگشت شی توسط اشاره گر *this
}
```

همان طور که گفتیم، وقتی در برنامه اشاره گر نداشتیم، نیازی به سربارگزاری عملگر نسبت دهی نیست. اما برای این که با نحوه سربارگزاری این عملگر آشنا شوید، آن را سربارگزاری کرده ایم. فقط هنگامی که اشاره گر داشتیم، باید عملگر نسبت دهی را سربارگزاری کنیم. همان طور که قبلا هم گفتیم، در اینجا ما از کاراکتر refrence (&) استفاده می کنیم، بدین صورت که همیشه برای سربارگزاری عملگر نسبت دهی، یک refrence را قبل از اسم کلاس و یکی را نیز قبل از پارامتر قرار می دهیم.

مرحله اول : ابتدا این متد را بدین شکل می نویسیم تا ساختار آن را پیاده سازی کنیم.
مرحله دوم : در این قسمت، عملیات نسبت دهی را انجام می دهیم، بدین صورت عمل می کنیم: تک تک متغییرهای قسمت private را می نویسیم، سپس کاراکتر مساوی قرار می دهیم و با استفاده از پارامتری که در تابع وجود دارد، که در این جا ob می باشد، تک تک متغییرهای قسمت private با استفتده از کاراکتر نقطه (.) در دسترس خود قرار می دهیم.
مرحله سوم : در این مرحله چون شی به وجود نیاورده ایم، باید اشاره گر this را با استفاده از دستور بازگشتی return بازگشت دهیم.

سربارگزاری عملگر نسبت دهی برای زمانی که در برنامه اشاره گر وجود داشت:

```
class vector
{
private:
        float *p;
        float n;
public:
        vector & operator=(vector & ob);    نوشتن متد مربوط به سربارگزاری عملگر در قسمت: public
};
polynomial & polynomial::operator=(polynomial & ob)        مرحله اول: نوشتن متد مربوط به
{                                                          سربارگزاری عملگر نسبت دهی
        if (n != ob.n)                                     مرحله دوم: نوشتن یک دستور if
        {
                delete[]p;                                 مرحله سوم: از بین بردن حافظه اشاره گر قبلی
                p = new float[ob.n];                       مرحله چهارم: ایجاد یک حافظه جدید با استفاده از آرایه های پویا
        }
        if (!p)                                            مرحله پنجم: یک شرط if قرار می دهیم و در داخل آن
        {
                cout << "Erorr!";                          یک پیغام " Erorr " چاپ می کنیم
```

```
        exit(1);                    و با استفاده از دستور exit(1) از برنامه خارج می شویم
    }
    n = ob.n;                       مرحله ششم: متغییرداخل private را مساوی با پارامتر متد قرار می دهیم
    for (int i = 0; i < ob.n; i++)  مرحله هفتم: یک حلقه ی for می نویسیم
    {
        p[i] = ob.p[i];             مرحله هشتم: تمام داده ها را مساوی با پارامتر متد قرار می دهیم
    }
    return *this;                   مرحله نهم: بازگشت اشاره گر this
}
```


به این نکته مهم دقت کنید که ساختار عملگر نسبت دهی دقیقا مانند ساختار سازنده کپی می باشد با این تفاوت که در این جا دو خط به به ساختار سازنده کپی اضافه شده و با نوشتن این دو خط در ابتدای سازنده کپی، می توان عملگر نسبت دهی را به راحتی سربارگزاری کرد.

مرحله اول : در ابتدا،این متد را بدین صورت می نویسیم و در مرحله های بعد ساختار آن را پیاده سازی می کنیم.
مرحله دوم : در این مرحله چک می کنیم که اگر متغییر قسمت private که همان n (تعداد بعدهای بردار)، مخالف با پاارمتر داخل متد بود، در مرحله بعد آن را از بین ببرد.
مرحله سوم : در این قسمت، حافظه ای که اشاره گر برنامه که در این جا p نام دارد به آن اشاره می کند را با استفاده از دستور delete، از بین می بریم.
مرحله چهارم : در این مرحله یک حافظه جدید به اندازه پارامتر فرستاده شده با استفاده از آرایه های پویا استفاده می کنیم. (روش ایجاد حافظه توسط آرایه های پویا را در مطالب قبلی گفته ایم).
مرحله پنجم : دراین مرحله یک دستور if می نویسیم و در داخل آن چک می کنیم که اگر حافظه ای به اشاره گر اختصاص داده نشده باشد یا این که اشاره گر وجود نداشته باشد، یک پیغام " Erorr " چاپ و با استفاده از تابع exit(1) از برنامه خارج می شویم.
مرحله ششم : سپس تعداد بعدهای بردار که در این جا n می باشد را مساوی پارامتر تابع قرار می دهیم.
مرحله هفتم : یک حلقه for می نویسیم و از صفر تا آخرین بعد بردار یعنی تا ob.n را پیمایش می کنیم.
مرحله هشتم : تمام داده ها را برابر پارامتر تابع قرار می دهیم.
مرحله نهم : و در آخر نیز با استفاده از دستور بازگشتی return، اشاره گر this را بازگشت می دهیم.

همیشه برای سربارگزاری عملگر نسبت دهی، به صورت قبل عمل می کنیم، بنابراین برای تمامی کلاس هایی که بخواهیم این عملگر را سربارگزاری کنیم، به صورت قبل آن را پیاده سازی می کنیم.

6-3 سربار گزاری عملگرهای ترکیبی

سربارگزاری این عملگرها هم از قاعده ی خاصی پیروی می کند که با توجه به این که در برنامه اشاره گر وجود داشته باشد یا نه، ساختار هرکدام را پیاده سازی می کنیم.
عملگر های ترکیبی شامل (جمع و تفریق و ضرب و تقسیم) می باشند که این عملگرها بدین صورت نوشته می شوند:

```
int a=20, b=15;     a+=b;     a-=b;     a*=b;     a/=b;
```

ای عملگرها دقیقا مانند عملگرهای (جمع و تفریق و ضرب و تقسیم) عمل می کنند فقط با این تفاوت که حاصل نتیجه را در عملگر دوم می ریزد. این عملگر به یکی از دو صورت زیر قابل استفاده قرار می گیرد:

```
vector v1(3, 2), v2(4, 1);
v1 += v2;
```
روش اول: جمع بردار v1 با بردار v2 و قرار دادن مقدار نتیجه در بردار v1

```
vector v1(3, 2);
v1 += 6;
```
روش دوم: جمع بردار v1 با عدد 6 و قرار دادن مقدار نتیجه در بردار v1

سربارگزاری عملگر ترکیبی =+ برای جمع دو بردار با هم. (بدون اشاره گر).

```
class vector
{
private:
        int x, y, z;
public:
        vector operator+=(vector ob1);
};
```
نوشتن تابع مربوط به سر بارگزاری عملگر در قسمت public

```
vector vector::operator+=(vector ob1)
{
        x = ob1.x + x;
        y = ob1.y + y;
        z = ob1.z + z;
        return *this;
}
```
مرحله اول: نوشتن متد مربوط به سربارگزاری عملگر ترکیبی

مرحله دوم: سربارگزاری عملگر ترکیبی

... ...

... ...

مرحله سوم: بازگشت اشاره گر this

در سربارگزاری بالا، ما روش اول را مورد بررسی قرار داده ایم. یعنی این که دو بردار را با هم جمع کرده ایم و نتیجه را در بردار اول ریخته ایم. بنابراین پارامتر ما از نوع خود کلاس یعنی vector می باشد.

مرحله اول : در ابتدا تابع مربوط به سربارگزاری عملگر را بدین صورت می نویسیم.
مرحله دوم : در این مرحله عملیات مربوط به سربارگزاری عملگر جمع به صورت ترکیبی را انجام می دهیم. برای این کار ما باید ابتدا تک تک متغییرهای داخل قسمت private را بنویسیم و سپس کاراکتر مساوی را قرار بدهیم و بعد پارامتر متد را که از نوع خود کلاس می باشد را می نویسیم و با ستفاده از کاراکتر نقطه (.) آن را به تک تک متغییرهای قسمت private در دسترس خود قرار می دهیم و سپس کاراکتر مورد نظر و در ادامه دوباره متغییر داخل قسمت private را می نویسیم. بنابراین به هر تعداد که متغییر در قسمت private داشته باشیم، این عملیات را باید انجام بدهیم.
مرحله سوم : چون شی ای را به وجود نیاورده ایم، بنابراین باید اشاره گر this را بازگشت بدهیم.

سربارگزاری عملگر ترکیبی =- برای تفریق یک بردار از یک عدد. (بدون اشاره گر)

```
class vector
{
private:
        int x, y, z;
public:
        vector operator-=(int a);
```
نوشتن تابع مربوط به سر بارگزاری عملگر در قسمت public

```
};
vector vector::operator-=(int a)          مرحله اول: نوشتن متد مربوط به سربارگزاری عملگر ترکیبی
{
        x = x - a;                        مرحله دوم: سربارگزاری عملگر ترکیبی
        y = y - a;                        ... ... ...
        z = z - a;                        ... ... ...
        return *this;                     مرحله سوم: بازگشت اشاره گر this
}
```

در سربارگزاری بالا ما روش دوم را مورد بررسی قرار داده ایم، یعنی این که یک بردار را با یک عدد جمع کرده ایم و نتیجه را در بردار ریخته ایم. بنابراین پارامتر ما باید از نوع خود کلاس نباشد، مثلا در این جا ما پارامتر را به صورت عدد صحیح در نظر گرفته ایم. این بدین معنی می باشد که یک بردار را با یک عدد صحیح می خواهیم جمع کنیم. دستورا پیاده سازی این عملگر به صورت قبل می باشد فقط با این تفاوت که در این جا دیگر شی نداریم که آن را با استفاده از کاتراکتر نقطه (.) در دسترس خود قرار دهیم.

سربارگزاری عملگر ترکیبی =* برای ضرب دو بردار با هم. (همراه اشاره گر).

```
class vector
{
private:
        float *a;
        int n;
public:
        vector operator*=(vector ob1);    نوشتن تابع مربوط به سربارگزاری عملگر در قسمت public
};
vector vector::operator*=(vector ob1)     مرحله اول: نوشتن متد مربوط به سربارگزاری عملگر ترکیبی
{
        for (int i = 0; i < n; i++)       مرحله دوم: نوشتن یک حلقه ی for
        {
                a[i] = ob1.a[i] * a[i];   مرحله سوم: سربارگزاری عملگر ترکیبی
        }
        return *this;                     مرحله چهارم: بازگشت اشاره گر this
}
```

همان طور که مشاهده می کنید، وقتی که در برنامه اشاره گر داشته باشیم برنامه ما کمی تغییر خواهد کرد و از یک حلقه ی for نیز برای شمارش تعداد اشیاء استفاده می کنیم. در برنامه بالا قرا است که دو شی را با استفاده از عملگر ترکیبی ضرب، در هم ضرب کنیم. بنابراین باید پارامتر ما از نوع خود کلاس یعنی از نوع vector باشد.

سربارگزاری عملگر =/ برای تقسیم یک بردار بر یک عدد. (همراه اشاره گر)

```
class polynomial
{
private:
        float *a;
```

```
        int d;
public:
        polynomial operator/=(int b);
};
polynomial polynomial::operator/=(int b)
{
        for (int i = 0; i < d; i++)
        {
                a[i] = a[i] / b;
        }
        return *this;
}
```

در این جا نیز اشاره گرها وجود دارند و مانند مثال قبل دستورات را پیاده سازی می کنیم. چون پارامتر فرستاده شده از نوع عدد صحیح می باشد بنابراین می فهمیم که باید شی ای را بر یک عدد صحیح تقسیم کنیم.

7-3 تابع دوست (friend)

ابتدا به سوال زیر جواب می دهیم تا فهم موضوع برای ما آسان شود. مثلا طبق ()main داده شده، ما باید چه عملگرهایی را سربارگزاری کنیم؟

```
int main()
{
        vector v1, v2, v3;                  ایجاد سه شی به نام های v1 و v2 و v3
        cin >> v1 >> v2 >> v3;              دریافت هر سه شی با استفاده از دستور ورودی cin
        v3 = v1 + 12;                       جمع کردن تمام بعدهای بردار v1 با عدد 12
        v3 = 8 + v1;                        جمع کردن عدد 8 با تمام بعدهای بردار v1
        cout << v3;                         نمایش شی v3 با استفاده از دستور خروجی cout
        getch();
        return 0;
}
```

در مطالب قبل هم گفتیم که، هرگاه در قسمت ()main، بر روی اشیاء عملیات محاسباتی انجام شود یا این که آن ها را با استفاده از دستور ورودی cin از صفحه کلید دریافت یا این که آن ها را با استفاده از دستور خروجی cout در خروجی نمایش داد، باید دقیقا همان عملگر را سربارگزاری کنیم. در قسمت ()main بالا نیز، در خط اول سه شی به وجود آمده است سپس در خط دوم این سه شی از صفحه کلید گرفته می شوند، بنابراین باید عملگر cin را سربارگزاری کنیم. در خط سوم، تمام بعدهای بردار v1 با عدد 8 جمع شده است، بنابراین باید عملگر جمع را سربارگزاری کنیم. در خط چهارم نیز این همان دستور مربوط به خط قبل می باشد با این تفاوت که این دفعه ابتدا عدد 8 نوشته شده است سپس این عدد را با همه ی بعدهای بردار v1 جمع کرده است. حالا چه عملگری را باید سربار گزاری کرد؟ برای جواب به این سوال و آشنایی با عملگر دوست، به نکته ی زیر دقت کنید.

نکته 3-15 : اگر در قسمت ()main مثلا برای عملگر ضرب، مانند خط چهارم مثال قبل نوشته شده باشد، یعنی این که ابتدا عددی را با شی ای مورد جمع یا .. قرار دهد، برای این نوع عملگرها، باید از تابع دوست استفاده کنیم.
بنابراین اگر داده ی اول از نوع کلاس نباشد، باید با استفاده از تابع دوست سربارگزاری را نجام داد.
همان طور که مشاهده می کنید، برای استفاده ازتابع دوست، باید به داده ی اول نگاه کنیم، اگر این داده جزء همان اشیایی بود که به وجود آورده ایم، پس نباید از تابع دوست استفاده کنیم. اما اگر داده ی اول از نوع خود کلاس نبود، باید از تابع دوست استفاده کنیم.

تابع دوست لازم نیست: v3=v1 + 14; یا v2=v3 / 6;

تابع دوست لازم است: v1=12 + v2; یا cin>> v1; یا cout<< v3;

نکته 3-16 : همیشه برای سربارگزاری عملگرهای cin و cout باید از تابع دوست استفاده می کنیم. چون این دو عملگر به صورت زیر عمل می کنند:
cin>> v1;، داده ی اول از نوع جریان ورودی cin می باشد
cout<< v1;، داده ی اول از نوع جریان خروجی cout می باشد.
بنابراین چون داده ی اول این دوعملگر عضو کلاس نیستند، پس حتما باید این دو عملگر را به صورت تابع دوست پیاده سازی کنیم.

نکته 3-17 : تابع دوست عضو کلاس نیست، بنابراین در خارج از کلاس نیز قابل استفاده می باشد. دقت کنید که در سربارگزاری عملگرهای قبلی، عملگرهای دو عملوندی دارای یک پارامتر و عملگرهای تک عملوری فاقد پامتر بودند، اما وقتی که از تابع دوست برای سربارگزاری عملگرها استفاده می کنیم، یک پارامتر بیش تر از حالت قبل داریم. یعنی این که برای سربارگزاری عملگرهای تک عملوندی، از یک پارامتر و برای سربارگزاری عملگرهای دو عملوندی، از دو پارامتر استفاده می کنیم.

نکته 3-18 : برای استفاده از تابع دوست در سربارگزاری عملگرها، فقط کافی است که کلمه ی کلیدی friend را در ابتدای متد سربارگزاری عملگر بنویسیم. مثلا برای سربارگزاری عملگر جمع به صورت زیر عمل می کنیم:

```
friend vector  operator+(int  ob1, vector ob2);
```

نکته 3-19 : در سربارگزاری عملگرهای cin و cout، همانند سربارگزاری عملگر نسبت دهی، باید از کاراکتر refrence (&) استفاده کنیم. اما این دفعه برای سربارگزاری cin از سه refrence و برای cout از دو refrence استفاده می کنیم. برای cin یکی را قبل از کلمه ی کلیدی operator، و برای هر دو پارامتر هم یک refrence قرار می دهیم.
اما برای cout از دو refrence استفاده می کنیم که اولی را قبل از کلمه ی کلیدی operator و refrence بعدی را نیز در پارامتر اول می نویسیم. پس برای سربارگزاری cout، پارامتر دوم را بدون refrence می نویسیم.

نکته 3-20 : در سربازگزاری عملگرها به صورت تابع دوست، اگر خواستیم عملگرهای دو عملوندی را سربارگزاری کنیم باید دقت کرد که: پارامتر اول همان نوع داده ی اول و پارامتر دوم همان نوع داده ی دوم می باشد. به صورت زیر:

```
friend vector  operator+(int  ob1, vector ob2);
```

خط بالا بدین معنی می باشد که داده ی اول از نوع عدد صحیح با همان int می باشد و داده ی دوم از نوع خود کلاس می باشد. بدین صورت که عددی را با شی ای جمع کرده ایم.

نکته 3-21 : برای سربارگزاری عملگرها به کمک تابع دوست، از عملگر حوضه دید (::) استفاده نمی کنیم. چون توابع دوست عضو کلاس نیستند بنابراین نباید از این عملگر استفاده کرد. فقط برای توابعی می توان از این عملگر استفاده کرد که عضو کلاس باشند.

3-8 سربارگزاری عملگر های cin و cout

در مطالب قبلی هم گفتیم که فایل کتابخانه ای مربوط به cin و cout، iostream می باشد که باید این فایل کتابخانه ای را به ابتدای همه ی برنامه های خود اضافه کنیم. عملگر cin از نوع جریان ورودی istream و عملگر cout از نوع جریان خروجی cout می باشد. بنابراین در تابع دوست برای سربارگزاری عملگرهای cin و cout، پارامتر اول همیشه از نوع iostream یا ostream و پارامتر دوم از نوع خود کلاس می باشد. بنابراین برای تعریف عملگرهای cin و cout به کمک تابع دوست، آن ها را به صورت زیر می نویسیم:

```
friend istream & operator>>(istream & I, vector & ob1);        :برای سربارگزاری cin
friend ostream & operator<<(ostream & O, vector ob2);          :برای سربارگزاری cout
```

سربارگزاری عملگر cin با استفاده از تابع دوست: (بدون اشاره گر)

```
class vector
{
private:
        float x, y, z;
public:
        friend istream & operator>>(istream & I, vector & ob1);   نوشتن متد مربوط به سربارگزاری
};                                                                عملگر در قسمت public
istream & operator>>(istream & I, vector & ob1)                   مرحله اول: نوشتن متد مربوط به
{                                                                 سربارگزاری عملگر cin
        I >> ob1.x;                                               مرحله دوم: انجام عملیات مربوط به دریافت داده ها
        I >> ob1.y;                                               ... ... ...
        I >> ob1.z;                                               ... ... ...
        return I;                                                 مرحله سوم: بازگشت شی I
}
```

نکته 3-22 : در قسمت پیاده سازی کلاس یا همان " قسمت دوم "، نباید کلمه ی کلیدی friend را بنویسیم.

مرحله اول : در ابتدا تابع را بدین صورت می نویسیم و در مرحله های بعد، ساختار مربوط به آن را پیاده سازی می کنیم. همان طور هم که در مطالب قبل گفتیم، در این قسمت یعنی در " قسمت دوم " نیازی به نوشتن عملگر حوضه دید (::) و کلمه ی کلیدی friend نمی باشد. در این قسمت نیز سه refrence قرار داده ایم و تابع ما نیز چون یک تابع دوست می باشد، برای عملگر cin که دو عملوندی است، دو پارامتر قرار داده ایم. این نکته را هم در قبل گفتیم که: همیشه برای سربارگزاری عملگرهای cin و cout، از دو پارامتر استفاده می کنیم که پارامتر اول باید نوع جریان ورودی که در این جا iostream می باشد، باشد و پارامتر دوم باید از نوع خود کلاس یعنی vector باشد.
مرحله دوم : در این مرحله باید عملیات مربوط به دریافت داده ها را انجام بدهیم. قبلا برای دریافت یک عدد از

ورودی بدین صورت عمل می کردیم: vector v1; cin>> v1;
همان طور که در کد بالا می بینید، نوع داده ی اول cin می باشد که قبلا هم گفتیم این عملگر از نوع جریان ورودی iostream و داده ی دوم از نوع خود کلاس می باشد. به خاطر همین است که پارامترهای عملگر cin را به ترتیب اولی را iostream و دومی را از نوع خود کلاس معرفی کرده ایم. مثلا در این مثال پارامتر iostream را I و پارامتر خود کلاس را ob1 نام گزاری کرده ایم، بنابراین به جای عملگر cin پارامتر آن یعنی I و به جای شی، پارامتر آن را یعنی ob1 می نویسیم. به صورت زیر:


```
cin>> v1;
I  >> v1;
```

به این نکته دقت کنید که با توجه به عملگرهای دیگری که تا حالا سربارگزاری کرده ایم، باید شی ob1 به تک تک متغییرهای قسمت private با استفاده از کاراکتر نقطه (.) دسترسی داشته باشد.
مرحله سوم : دقت کنیدکه نوع بازگشتی این متد همیشه به صورت iostream می باشد، بنابراین باید پارامتر آن که I نام دارد را بازگشت دهیم.

سربارگزاری عملگر cout با استفاده از تابع دوست: (بدون اشاره گر)

```
class vector
{
private:
        float x, y, z;
public:
        friend ostream & operator<<(ostream & O, vector  ob2);
};
ostream & operator<<(ostream & O, vector ob2)
{
        O << ob2.x;
        O << ob2.y;
        O << ob2.z;
        return O;
}
```

نوشتن متد مربوط به سربارگزاری عملگر در قسمت: public
مرحله اول: نوشتن متد مربوط به سربارگزاری عملگر cout
مرحله دوم: انجام عملیات مربوط به نمایش داده ها
... ...
... ...
مرحله سوم: بازگشت شی O

سربارگزاری عملگر cout نیز دقیقا مانند سربارگزاری عملگر cin می باشد فقط با این تفاوت که این بار به جای iostream از جریان خروجی ostream استفاده می کنیم و طبق مطالبی که در قبل گفتیم، در این جا از دو refrence استفاده می کنیم.

مرحله اول : ابتدا بدبن صورت آن را می نویسیم. دقت کنید که در این قسمت نیازی به نوشتن عملگر حوضه دید (::) و کلمه ی کلیدی friend نمی باشد.
مرحله دوم : در این قسمت نیز عملیات مربوط به نمایش داده ها را انجام می دهیم. برای این کار نیز همانند عملگر cin عمل می کنیم فقط با این تفاوت که به جای عملگر (<<) از عملگر (>>) استفاده می کنیم. در این قسمت پارامتر ostream، O و پارامتر خود کلاس، ob2 می باشد. و باید شی پارامتر را با استفاده از کاراکتر نقطه (.) به تک تک متغییرهای قسمت private در دسترس خود قرار دهیم.
مرحله سوم : دقت کنیدکه نوع بازگشتی این متد همیشه به صورت ostream می باشد، بنابراین باید پارامتر آن را که O نام دارد را بازگشت دهیم.

سربارگزاری عملگر cin با استفاده از تابع دوست: (همراه اشاره گر)

```
class vector
{
private:
        float *p;
        int n;
public:
        friend istream & operator>>(istream & I, vector & ob1);
};
istream & operator>> (istream & I, vector & ob1)
{
        for (int i = 0; i < ob1.n; i++)
        {
                I >> ob1.p[i];
        }
        return I;
}
```

نوشتن متد مربوط به سربارگزاری عملگر در قسمت: public

مرحله اول: نوشتن متد مربوط به سربارگزاری عملگر cout

مرحله دوم: نوشتن یک حلقه for

مرحله سوم: انجام عملیات مربوط به دریافت داده ها

مرحله چهارم: بازگشت شی I

مرحله اول : ابتدا متد این عملگر را بدین صورت می نویسیم.
مرحله دوم : همان طور که در مطالب قبل هم گفتیم، هروقت اشاره گر داشته باشیم، از حلقه ی for استفاده می کنیم و اندیس شروع حلقه را از صفر تا اخرین شی را پیمایش می کنیم.
مرحله سوم : در مطالب قبل هم گفتیم که وقتی اشاره گر داریم، تمام داده های ما در داخل اشاره گر قرار می گیرند و برای انجام محسبات، باید این عملیات را بر روی اشاره گرها انجام دهیم.

وقتی اشاره گر نداشتیم: I>> ob1.x;

وقتی اشاره گر داشتیم: for (int i = 0; i < ob1.n; i++) { I >> ob1.p[i]; }

مرحله چهارم : دقت کنیدکه نوع بازگشتی این متد همیشه به صورت istream می باشد، بنابراین باید پارامتر آن را که I نام دارد را بازگشت دهیم.

سربارگزاری عملگر cout با استفاده از تابع دوست: (همراه اشاره گر)

```
class vector
{
private:
        float *p;
        int n;
public:
        friend ostream & operator<<(ostream & O, vector  ob2);
};
ostream & operator<< (ostream & O, vector  ob2)
{
        for (int i = 0; i < ob2.n; i++)
```

نوشتن متد مربوط به سربارگزاری عملگر در قسمت: public

مرحله اول: نوشتن متد مربوط به سربارگزاری عملگر cout

مرحله دوم: نوشتن یک حلقه for

```
        {
                O << ob2.p[i];                    مرحله سوم: انجام عملیات مربوط به نمایش داده ها
        }
        return O;                                 مرحله چهارم: بازگشت شی O
}
```


دقت کنید که این عملگر نیزدقیقا مانند عملگر cin سربارگزاری می شود با این تفاوت که به جای iostream از جریان خروجی ostream استفاده می کنیم. دقت کنید که در این قسمت، از دو refrence استفاده می کنیم.

فصل چهارم: ارث بری

ارث بری یکی از اصول مهم شی گرایی می باشد که با استفاده از این روش می توان یک کلاس مبنا را به وجود آورد و کلاس های بعدی ویژه گی های کلاس مبنا را به ارث ببرند. به کلاسی که آن را به وجود می آوریم تا بقیه ی کلاس ها ویژه گی خود را از آن به ارث ببرند، کلاس مبنا یا کلاس پایه گفته می شود.
به کلاسی که ویژه گی های خود را از کلاس مبنا به ارث می برد، کلاس مشتق شده می گویند.
کلاسی که می خواهد ویژه گی های خود را از کلاس مبنا یا پایه به ارث ببرد، باید نحوه این ارث بری را نیز مشخص کند. دقت کنید که کلاس مشتق شده نمی تواند به اعضای خصوصی (private) کلاس مبنا یا پایه دسترسی داشته باشد و فقط می تواند به اعضای عمومی (public) کلاس مبنا دسترسی داشته باشد.

نحوه ارث بری یک کلاس از کلاس دیگر به صورت زیر می باشد:

```
#include<iostream.h>
#include<conio.h>
class A                                   کلاسی به نام A که همان کلاس مبنا می باشد
{
private:
        float x;                          کلاس مبنا متغییری خصوصی به نام x دارد
public:
        void insert();                    کلاس مبنا متدی به نام insert دارد
        void print();                     کلاس مبنا متدی به نام print دارد
};

class B:public A          کلاسی به نام B که نحوه ارث بری را به صورت public از کلاس A به ارث می برد
{
private:                                  اعضای خصوصی کلاس مشتق شده
        … … …;
public:                                   متدهای کلاس مشتق شده
        … … …;
};
int main()                                تابع main()
```

```
{
        B ob1;                          شی به نام ob1 از نوع کلاس B
        ob1.insert();     دسترسی به تمام متدهای کلاس مبنا با استفاده از شیob1
        ob1.print();                    ... ... ...
        getch();
        return 0;
}
```


توضیح کلاس : برای ارث بری ابتدا یک کلاس را به عنوان کلاس مبنا به وجود می آوریم و سپس کلاس های دیگری را به وجود می آوریم تا ویژه گی های کلاس مبنا را به ارث ببرند. در مثال بالا نیز همین کار را کرده ایم، یعنی این که ابتدا یک کلاس مبنا به نام کلاس A به وجود آمده است که داری یک متغییر و چندین متد می باشد و کلاس دیگری به نام کلاس B نیز پیاده سازی شده است که ویژه گی های کلاس مبنا را به صورت public به ارث می برد.

دقت کنید که کلاس مشتق شده نمی تواند اعضای خصوصی کلاس مبنا را به ارث ببرد. وقتی که نحوه ارث بری را به صورت public معرفی می کنیم یعنی این که تمام متغییرها و متدهای قسمت public کلاس مبنا به متد های قسمت public کلاس مشتق شده اضافه می شوند. واگر نحوه ارث بری را به صورت private معرفی کنیم بدین معنی می باشد که تمام متغییرها و متدهای قسمت public کلاس مبنا به قسمت خصوصی یا private کلاس مشتق شده اضافه می شوند. برنامه بالا نحوه ارث بری را به صورت public معرفی کرده است و این بدین معنی می باشد که تمام متغییرها و متدهای قسمت public کلاس مبنا به متدهای کلاس مشتق شده نیز اضافه می شوند. بنابراین کلاس مشتق شده نیز می تواند به متدهای ()insert و ()print نیز دسترسی داشته باشد.

مثال 4-1 : کلاسی به نام A که کلاس پایه می باشد را به وجود آورید و کلاسی دیگر به نام B ایجاد کنید که خصوصیات کلاس A را به صورت public به ارث ببرد.

```
#include<iostream.h>
#include<conio.h>
class A                                 ایجاد کلاسی به نام A
{
private:
        float x, y;                     متغییر های خصوصی کلاس مبنا
public:
        void insert();                  متدهای کلاس مبنا
        void print();                   ... ... ...
};
void A::insert()                        پیاده سازی متد ()insert
{
        cout << "enter x,y: ";
        cin >> x >> y;
}
void A::print()                         پیاده سازرزی متد ()print
{
        cout << "x: " << x << endl;
        cout << "y: " << y;
```

```
}

class B:public A          ایجاد کلاسی به نام B که ویژه گی های کلاس A را به صورت public به ارث می برد
private:
        float z;          متغییرهای خصوصی کلاس مشتق شده
public:
        void inpute();    متدهای کلاس مبنا
};
int main()                تابع main()
{
        B ob1;            ایجاد شی ای به نام ob1 از کلاس مشتق شده یا همان B
        ob1.insert();     شی ob1 می تواند به تمام متدهای public کلاس مبنا دسترسی داشته باشد
        ob1.print();      ... ... ...
        getch();
        return 0;
}
```

توضیح کلاس : در ابتدا کلاسی به نام A پیاده سازی می کنیم که این کلاس دارای دو متغییر به نام های x و y می باشد و هم چنین دارای دو متد، به نام insert() برای دریافت داده ها و متد print() برای نمایش داده ها می باشد.
کلاس دیگری به نام B پیاده سازی می کنیم که ویژه گی های کلاس A را به صورت public به ارث می برد.
بنابراین کلاس A یک کلاس مبنا و کلاس B یک کلاس مشتق شده می باشد. در مطالب قبل هم گفتیم که کلاس مشتق شده به هیچ وجه نمی تواند به اعضای خصوصی یا private کلاس مبنا دسترسی داشته باشد. وقتی که در این جا نحوه ارث بری را به صورت public معرفی کرده ایم بدین معنی می باشد که تمام متدهای قسمت public کلاس مبنا به متدهای public کلاس مشتق شده اضافه می شوند. بنابراین کلاس مشتق شده می تواند به تمام متغییرها و متدهای قسمت public کلاس مبنا دسترسی داشته باشد.

1-4 نحوه دسترسی protected

همان طور که می دانیم نحوه دسترسی private به صورت خصوصی و نحوه دسترسی public به صورت عمومی می باشد. بنابراین در تابع main() به متغییرهای قسمت private نمی توان دسترسی پیدا کرد، اما به متغییرها و متدهای قسمت public می توان دسترسی داشت.
علاوه بر این دو نوع، یک نحوه دسترسی دیگر به نام protected وجود دارد که بیش تر در کلاس ها مورد استفاده قرار می گیرد. این نحوه دسترسی، نه به طور کامل خصوصی و نه به طور کامل عمومی می باشد، یعنی تقریبا هم ویژه گی های private و public را تا حدودی دارا می باشد.

نکته 1-4 :عضوی که به صورت protected معرفی می شود، نمی تواند در داخل main() مورد استفاده قرار بگیرد اما این عضو میتواند در داخل کلاس مشتق شده مورد استفاده قرار بگیرد. مثال زیر خلاصه ای از این سه نحوه دسترسی را بیان می کند.

مثال 2-4 : با توجه به کلاس های زیر، هرقسمتی از برنامه که دارای اشکال می باشد را مشخص و نحوه صحیح نوشتن آن را بنویسید.

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
class A                                  نوشتن کلاسی به نام A
{
private:
        float x, y, z;                   متغییرهای خصوصی کلاس پایه
protected:
        float n;                         متغییرهای protected کلاس پایه
public:
        void inpute();                   متدهای کلاس پایه
        void calculate();                ... ... ...
        void print1();                   ... ... ...
};
void A::inpute()                         پیاده سازی متد inpute
{
        cout << "enter x,y: ";
        cin >> x >> y;
}
void A::calculate()                      پیاده سازی متد calculate
{
        z = x*x + y*y;
}
void A::print1()                         پیاده سازی متد print1
{
        cout << x << " " << n << " " << a << " " << w << endl;      // 1 خطا
}

class B:public A        ایجاد کلاسی به نام B که به صورت public از کلاس مبنا ارث می برد
{
private:
        float a, b, c;                   متغییرهای خصوصی کلاس مشتق شده
protected:
        float w;                         متغییرهای protected کلاس مشتق شده
public:
        void set(float a1, float b1, float c1);   متدهای کلاس مشتق شده
        void print2();                   ... ... ...
};
void B::set(float a1, float b1, float c1)   پیاده سازی متد set
{
        if (a1 || b1 || c1 < 0)
```

```
        {
                cout << "Erorr!";
                exit(1);
        }
        a = a1;
        b = b1;
        c = c1;
}
void B::print2()                                        پیاده سازی متد print2
{
        cout << x << " " << n << " " << a << " " << w << endl;          // خطا 2
}
int main()                                              تابع main()
{
        A ob1;                          ایجاد شی ای به نام ob1 از نوع کلاس مبنا
        B ob2;                          ایجاد شی ای به نام ob2 از نوع کلاس مشتق شده
        ob1.x = 3;                                                      // خطا 3
        ob1.n = 15;                                                     // خطا 4
        ob1.inpute();
        ob1.a = 12;                                                     // خطا 5
        ob1.w = 24;                                                     // خطا 6
        ob1.set(15, 8, 12);                                             // خطا 7

        ob2.x = 15;                                                     // خطا 8
        ob2.n = 18;                                                     // خطا 9
        ob2.inpute();
        ob2.a = 19;                                                     // خطا 10
        ob2.w = 38;                                                     // خطا 11
        ob2.set(4, 8, 12);
        getch();
        return 0;
}
```

توضیح کلاس : در برنامه بالا دو کلاس به وجود آمده است. اولی کلاس مبنا که A نام دارد و دومی کلاس مشتق شده که B نام دارد. کلاس مشتق شده، ویژه گی های خود را از کلاس مبنا به صورت public به ارث می برد، یعنی این که همه ی متدهای موجود در کلاس مبنا به قسمت public کلاس مشتق شده اضافه می شوند و کلاس مشتق شده می تواند از این متدها استفاده کند. در این برنامه همان طور هم که مشاهده می کنید، 11 خطا وجود دارد که به بررسی هرکدام از آن ها می پردازیم و نحوه صحیح آن را می نویسیم.

خطا 1 : در این خط متغییر که مربوط به متد print1 می باشد و وظیفه چاپ اعداد را برعهده دارد. متد print1 که عضو کلاس مبنا یا A می باشد، بنابراین طبق مطالبی که در قبل گفتیم، متدی که عضو کلاس باشد می تواند داد ه های قسمت private و protected را مورد استفاده خود قرار دهد. بنابراین نوشتن متغییرهای x و n در این متد هیچ

مشکلی را به وجود نمی آورد. اما خطای برنامه مربوط به متغییر های a و w می باشد. متغییر a عضو خصوصی و متغییر w عضو protected کلاس B می باشد. همان طور هم که در قبل گفتیم، عضوهای خصوصی یک کلاس، فقط در داخل همان کلاس قابل دسترسی می باشند و خارج از کلاس حق استفاده از آن ها را نداریم، بنابراین استفاده از این متغییر در داخل کلاس A اشتباه می باشد و فقط حق استفاده از این متغییر را در داخل کلاس B را داریم. متغییر w هم که عضو protected کلاس B می باشد و عضوهای protected فقط در داخل کلاس مشتق شده قابل دسترسی می باشند نه در داخل کلاس مبنا. بنابراین در داخل متد print1 فقط حق استفاده از متغییرهای x و n را داریم.

خطا 2 : متد print2 عضو کلاس مشتق شده یا B می باشد بنابراین نمی توانیم اعضای خصوصی کلاس دیگری را در این متد مورد استفاده قرار دهیم، بنابراین متغییر x عضو خصوصی کلاس مبنا می باشد و نباید آن را در کلاس دیگری مورد استفاده قرار داد. متغییر n که عضو protected کلاس مبنا می باشد می تواند در کلاس مشتق شده مورد استفاده قرار بگیرد بنابراین استفاده از متغییر n در این قسمت هیچ مشکلی را پیش نمی آورد. متغییر a که عضو خصوصی خود کلاس و متغییر w هم که عضو protected کلاس می باشد بنابراین حق استفاده از آن ها را در این متد نیز داریم. پس فقط حق استفاده از متغییر x را نداریم، چون عضو خصوصی کلاس مبنا می باشد.

خطا 3 : شی ob1 را از کلاس A به وجود آمده است. این شی متغییر x را مقدار دهی کرده است. همان طور که می دانید، متغییرهای قسمت private به هیچ وجه حق استفاده در تابع ()main را ندارند. بنابراین این دستور اشتباه می باشد.

خطا 4 : شی ob1، متغییر n را که عضو protected کلاس مبنا می باشد مقدار دهی کرده است. این دستور نیز اشتباه می باشد، چون عضوهای protected به هیچ وجه در داخل تابع ()main نمی توانند بیایند.

خطا 5 : متغییر a هم که عضو خصوصی کلاس مشتق شده می باشد.همان طور هم که می دانیم،عضوهای خصوصی نمی توانند درتابع ()main بیایند بنابراین این متغییر هم نمی تواند در تابع ()main حضور داشته باشد.

خطا 6 : متغییر w عضو protected کلاس مشتق شده می باشد، بنابراین این دستور نیز اشتباه می باشد.

خطا 7 : متد ()set عضو متدهای public کلاس مشتق شده می باشد، بنابراین فقط با شی ای که توسط کلاس مشتق شده یا B به وجود می آید می توان از این متد در تابع ()main استفاده کرد. اما در این جا با استفاده از شی ای که توسط کلاس مبنا یا A ایجاد شده است، مورد استفاده قرار می یگرد، بنابراین این دستور نیز اشتباه می باشد.

خطا 8 : شی ob2 از کلاس B به وجود آمده است. این شی متغییر x را مقدار دهی کرده است. و چون متغییر x نیز یک عضو خصوصی می باشد، بنابراین نمی توانیم از این متغییر درداخل تابع()main استفاده کنیم. این دستور نیز اشتباه می باشد و برنامه را دچار اشکال خواهد کرد.

خطا 9 : متغییر n عضو protected کلاس مبنا می باشد و در قبل هم گفتیم که عضوهای protected به هیچ وجه در تابع ()main نمی آیند، بنابراین این دستور نیز اشتباه می باشد.

خطا 10 : چون متغییر a یک عضو خصوصی می باشد، بنابراین نمی توان آن را در تابع ()main آورد.

خطا 11 : متغییر w نیز یک عضو protected می باشد، بنابراین چون عضوهای protected نمی توانند در تابع ()main حضور داشته باشند، بنابراین این دستور نیز اشتباه می باشد.

2-4 سازنده ها و مخرب ها در ارث بری

در فصل های قبل با سازنده ها و مخرب ها به طور کامل آشنا شدیم و نحوه پیاده سازی و استفاده از آن ها را نیز یاد گرفتیم. سازنده ها و مخرب ها نیز در ارث بری می توانند مورد استفاده قرار بگیرند.
زمانی که شی ای از نوع کلاس مشتق شده ایجاد می شود، علاوه بر متد سازنده کلاس مشتق شده، متد سازنده کلاس مبنا نیز صدا زده می شود و زمانی که شی ای از نوع کلاس مشتق شده از بین می رود، علاوه بر فراخوانی متد مخرب کلاس مشتق شده، متد مخرب کلاس مبنا نیز صدا زده می شود.
ترتیب فراخوانی ها نیز بدین صورت می باشد که برای فراخوانی متد سازنده: ابتدا متد سازنده کلاس مبنا و سپس متد سازنده کلاس مشتق شده فرخوانی می شود اما برای فراخوانی متد مخرب: برعکس می باشد، یعنی این که ابتدا متد مخرب کلاس مشتق شده و سپس متد مخرب کلاس مبنا فراخوانی می شود.
اگر بخواهیم در کلاس مشتق شده از سازنده های پارامتر دار استفاده کنیم، با استفاده از تکنیک " رد کردن پارامتر " می توان این پارامترها را به کلاس مبنا ارسال کرد. مانند دستور زیر:

```
#include<iostream.h>
#include<conio.h>
class A
{
private:
        float a;
public:
        A(float x1);                    متد سازنده کلاس A با پارامتر x1
        ~A();                           متد مخرب کلاس A
};
A::A(float x1)                          پیاده سازی متد سازنده کلاس A
{
        a = x1;
        cout << "call A constructor" << endl;
}
A::~A()                                 پیاده سازی متد مخرب کلاس A
{
        cout << "call A destructor" << endl;
}

class B :public A       کلاس B که به صورت public از کلاس A ارث می برد
{
private:
        float b;
public:
        B(float x2);                    متد سازنده کلاس B با پارامتر x2
        ~B();                           متد مخرب کلاس B
};
B::B(float x2) :A(x2)                   پیاده سازی متد سازنده کلاس B
```

```
{
        b = x2;
        cout << "call B constructor" << endl;
}
B::~B()                                  پیاده سازی متد مخرب کلاس B
{
        cout << "call B destructor" << endl;
}
int main()                               تابع main()
{
        B ob2(12);          ایجاد سازنده ای به نام ob2 از نوع کلاس B
        getch();
        return 0;
}
```

توضیح کلاس : در برنامه بالا یک کلاس مبنا به نام A و یک کلاس مشتق شده به نام B وجود دارد. هر کدام از این دو کلاس دارای یک متد سازنده و یک متد مخرب می باشند. همان طور هم که در قبل گفتیم، اگر سازنده پارامتر دار در برنامه داشتیم، باید به صورت ترفند " رد کردن پارامتر "، پارامتر های کلاس مشتق شده را برای کلاس مبنا بفرستیم. برای استفاده از این ترفند، در قسمت پیاده سازی متدها یا همان " قسمت دوم " به صورت زیر عمل می کنیم:

```
B::B(float x2) :A(x2)
```

برای نوشتن متد سازنده، دقیقا به صورت قبل عمل می کنیم فقط با این تفاوت که درادامه، کاراکتر (:) را می نویسیم و سپس نام کلاس مبنا و و پارامتر داخل آن را باید همان پارامتری بنویسیم که قرار است برای کلاس مبنا بفرستیم. پس دقت کنید که در ادامه دستور A(x2): به سازنده اضافه شده می باشد. پس حتما باید در سازنده کلاس A، همان پارامتری را قرار بدهیم که قرار است به کلاس مبنا فرستاده شود، یعنی پارامتر سازنده کلاس B را باید در داخل سازنده کلاس A بنویسیم.

3-4 توابع مجازی

گاهی اوقات شرایطی پیش می آید که ما می خواهیم، متدی را که در کلاس مبنا تعریف شده است، تعریف جدیدی را در کلاس مشتق شده نیز از آن داشته باشیم. دقت کنید که متد کلاس مبنا به هر تعداد پارامتر دارد، باید در داخل متد مشتق شده نیز همان پارامترها با همان نام را بنویسیم. در چنین مواردی از متد مجازی استفاده می کنیم.
برای این که یک متد را به صورت، متد مجازی معرفی کنیم، کافی است که در کلاس مبنا قبل از معرفی متد، کلمه ی کلیدی virtual را قرار بدهیم. که در این صورت، کلاس مشتق شده می تواند متد مجازی را به صورتی دیگر پیاده سازی کند. اما اگر یک متد را در کلاس مبنا به صورت متد مجازی معرفی کردیم، اما در داخل کلاس مشتق شده، دستورات آن را پیاده سازی نکردیم، بنابراین این متد از همان تعریفی که در کلاس مبنا وجود دارد، استفاده می کند. به صورت زیر:

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

```
class A                                  کلاس مبنا یا همان A
{
private:
        int x;
public:
        virtual void show();             یک متد مجازی
};
void A::show()                           پیاده سازی متد مجازی
{
        cout << "class A";
}

class B :public A        کلاس B که به صورت public از کلاس A ارث می برد
{
private:
        int y;
public:
        void show();
};
void B::show()           تعریفی جدیدی ازمتد مجازی در داخل کلاس مشتق شده
{
        cout << "class B";
}
int main()
{
        B ob2;                           ایجاد یک شی به نام ob2 از نوع کلاس B
        ob2.show();
        getch();
        return 0;
}
```

توضیح کلاس : در برنامه بالا یک کلاس مبنا به نام A و یک کلاس مشتق شده به نام B وجود دارد. در داخل کلاس مبنا یک متد به نام show() وجود دارد که وظیه چاپ پیغام " class A " را بر عهده دارد. قبل از این متد از کلمه ی کلیدی virtual استفاده شده است، این بدین معنی است که متد show() یک متد مجازی می باشد. بنابراین کلاس مشتق شده می تواند تعریف جدیدی از این متد را در داخل خود داشته باشد. در داخل متد مشتق شده برای متد show()، پیغام " class B " نوشته شده است. بنابراین در داخل main() که شی ob2 به متد show() دسترسی دارد، در خروجی پیغام " class B " چاپ می شود.

4-4 متد مجازی محض

همان طور که در قبل گفتیم، متد مجازی کلاس مشتق شده می تواند تعریف جدیدی از متد مجازی کلاس مبنا را ارائه ندهد و از همان تعریف پیش فرض کلاس مبنا استفاده کند. اما اگر کلاس مبنا به کلاس مشتق شده دستور بدهد که حتما باید تعریف جدیدی از این متد را داشته باشد و از تعریف پیش فرض کلاس مبنا استفاده نکند، از متد مجازی محض بدین صورت استفاده می کنیم: ابتدا باید هیچ تعریفی از متد مجازی را در داخل کلاس مبنا نداشته باشیم و آن را مساوی با صفر قرار دهیم. به صورت زیر:


```
#include<iostream.h>
#include<conio.h>
class A                                         کلاس مبنا یا همان A
{
private:
        int x;
public:
        void insert();
        virtual void show() = 0;                یک متد مجازی محض
};

class B :public A               کلاس B که به صورت public از کلاس A ارث می برد
{
private:
        int y;
public:
        void show();
};
void B::show()                                  پیاده سازی متد مجازی محض
{
        cout << "class B";
}
int main()
{
        B ob2;                          ایجاد یک شی به نام ob2 از نوع کلاس B
        ob2.show();
        getch();
        return 0;
}
```

توضیح کلاس : در برنامه بالا یک کلاس مبنا به نام A و یک کلاس مشتق شده به نام B وجود دارد. در داخل کلاس مبنا یک متد به نام show() وجود دارد که مساوی با صفر قرار داده شده است. بنابراین این متد، یک متد مجازی محض می باشد و کلاس مشتق شده حتما باید یک تعریف جدید از این متد را داشته باشد. بنابراین اگر یک متد مجازی در داخل کلاس مبنا وجود داشت و مساوی با صفر قرا داده شد، نتیجه می گیریم که آن متد، یک متد مجازی محض می باشد. بنابراین حتما باید یک تعریف جدید از این متد را اشته باشیم.

به کلاسی که حداقل یک متد مجازی محض داشته باشد، کلاس تجریدی یا انتزاعی (abstract) گفته می شود. کلاس A مثال قبل، چون یک متد مجازی محض دارد، بنابراین کلاس A یک کلاس تجریدی یا انتزاعی می باشد.

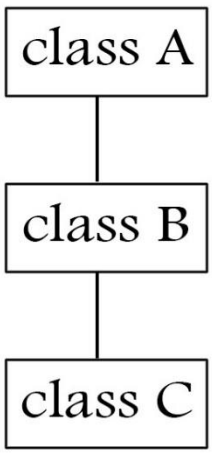
5-4 تفاوت های متد مجازی با سربارگزاری توابع

1- در سربارگزاری توابع حتما باید تعداد پارامترها و نوع آن ها متفاوت باشد در حالی که در متدهای مجازی تعداد پارامترها و نوع پارامترها یکسان می باشد.
2- در سربارگزاری توابع، می توان تابعی که عضو کلاس نیست را سربارگزاری کرد در حالی که برای سربارگزاری متدهای مجازی، تابع باید حتما عضو کلاس باشد.
3- در سربارگزاری توابع، می توان سازنده کلاس را به فرم های مختلفی سربارگزاری کرد. در حالی که استفاده از متد مجازی برای سربارگزاری سازنده ها بی معنی می باشد.

6-4 ارث بری چندگانه

در مثال های قبل ما دو کلاس به وجود آوردیم که اولی را یک کلاس مبنا و دومی را یک کلاس مشتق شده معرفی کردیم و کلاس دومی از کلاس اولی ویژه گی های خود را به ارث می برد. ارث بری چندگانه بدین صورت می باشد که کلاس مشتق شده ویژه گی های خود را از چندین کلاس به ارث ببرد و دارای چندین کلاس مبنا باشد.
ارث بری چندگانه به دو صورت قابل پیاده سازی می باشد.

1- ارث بری چندگانه غیر مستقیم : این نوع ارث بری بدین صورت می باشد که مثلا کلاس B از کلاس A ارث می برد و کلاس C نیز از کلاس B ارث می برد. بنابراین می توان گفت که کلاس C هم ویژه گی های کلاس B را به صورت مستقیم و هم ویژه گی های کلاس A را به صورت غیر مستقیم به ارث می برد. شکل این کلاس به صورت زیر می باشد و کدهای آن را نیز در زیر پیاده سازی کرده ایم:

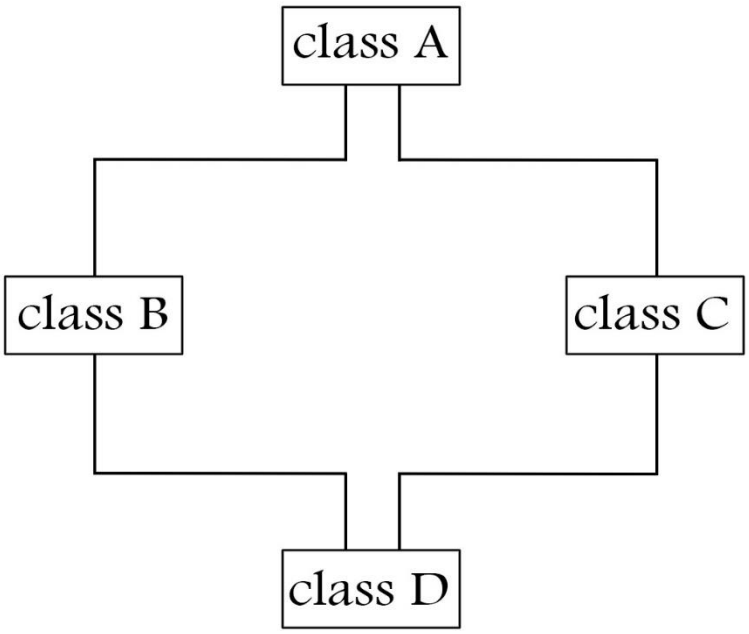


```
class A
{
… … …;
}

class B :public A
{
… … …;
}

class C :public B
{
… … …;
}
```

2- ارث بری چندگانه مستقیم : این نوع ارث بری بدین صورت می باشد که مثلا کلاسی مبنا به نام کلاس A وجود دارد و هم زمان دو کلاس به نام های B و C یا بیش از دو کلاس از آن ارث می برند و در آخر سر هم یک کلاس به نام کلاس D از کلاس B و C ارث می برد. بنابراین خصوصیات کلاس A هم در کلاس B و هم در کلاس C وجود دارد، در نتیجه کلاس D که هم از کلاس B و هم از کلاس C ارث می برد، به ناچار خصوصیات کلاس A را دو بار به ارث می برد. بنابراین برنامه ما دچار اشکال خواهد شد و برای رفع این مشکل ما از کلاس مبنای مجازی به صورت زیر استفاده می کنیم:



```
class A
{
… … … ;
}

class B : virtual public A
{
… … … ;
}

class C : virtual public A
{
… … … ;
}

class D : virtual public B, public C
{
… … … ;
}
```

نمونه سوالات امتحانی

1- لزوم استفاده از کلاس مبنای مجازی چیست؟ با استفاده از کد توضیح دهید.

اگر کلاسی مبنا به نام کلاس A وجود داشته باشد و هم زمان دو کلاس به نام های B و C یا بیش از دو کلاس از آن ارث ببرند و در آخر سر هم یک کلاس به نام کلاس D از کلاس B و C ببرد، در نتیجه خصوصیات کلاس A هم در کلاس B و هم در کلاس C وجود دارد، پس کلاس D که هم از کلاس B و هم از کلاس C ارث می برد، به ناچار خصوصیات کلاس A را دو بار به ارث می برد. بنابراین برنامه ما دچار اشکال خواهد شد و برای رفع این مشکل ما از کلاس مبنای مجازی به صورت زیر استفاده می کنیم:

```
class A
{
… … … ;
}

class B : virtual public A
{
… … … ;
}

class C : virtual public A
{
```

```
… … … ;
}

class D : virtual public B, public C
{
… … … ;
}
```

2- کلاس زیر برای کار با بردارها در فضای سه بعدی در نظر گرفته شده است. با توجه به ()main داده شده، متدهای لازم را پیاده سازی کنید.

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
class vector
{
private:
        float x, y, z;
public:
        … … …;
        … … …;
};
int main()
{
        vector a(2, 5, 9);              سازنده سه پارامتری
        vector b, c;                    سازنده بدون پارامتر
        cin >> b;                       سربارگزاری عملگر cin
        c = a++;                        سربارگزاری عملگر افزایشی
        cout << c;                      سربارگزاری عملگر cout
        if (a < b)                      سربارگزاری عملگر مقایسه ای
        {
                c = b - a;              سربارگزاری عملگر تفریق
                cout << c;              سربارگزاری عملگر cout
        }
        getch();
        return 0;
}
```

کلاس بالا به صورت زیر پیاده سازی می شود:

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

```
#include<math.h>
class vector
{
private:
        float x, y, z;
public:
        vector(float x1, float y1, float z1);                          متد سازنده سه پارامتری
        vector();                                                      متد سازنده بدون پارامتر
        friend istream & operator>>(istream & I, vector & r);          سربارگزاری عملگر cin
        friend ostream & operator<<(ostream & O, vector  rp);          سربارگزاری عملگر cout
        vector operator++(int n);                                      سربارگزاری عملگر افزایشی
        int operator<(vector rop;                                      سربارگزاری عملگر مقایسه ای
        vector operator-(vector rp2);                                  سربارگزاری عملگر تفریق
        int calculate();                                               تابع محاسبه
};
vector::vector()                                                       پیاده سازی متد سازنده بدون پارامتر
{
        x = 0;
        y = 0;
        z = 0;
}
vector::vector(float x1, float y1, float z1)                           پیاده سازی متد سازنده سه پارامتری
{
        x = x1;
        y = y1;
        z = z1;
}
istream & operator>>(istream & I, vector & rop)                        پیاده سازی عملگر cin
{
        I >> rop.x;
        I >> rop.z;
        I >> rop.y;
        return I;
}
ostream & operator<<(ostream & O, vector  rp)                          پیاده سازی عملگر cout
{
        O << rp.x;
        O << rp.z;
        O << rp.y;
        return O;
}
```

پیاده سازی عملگر افزایشی

```
vector vector::operator++(int n)
{
	vector r;
	r.x = x;
	r.y = y;
	r.z = z;
	x = x + 1;
	y = y + 1;
	z = z + 1;
	return r;
}
```

پیاده سازی متد محاسبه

```
int vector::calculate()
{
	float s;
	s = (x*x) + (y*y) + (z*z);
	return sqrt(s);
}
```

پیاده سازی عملگر مقایسه ای

```
int vector::operator<(vector rop)
{
	if (calculate() < rop.calculate())
		return 1;
	return 0;
}
```

پیاده سازی عملگر تفریق

```
vector vector::operator-(vector rp2)
{
	vector t;
	t.x = rp2.x + x;
	t.y = rp2.y + y;
	t.z = rp2.z + z;
	return t;
}
```

تابع main()

```
int main()
{
	vector a(2, 5, 9);
	vector b, c;
	cin >> b;
	c = a++;
	cout << c;
	if (a < b)
	{
		c = b - a;
```

```
            cout << c;
        }
        cin.get();
        cin.get();
        return 0;
}
```

توضیح کلاس : در این جا ما باید به تابع ()main نگاه کنیم و هر شی ای را که مورد عملیات محاسباتی قرار گرفته است را سربارگزاری کنیم.همان طور هم که در بالا مشاهده می کنید، عملگرهایی که باید سربارگزاری بکنیم را مشخص کرده ایم. این عملگرها عبارت اند از: 1- عملگر cin 2- عملگر cout 3- عملگر مقایسه ای 3- عملگر تفریق 4- عملگر افزایشی.

علاوه بر این 4 عملگر که باید به ترتیب آن هاذ را سربارگزاری کنیم، در قسمت ()main سه شی به نام های a و b و c وجود دارد که شی a دارای سه پارامتر می باشد، بنابراین a سازنده سه پارامتری می باشد. شی های b و c نیز هیچ پارامتری ندارند، بنابراین این ها نیز سازنده های بدون پارامتر می باشند.
دقت کنید که در قسمت private اشاره گر وجود ندارد، بنابراین ما باید تمامی این عملگر ها را با روشی که در مطالب قبل در نبود اشاره گر در برنامه داشتیم را پیاده سازی کنیم. تک تک این عملگر ها و سارنده ها را در مطالب قبلی سربارگزاری کرده ایم پس نیازی به توضیح آن ها نمی باشد و با توجه به تعریفی که از هر کدام در قبل داشته ایم، آن ها را پیاده سازی می کنیم.

3- کلاس زیر برای کار با چند جمله ای ها در نظر گرفته شده است. با توجه به ()main داده شده، متدهای لازم را پیاده سازی کنید. چنان چه در زمان ایجاد شی، درجه ی چند جمله ای مشخص نشده باشد به طور پیش فرض مقدار آن را صفر در نظر بگیرد. ضرایب چند جمله ای نیز در آغاز صفر می باشد.

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
class polynomial
{
private:
        double *a;
        int d;
public:
        ... ... ...;
        ... ... ...;
}
int main()
{
        polynomial ob1(7), ob2(4);
        polynomial ob3;
        int k = 1;
```

```
        cin >> ob1 >> ob2;
        ob3 = ob2 + ob1;
        cout << ob3;
        while (k > 0)
        {
                polynomial ob4 = ob1;
                cin >> k;
        }
        cout << ob1;
        getch();
        return 0;
}
```

کلاس بالا به صورت زیر پیاده سازی می شود:

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
class polynomial
{
private:
        double *a;
        int d;
public:
        polynomial();                                                   متد سازنده بدون پارامتر
        polynomial(int n);                                              متد سازنده تک پارامتر
        polynomial(const polynomial & ob);                              متد سازنده کپی
        polynomial operator+(polynomial rp);                            سربارگزاری عملگر جمع
        polynomial & operator=(polynomial & p);                         سربارگزاری عملگر نسبت دهی
        friend istream & operator>>(istream & I, polynomial & rop);     سربارگزاری عملگر cin
        friend ostream & operator<<(ostream & O, polynomial  rop2);     سربارگزاری عملگر cout
};
polynomial::polynomial()                                                پیاده سازی متد سازنده بدون پارامتر
{
        d = 0;
}
Polynomial & polynomial::operator=(polynomial & p)                      پیاده سازی عملگر نسبت دهی
{
        if (d != p.d)
        {
                delete[]a;
```

```
            a = new double[p.d];
        }
        if (!a)
        {
            cout << "Erorr!";
            exit(1);
        }
        d = p.d;
        for (int i = 0; i < p.d; i++)
        {
            a[i] = p.a[i];
        }
        return *this;
}
polynomial::polynomial(int n)
{
        a = new double[n + 1];
        if (!a)
        {
            cout << "Error!";
            exit(1);
        }
        d = n;
        for (int i = 0; i <= n; i++)
        {
            a[i] = 0;
        }
}
polynomial::polynomial(const polynomial & ob)
{
        a = new double[ob.d + 1];
        if (!a)
        {
            cout << "Error!";
            exit(1);
        }
        d = ob.d;
        for (int i = 0; i <= d; i++)
        {
            a[i] = ob.a[i];
        }
```

پیاده سازی متد سازنده تک پارامتری

پیاده سازی متد سازنده کپی

علی نظری زاده

```
}
polynomial polynomial::operator+(polynomial rp)
{
        polynomial r(rp.d);
        if (d != rp.d)
        {
                cout << "Erorr!";
                exit(1);
        }
        for (int i = 0; i <= d; i++)
        {
                rp.a[i] = a[i] + rp.a[i];
        }
        return r;
}
ostream & operator<< (ostream & O, polynomial  rop2)
{
        for (int i = 0; i <= rop2.d; i++)
        {
                O << rop2.a[i];
        }
        return O;
}
istream & operator>> (istream & I, polynomial & rop)
{
        for (int i = 0; i <= rop.d; i++)
        {
                I >> rop.a[i];
        }
        return I;
}
int main()
{
        polynomial ob1(7), ob2(4);
        polynomial ob3;
        int k = 1;
        cin >> ob1 >> ob2;
        ob3 = ob2 + ob1;
        cout << ob3;
        while (k > 0)
        {
```

پیاده سازی عملگر جمع

پیاده سازی عملگر cout

پیاده سازی عملگر cin

```
            polynomial ob4 = ob1;
            cin >> k;
        }
        cout << ob1;
        cin.get();
        cin.get();
        return 0;
}
```


توضیح کلاس : همان طور که مشاهده می کنید، در این قسمت نیز تابع ()main را به ما داده اند و از ما خواسته شده که متدها را پیاده سازی کنیم. به این نکته دقت کنید که ما اصلا نباید به اسم کلاس توجه کنیم، فقط و فقط باید ببینیم که در قسمت ()main چه متدها و عملگرهایی مورد استفاده قرار گرفته شده است و با توجه به این که در قسمت ()main اشاره گر وجود دارد یا نه باید متدها را تک به تک پیاده سازی کنیم. در قسمت ()main شی ob1 و شی ob2 دارای یک پارامتر می باشند، بنابراین هر دو سازنده تک پارامتری می باشند و با ید کدهای مربوط به هرکدام را که در فصل سوم مورد یررسی قرار دادیم را پیاده سازی کنیم. در قسمت private اشاره گر وجود دارد، بنابراین باید همه ی عملگرها را با توجه به این که در برنامه اشاره گر وجود دارد را پیاده سازی کنیم. به این نکته خوب دقت کنید که ما در قبل هم گفتیم که وقتی عملیات محاسباتی بر روی اشیاء صورت بگیرد، باید دقیقا همان عملگر را سربارگزاری کنیم، در تابع ()main یک متغییر به نام k از نوع عدد صحیح وجود دارد که از بر روی این عملگر نیر عملیات محاسباتی صورت گرفته شده است، اما دقت کنید که k فقط یک متغییر می باشد نه شی، پس هر محاسباتی که بر روی آن صورت بگیرد نباید آن را سربارگزاری کنیم. عملیات سربارگزاری فقط برای اشیاء می باشد نه متغییرها.

فهرست مطالب

فصل اول: مقدمه ای بر برنامه نویسی 2
1-1 حلقه های تکرار و دستورات شرطی 3
2-1 آرایه ها 8
3-1 تابع 9
4-1 اشاره گرها 10
فصل دوم: کلاسها و اشیاء 11
1-2 سازنده ها و مخرب ها 22
2-2 اشاره گرها 26
3-2 متد سازنده کپی 28
فصل سوم: سربارگزاری عملگرها 48
1-3 سربارگزاری عملگرهای جمع و تفریق و ضرب و تقسیم 53
2-3 سربارگزاری عملگر قرینه 55
3-3 سربارگزاری عملگرهای افزایشی و کاهشی 57
4-3 سربارگزاری عملگرهای مقایسه ای 61
5-3 سربارگزاری عملگر نسبت دهی 65
6-3 سربار گزاری عملگرهای ترکیبی 65
7-3 تابع دوست (friend) 68
8-3 سربارگزاری عملگر های cin و cout 70
فصل چهارم: ارث بری 73
1-4 نحوه دسترسی protected 75
2-4 سازنده ها و مخرب ها در ارث بری 79
3-4 توابع مجازی 80
4-4 متد مجازی محض 81
5-4 تفاوت های متد مجازی با سربارگزاری توابع 83
6-4 ارث بری چندگانه 83
نمونه سوالات امتحانی 85

علی نظری زاده