جزوه درس

برنامه‌سازی پیشرفته

مباحث درس:

- ☐ توابع
- ☑ آرایه دوبعدی
- ☐ کاراکترها
- ☐ رشته‌ها
- ☐ ساختارها
- ☐ اشاره‌گرها
- ☐ لیست پیوندی
- ☐ توابع بازگشتی
- ☐ برنامه‌نویسی ویژوال
- ☐ برنامه‌سازی شیءگرا

ارزشیابی:

- حضور و فعالیت در کلاس، تمرین‌ها و پروژه‌ها، کلاس حل تمرین
- امتحان میان‌ترم
- امتحان عملی
- امتحان پایان‌ترم

تاریخ امتحان میان‌ترم: . . . . . (در جلسات اول کلاس تعیین می‌شود)

توابع

منظور از تابع زیربرنامه‌هایی است که حل بخشی از مسأله به عهده آنها است. توابع دو نوع هستند:

- توابع کتابخانه‌ای (ازپیش تعریف شده)
- توابعی که توسط خود برنامه‌نویس تعریف می‌شوند

نوشتن تابع توسط برنامه نویس

سه مزیت استفاده از زیربرنامه (تابع) توسط برنامه‌نویسان:

۱- قابلیت استفاده مجدد از کد: چندین بار استفاده از کد با فراخوانی مکرر تابع

۲- کاهش پیچیدگی تولید برنامه و افزایش قابلیت فهم برنامه

۳- افزایش قابلیت کار تیمی در تولید نرم‌افزار

مثال ۱- می‌خواهیم برنامه‌ای بنویسیم که با دریافت مقدار دو عدد n و r، حاصل فرمول زیر را محاسبه کند:

$$C_{n,r} = \frac{n!}{r!\,(n-r)!}$$

```
using namespace std;
#include “iostream”
#include “conio.h”
long int fact (int a)
{
        long int f = 1;
        for (int i = 1 ; i <= a ; i++)
                f  *= i;
        return f;
}
void main( )
{
        int n , r;
        long int result , x1 , x2 , x3;
```

```
        cin >> n >> r;
        x1 = fact ( n );
        x2 = fact ( r );
        x3 = fact ( n – r );
        result = x1 / ( x2 * x3 );
        cout << result;
}
```

چند نکته:

الف– یک برنامه ممکن است چند تابع داشته باشد که نام یکی از آنها حتماً main است. اجرا همیشه از main شروع می‌شود.

ب– شکل کلی تابع به صورت زیر است:

```
نوع نتیجه تابع      نام‌تابع    (نوع و نام پارامترها)
{
        .
        .
        .
        return ...;      (معمولاً)
}
```

پ– ممکن است تابع پارامتری نداشته باشد اما بازهم، هم در تعریف تابع و هم در فراخوانی آن یک جفت پرانتز خالی می‌آید.

ت– پارامترهای حقیقی یعنی پارامترهای هنگام فراخوانی باید از نظر تعداد و نوع و ترتیب با پارامترهای ظاهری (فرمال) یعنی پارامترهای تعریف تابع سازگار باشند.

ث– به‌طور معمول، تابع فراخوانی‌شونده باید قبل از فراخواننده تعریف شود. اما می‌توانیم ابتدا تعریف خلاصه‌ای از تابع فراخوانی‌شونده که اصطلاحاً prototype نامیده می‌شود ذکر کنیم و بعد تابع فراخواننده و بعد تابع فراخوانی شونده را بیاوریم:

```
using namespace std;
#include “iostream”
#include “conio.h”
long int fact (int );        // fact تابع prototype
void main( )
{
        .
        .
        x1 = fact(n);
        .
        .
```

```
}
long int fact ( int a )
{
        .
        .
}
```

مثال۲- در این برنامه، مقدار یک زاویه برحسب درجه دریافت می‌شود و پس از تبدیل به رادیان با محاسبه‌ی بیست جمله از سری زیر سینوس آن تعیین می‌شود:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots$$

```
using namespace std;
#include “iostream”
#include “conio.h”
double fact ( int a )
{
        double f = 1;
        for (int i = 1; i<= a; i++)
                f  *= i;
        return f;
}
float power ( float a,  int b )
{
        float p = 1;
        for ( int k = 1; k<= b; k++)
                p  *=  a;
        return p;
}
float sin ( float x )
{
        float sum = 0;
        int z = 1;
        int m = 1;
        for (int k = 1; k <= 20; k++)
        {
```

```
            sum += z * power(x , m) / fact(m);
            m += 2;
            z = -z;
        }
        return sum;
}
void main ( )
{
        float x;
        cin  >>  x;
        x = x * 3.14 / 180;
        cout << sin (x);
}
```

توابع void

توابعی که مقدار خاصی را به عنوان نتیجه برنمی‌گردانند، بلکه فقط عملیاتی را انجام می‌دهند و معمولاً نتایج محاسبات خود را به جای برگرداندن، در داخل خود تابع چاپ می‌کنند، توابع void نامیده می‌شوند.

مثال۳- برنامه زیر دو عدد را دریافت می‌کند و همه مضرب های عدد ۷ را که بین این دو عدد واقع هستند، به ترتیب صعودی چاپ می‌کند. هر یک از این دو عدد ممکن است بزرگ‌تر باشد.

```
. . .
void multiples ( int min , int max )
{
        for ( int n = min+1; n <= max-1; n++)
                if ( n% 7 = = 0)
                        cout << n << “    “;
}
void main ( )
{
        int a , b;
        cin >> a >> b;
        if ( a < b )
                multiple ( a, b );
```

```
        else
                multiple ( b , a );
}
```

مثال۴- دربرنامه زیر ضرایب یک معادله درجه دوم دریافت می‌شود و توسط یک زیربرنامه، معادله به شکل متداول در ریاضیات با خلاصه‌نویسی‌های معمول چاپ می‌شود. (مثلاً $x^2 - 6x + 5 = 0$) . یک زیربرنامه دیگر معادله را حل می‌کند.

```
. . .
void print ( float a, float b, float c)
{
        if ( a = = 0 )
        {
                cout << “Error”;
                return;
        }
        if (a = = -1)
                cout << “-”;
        else
                if (a! = 1)
                        cout << a;
        cout << “x^2”;
        if ( b )
        {
                If ( b > 0 )
                        cout << “+”;
                if  ( b = = -1 )
                        cout << “-”;
                else
                        if (b != 1)
                                cout << b;
                cout << “x”;
        }
        if ( c > 0 )
                cout << “+” << c;
        if ( c < 0 )
                        cout << c;
        cout << “ = 0”;
}
```

```
void solve ( float a, float b, float c)
{
        float delta, x1, x2;
        if ( a = = 0)
                return;
        delta = b*b – 4*a*c;
        if ( delta<0 )
                cout << “Not Root!”;
        if ( delta = = 0 )
        {
                x = -b / (2*a);
                cout << “x1 = x2 = ” <<  x1;
        }
        if ( delta > 0 )
        {
                x1 = (-b + sqrt(delta)) / (2*a);
                x2 = (-b - sqrt(delta)) / (2*a);
                cout << “x1 = ” << x1 << “  x2 =  ” << x2;
        }
}
void main ( )
{
        float a, b, c;
        cin >> a >> b >> c;
        print (a , b , c);
        solve (a , b , c);
}
```

نکته۱- یک آرایه را می‌توان به عنوان پارامتر به تابع تحویل داد. در تعریف تابع باید نام آرایه همراه با [ ] ذکر شود. در هنگام فراخوانی نام آرایه بدون [ ] می‌آید.

مثال: مرتب‌سازی یک آرایه به کمک یک تابع

```
void sort ( int A[], int n)
{
        .
        .
        .
}
```

```
void main ( )
{
        int list [100], N;
        .
        .
        sort (list , N);
        .
        .
}
```

نکته۲- در هر تابع درصورتی که دستور return اجرا شود، تابع از همان جا قطع می‌شود و اجرا به فراخواننده برمی‌گردد.

مثال- در این برنامه به کمک یک تابع بررسی می‌شود که آیا یک تاریخ داده‌شده معتبر است یا نه (مثلاً تاریخ‌های ۴/۳۱/ و ۹/۱۶/ معتبر هستند اما تاریخ‌های ۱۴/۲۰/ و ۵/۳۹/ و ۸/۳۱/ معتبر نیستند).

```
. . .
int isOK ( int d, int m )
{
        if ( m > 12 || m < 1 )
                return 0;
        if ( d > 31 || d < 1)
                return 0;
        if ( m > 6 && d = = 31 )
                return  0;
        if  (m = = 12 && d = = 30)
                return 0;
        return 1;
}
void main ( )
{
        int day , month;
        cin >> day >> month;
        if ( isOK (day , month) )
                cout << “Date is valid.”;
```

```
        else
                cout << “Date is invalid.”;
}
```

متغیرهای محلی (local) و سراسری (global)

- متغیرهای محلی متغیرهایی هستند که در داخل یک تابع تعریف شده‌اند. این متغیرها فقط در محدوده همان تابع، شناخته‌شده هستند.
- متغیرهای سراسری متغیرهایی هستند که در ابتدای برنامه قبل از همه توابع تعریف شده‌اند. این متغیرها در سراسر برنامه شناخته‌شده هستند و به عبارت دیگر کلیه توابع به این متغیرها دسترسی دارند.

مثال: در برنامه زیر متغیرهای محلی مربوط به هر تابع و متغیرهای سراسری را تعیین کنید.

```
using namespace std;
#include “iostream”
int  a, x;
float m;
… f1 ( … )
{
        int  k, p;
        char c;
        .
        .
}
… f2 ( … )
{
        int  sum;
        float a, t;
        .
        .
}
void main ( )
{
        int  k, z;
        char d;
        .
        .
}
```

نکته: برنامه مثال ۴ قبلی(برنامه مربوط به معادله درجه دوم) را با استفاده از متغیرهای سراسری به صورت زیر هم می‌توان نوشت. البته این روش، انعطاف‌پذیری برنامه را از بین می‌برد و توصیه نمی‌شود.

```
using namespace std;
#include “iostream”
float a, b, c;
void print( )
{
        .
        .
        .
}
void solve( )
{
        .
        .
        .
}
void main( )
{
        cin >> a >> b >> c;
        print( );
        solve( );
}
```

آرایه‌های دوبعدی

آرایه دوبعدی ساختمان داده‌ای است که برای نگهداری و پردازش اطلاعاتی که به شکل جدول (ماتریس) هستند، استفاده می‌شود.

تعریف (اعلان) آرایه دوبعدی، مثال :

```
int A[10][16];
float Table[7][5];
char M[100][21];
```

برای ارجاع به عنصر خاصی از آرایه دوبعدی، ابتدا نام آرایه و به دنبال آن شماره سطر و شماره ستون هرکدام داخل کروشه می‌آید. مثلا:

```
if ( A[4][8] > 0 )
        . . .
```

در آرایه دوبعدی هم مانند آرایه یک بعدی، اندیس (شماره سطرها و ستونها) از صفر شروع می‌شود مثلا عنصر A[4][8] در واقع عنصرسطر پنجم و ستون نهم است.

مثال ۱: در برنامه زیر تعداد بازدیدکنندگان یک سایت وب در هر ساعت از هر روز یک ماه ۳۰ روزی دریافت می‌شود و برنامه باید:

اولاً مشخص کند که ساعت ۱۰ کدام روز بیشترین بازدیدکننده را داشته است.

ثانیاً مجموع بازدیدکنندگان هر روز را به تفکیک تعیین کند.

```
. . .
void main ()
{
        int A[30][24], i, j, sum, imax;
        for(i=0; i<30; i++)
                for(j=0; j<24; j++)
                        cin >> A[i][j];
        imax=0;
        for(i=1; i<30; i++)
                if( A[i][10] > A[imax][10] )
                                        imax=i;
        cout << imax+1;
```

```
        for(i=0; i<30; i++)
        {
                sum=0;
                for(j=0; j<24; j++)
                        sum+=A[i][j];
                cout << "Day " << i+1 << "sum=" << sum <<"\n";
        }
}
```

مثال۲: در این برنامه ابتدا یک ماتریس مربعی ۱۰*۱۰ دریافت می‌شود و موارد زیر تعیین می‌شود:

۱- مجموع عناصر قطر اصلی

۲- مجموع عناصر قطر فرعی

۳- مجموع عناصر پایین قطر اصلی

۴- مجموع عناصر بالای قطر اصلی

```
. . .
void main()
{
        float M[10][10];
        int sum1=0, sum2=0, sum3=0, sum4=0;
        for(int i=0; i<10; i++)
                for(int j=0; j<10; j++)
                        cin >> M[i][j];
        for(i=0; i<10; i++)
                sum1+=M[i][i];
        for (i=0; i<10; i++)
                sum2 += M[i][9-i];
        for(i=0; i<10; i++)
                for(j=0; j<i; j++)
                        sum3 += M[i][j];
        for (i=0; i<10; i++)
                for(j=i+1; j<10; j++)
                        sum4 += M[i][j];
        cout << sum1 << "   "<< sum2 << "   " << sum3 << "   " <<  sum4;
}
```

مثال ۳: در برنامه زیر ابتدا مقدار m، n و p دریافت می‌شود ($m,n,p \leq 20$). سپس عناصر ماتریس‌های $A_{m\times n}$ و $B_{n\times p}$ گرفته می‌شود و ماتریس $A\times B$ تعیین می‌شود.

```
. . .
void main()
{
        int A[20][20], B[20][20], C[20][20];
        int  i, j, k, m, n, p;
        cin >> m >> n >> p;
        for(i=0; i<m; i++)
                for(j=0; j<n; j++)
                        cin >> A[i][j];
        for(i=0; i<n; i++)
                for(j=0; j<p; j++)
                        cin >> B[i][j];
        for( i=0; i<m; i++)
                for(j=0; j<p; j++)
                {
                        C[i][j]=0;
                        for(k=0; k<n; k++)
                                C[i][j] += A[i][k]*B[k][j];
                }
        for(i=0; i<m; i++)
        {
                for(j=0; j<p; j++)
                        cout << C[i][j] << "   ";
                cout << endl;
        }
}
```

مثال۴: [**تحویل آرایه دو بعدی به تابع**]: میزان تولید یک کارخانه در ماه‌های مختلف ده سال اخیر از کاربر دریافت می‌شود. می‌خواهیم به کمک یک تابع مشخص کنیم که آیا میزان تولید سال‌های مختلف همواره صعودی بوده است.

```
. . .
int Check ( int Table[][12] )
{
        int oldsum=0, sum;
        for(int j=0; j<12; j++)
                oldsum +=Table[0][j];
        for(int i=1; i<10; i++)
        {
                sum = 0;
                for( int j=0; j<12; j++ )
                        sum += Table[i][j];
                if( sum < oldsum )
                        return 0;
                oldsum = sum;
        }
        return 1;
}
void main()
{
        int A[10][12];
        for(int y=0; y<10; y++)
                for(int m=0; m<12; m++)
                        cin >> A[y][m];
        if( Check(A) )
                cout << "Yes" ;
        else
                cout<<"No";
}
```

کاراکترها

هریک از حروف کوچک، حروف بزرگ، رقم‌ها و علامت‌های دیگری مانند !، #، <، Esc، Enter و ... یک کاراکتر محسوب می‌شوند.

برای نمایش هر کاراکتر در حافظه کامپیوتر از کد مخصوص آن استفاده می‌شود. یکی از روش‌های اصلی کدگذاری کد ASCII نام دارد. در این روش به هر کاراکتر یک کد یک‌بایتی اختصاص یافته است. به عنوان نمونه، کد تعدادی از کاراکترها به صورت زیر است:

NULL	0
Enter	13
Space	32
‘0’ , ‘1’, …, ‘9’	48, 49, …, 57
‘A’, ’B’, ..., ’Z’	65,66,…,90
‘a’, ’b’, …, ’z’	97,…,122

البته روش جدیدتر کدگذاری unicode نام دارد که برای هر کاراکتر دو بایت در نظر می‌گیرد و برای کاراکترهای کلیه زبان‌ها و خط‌های مختلف، کد خاصی دارد.

توابع کتابخانه‌ای مهم به کاراکترها:

توضیح: اکثر این توابع در فایل ctype.h تعریف شده‌اند.

```
char getch()

char getche()

int isalpha(char x)

int isdigit(char x)

int islower(char x)

int isupper(char x)

char tolower(char x)

char toupper(char x)
```

برنامه نویسی بر روی کاراکترها

مثال۱- پیاده‌سازی تابع isalpha

```
int myisalpha ( char x )
{
        if ( x >= 'A' && x <= 'Z' || x >= 'a' && x <= 'z')
                return 1;
        return 0;
}
```

مثال۲- تبدیل یک کاراکتر رقمی به معادل عددی آن

```
char x;
int a;
x = getche();
a = x - 48;    //  or:   a = x - '0';
```

مثال۳- پیاده‌سازی تابع toupper

```
char mytoupper ( char x )
{
        if  ( x >= 'a' && x <= 'z' )    // or if ( x >= 97 && x <= 122 ) or if ( islower ( x ) )
                return x - 32;       // or return  x - ( 'a' - 'A' );
        return x;
}
```

رشته‌ها

یک رشته دنباله ای از کاراکترها است مانند نام یک شخص، یک لغت، یک جمله و.... در زبان C نوع‌داده مستقلی با عنوان رشته وجود ندارد بلکه رشته به‌صورت آرایه‌ای از کاراکترها تعریف می‌شود مثلا:

```
char name [10];
char W[15];
```

در انتهای کاراکترهای واقعی رشته، کاراکتر NULL (کاراکتر تهی یا پوچ با کدصفر) ذخیره می‌شود. مثلا اگر محتوای متغیر Name برابر با "Reza" باشد، در حافظه به صورت زیر خواهد بود:

R	e	z	a	NULL					
0	1	2	3	4	5	6	7	8	9

توابع کتابخانه ای مهم مربوط به رشته ها

توضیح: اکثر این توابع در فایل string.h تعریف شده‌اند.

strcpy(s1, s2)

strcat (s1, s2)

strlen (s)

strcmp (s1, s2)

مثال ۱- strcmp(“bicycle”,”cat”)

مثال۲- strcmp(“worshop”,” work”)

مثال۳- strcmp(“Hello”,”Hello”)

مثال۴- strcmp(“Door”,”cat”)

strchr (s, ch)

strlwr (s)

strupr (s)

strrev (s)

strstr (s1, s2)

gets (s)

puts (s)

atoi (s)

atof (s)

itoa (v, s, r)

برنامه نویسی بر روی رشته‌ها

مثال۱- می‌خواهیم برنامه‌ای بنویسیم که هر بار نام یک کاربر را دریافت کند و پیام خوش‌آمد مناسبی خطاب به او چاپ کند. مثلاً اگر نام کاربر maryam داده شود، پیام باید به صورت زیر باشد:

Welcome Maryam

مانند نمونه فوق در خروجی باید نام کاربر با حرف بزرگ شروع شود و بقیه حروف آن کوچک باشد. هرگاه به جای نام کاربر "***" وارد شود، باید برنامه پایان یابد.

```
using namespace std;
#include "iostream"
#include “stdio.h”
#include “string.h”
#include “ctype.h”
void main ( )
{
        char name [ 12 ], message [ 20 ];
        clrscr();
        while ( 1 )
        {
                cin >> name;        //  یا:  gets ( name );      یا:  scanf(“%s”, name);
                if ( ! strcmp ( name , “***” ) )
                        break;
```

```
            strlwr(name);
            name [ 0 ] = toupper ( name [ 0 ] );
            strcpy ( message, “Welcome  ”);
            strcat ( message, name );
            cout << message;
        }
}
```

مثال۲- پیاده سازی تابع strlen

```
int mystrlen ( char s [ ] )
{
        int  i = 0;
        while ( s [ i ] != NULL)     //  یا:  while ( s [ i ] != ’\0’ )       یا:   while ( s [ i ] )
                i++;
        return i;
}
```

مثال ۳- می‌خواهیم برنامه‌ای بنویسیم که یک عبارت رشته‌ای شامل دو عدد طبیعی و نماد یکی از چهار عمل اصلی بین آن دو را به صورت پارامتر تحویل بگیرد وحاصل عددی عبارت را تعیین کند. مثلاً اگر عبارت پارامتر "345+26" باشد، برنامه باید مقدار 371 را چاپ کند.

```
. . .
int compute ( int x, int y, char oper )
{
        if ( oper = = '+' )    return x + y;
        if ( oper = = '-' )    return x - y;
        if ( oper = = '*' )    return x * y;
        if ( oper = = '/' )    return x / y;
}
```

// **روش اول:**

```
void main ( )
{
        char S[ 10 ];
```

```
        int  a=0, b=0, j;
        char op;
        clrscr();
        cin >> S;
        for ( int i =0; isdigit ( S [ i ]; i++)
        {
                int k = S [ i ] - 48;
                a = a * 10 + k;
        }
        op = S [ i ];
        for ( i++; S[ i ] != NULL; i++ )
        {
                int k = S [i] – 48 ;
                b = b * 10 + k ;
        }
        cout << compute (a, b, op ) ;
        getch ( ) ;
}
```

// **روش دوم:**

```
void main ( )
{
        char S [ 10 ], S1 [ 10 ], S2 [ 10 ];
        int a, b;
        char op;
        clrscr ( );
        cin >> S ;
        for ( int i = 0; isdigit ( S [ i ] ) ; i++)
                S1 [ i ] = S [ i ];
        S1 [ i ] = NULL;
        a =atoi ( S1 );
        op = S [ i ];
        for (i++, j=0; s [ i ] != NULL; )
```

```
            S2 [ j++ ] = S [ i++ ];
        S2 [ i ] = NULL;
        b = atoi ( S2 );
        cout << compute ( a, b, op );
}
```

مثال۴– در این برنامه نام تعدادی شهر به صورت آرایه‌ای از رشته‌ها (یک آرایه دوبعدی از کاراکترها) دریافت می‌شود و اولاً نام "Mashhad" در صورت وجود حذف می‌شود و ثانیاً تعداد شهرهایی که نام آنها به "abad"ختم می‌شود، مشخص می‌شود. تعداد شهرها حداکثر ۵۰ و نام هر شهر حداکثر ۲۰ کاراکتر فرض می‌شود.

```
. . .
void main ( )
{
        char C [ 50 ] [ 21 ];
        int n , cnt = 0, i, j;
        clrscr ( );
        cin >> n;
        for ( i=0; i<n; i++)
                cin >> C [ i ] ;
        for ( i=0; i<n; i++)
                if ( ! strcmp ( C[i], “Mashhad” ) )      //  if  C[i] = ”Mashhad” یعنی
                {
                        for ( int j= i ; j < n-1; j++)
                                strcpy ( C[ j ], C [ j + 1 ] );
                        n --;
                }
        for ( i = 0; i < n; i++ )
                cout << C[i] << "   ";
        cout << ” \n\n\n ”;
        for ( i = 0; i < n; i++ )
        {
            int k = strlen ( C [ i ] );
            if ( C[ i ] [ k – 1 ] == ’d’ && C[ i ] [k – 2 ] = = ’a’ && C  [ i ] [k - 3] = = ’b’ && C [ i ] [k -4] == ’a’)
                        cnt++;
        }
        cout << cnt;
```

}

برنامه‌های سری اول:

۱- برنامه‌ای بنویسید که یک جمله را به صورت یک رشته دریافت کند و مشخص کند که چند کلمه دارد. فرض کنید بین هر دو کلمه یک یا چند فضای خالی وجود دارد.

۲- تابعی مشابه strstr پیاده‌سازی کنید. تابع شما باید در صورت پیداکردن رشته دوم در رشته اول، اندیس محل پیداشدن و در غیر این صورت عدد1- را برگرداند.

ساختارها (ساختمان‌ها) (structures)

یک **ساختار** مجموعه‌ای از داده‌ها است که به هم مربوطند اما نوع آنها الزاماً یکی نیست. مثلاً اطلاعات یک کارمند شامل نام، شماره پرسنلی، کد جنسیت و ... را می‌توان به صورت یک ساختار نمایش داد. هریک از اجزای تشکیل‌دهنده یک ساختار (مثل نام، شماره پرسنلی و ...) یک **فیلد** نامیده می‌شود.

تعریف ساختاردر زبان C

مثال- ساختار مربوط به مشخصات کارمند را می‌توان به صورت زیر تعریف کرد:

```
struct Employee
{
        char name [ 20 ];
        int ID;
        char code;
};
```

Employee درواقع یک نوع است و اکنون می توان از این نوع، متغیر تعریف کرد:

```
Employee x, temp;
```

برای استفاده از یک فیلد خاص از یک متغیر ساختاری ابتدا نام متغیر ساختاری و سپس نقطه و نام فیلد مورد نظر ذکر می‌شود، مثلاً:

```
if ( x.code == 'F' )
        . . .
```

مثال ۱- در یک مناقصه، n شرکت حضور دارند که برای هر شرکت، اطلاعاتی شامل نام شرکت، نوع شرکت ('G' به معنی دولتی و 'N' به معنی غیردولتی) و مبلغ پیشنهادی داده می‌شود. برنامه زیر با دریافت این اطلاعات، شرکت برنده را تعیین و مشخصات آن را چاپ می‌کند.

```
. . .
void main()
{
        struct  Co
        {
                char name [ 21 ];
                char Type;
                int price;
        };
        Co A , Winner;
        int n;
        cout << " n = ? ";
```

```
        cin >> n;
        Winner.price = 1000000;
        for ( int i=1 ; i<=n ; i++ )
        {
                cout << " \n\n Enter next co: ";
                cout << " name = ? ";           cin >> A.name;
                cout << " Type (G/N)? ";        A.Type = toupper( getche() );
                cout << "  price = ? ";         cin >> A.price;
                if ( A.price < Winner.price )
                        Winner = A;
        }
        cout << " \n\n Winner name:" << Winner.name << " Price: " << Winner.price;
        if ( Winner.Type == 'G' )
                cout << "شرکت برنده یک شرکت دولتی است" ;
        else
                cout << " شرکت برنده یک شرکت غیر دولتی است" ;
}
```

مثال۲: [**آرایه‌ای از ساختارها**] در مسابقات برنامه‌نویسی ACM/ICPC هر تیمی که تعداد مسائل حل‌شده‌اش بیشتر باشد، رتبه بالاتری دارد. در مورد تیم‌هایی که تعداد مسائل حل‌شده آنها یکسان است، تیمی که تأخیر کمتری در ارسال پاسخ‌ها داشته است، رتبه بالاتری کسب می‌کند. در برنامه زیر، مشخصات تیم‌ها به صورت یک آرایه از ساختارهای سه‌فیلدی (شامل نام تیم، تعداد مسائل حل‌شده و تأخیر در ارسال پاسخ) دریافت می‌شود و لیست تیم‌ها براساس ملاک‌های فوق مرتب می‌شود.

```
…
struct team
{
        char name [16];
        int solved, delay;
} List [ 100 ] ;
void swap ( team &x, team &y )
{
        team temp;
           temp = x;
           x = y;
           y = temp;
}
```

```
void sort ( team A[], int n )
{
        for ( int i = 0 ; i < n-1 ; i++ )
        {
                int max = i;
                for ( int j = i+1; j < n ; j++ )
                        if ( A[j].solved > A[max].solved ||
                                A[j].solved == A[max].solved && A[j].delay < A[max].delay )
                                                                        max = j;
                swap ( A[i], A[max] );
        }
}
void main()
{
        int n;
        cout << " n = ? "
        cin >> n;
        for ( int i=0; i<n; i++ )
        {
                cout << "\n Please enter next team:";
                cin >> list[i].name >> list[i].solved >> list[i].delay;
        }
        sort ( list, n );
        for ( i=0; i<n; i++ )
                cout << list[i].name << "  " << list[i].solved << "  " << list[i].delay << "\n";
}
```

اشاره‌گرها (Pointers)

یک اشاره‌گر نوع خاصی از متغیر است که آدرس محل خاصی از حافظه را در خود دارد. به عنوان مثال فرض کنید x یک متغیر int معمولی باشد. آدرس محل ذخیره‌سازی x در حافظه می‌تواند در یک متغیر از نوع اشاره‌گر مثل p نگهداری شود. در این صورت اصطلاحاً گفته می‌شود p به x اشاره می‌کند.

مثال‌هایی از تعریف متغیر از نوع اشاره‌گر:

```
int *p;

float *q;
```

عملگرهای & و *

& یعنی «آدرسِ . . .»
* یعنی «محتوای محل مورد اشاره‌ی . . .»

بنابراین اگر x یک متغیر int با مقدار 15 و p یک اشاره‌گر باشد و دستور زیر اجرا شود:

```
p = & x;
```

در این صورت، آدرس x در حافظه در متغیر اشاره‌گری x قرار می‌گیرد. اکنون اگر دستور زیر اجرا شود:

```
cout << *p;
```

محتوای محل مورد اشاره‌ی p که همان مقدار x یعنی 15 است، چاپ خواهد شد.

به طور کلی همواره قاعده زیر برقرار است:

p = &x ➔ *p = x

عملگرهای دیگر روی اشاره‌گرها:

۱) مقایسه:

(p1 == p2) if یعنی اگر آدرس واقع در p1 با آدرس واقع در p2 برابر است (به عبارت دیگر p1 و p2 هر دو به یک محل از حافظه اشاره می‌کنند)

(p1 < p2) if یعنی اگر آدرس واقع در p1 از آدرس واقع در p2 کوچکتر است (به عبارت دیگر محلی که p1 به آن اشاره می‌کند، در حافظه قبل‌تر از محلی است که p2 به آن اشاره می‌کند)

مثال ۱) کدام یک از دو نتیجه‌گیری زیر درست است:

الف) p1 = p2 ➔ *p1 = *p2

الف) *p1 = *p2 ➔ p1 = p2

جواب:

مثال ۲) فرض کنید x و y دو متغیر int به ترتیب با مقادیر 10 و 20 باشد که به ترتیب در آدرس‌های 5000 و 3000 حافظه ذخیره شده‌اند. اگر دستورهای p1 = &x; و p2 = &y; اجرا شود، کدام یک از شرط‌های زیر برقرار خواهد بود:

الف) if(p1 < p2)

. . .

الف) if(*p1 < *p2)

. . .

جواب:

۲) انتساب:

دستور p2=p1; باعث می‌شود که آدرس واقع در p1 در p2 کپی شود. به عبارت دیگر بعد از اجرای این دستور، اشاره‌گر p2 به همان جایی اشاره خواهد کرد که p1 اشاره می‌کند.

۳) افزایش و کاهش:

هر واحد افزایش و کاهش یک متغیر اشاره‌گری، مقدار آدرس واقع در آن را به اندازه سایز نوعی که اشاره‌گر مذکور به آن اشاره می‌کند، افزایش یا کاهش می‌دهد.

مثال) فرض کنیم:

```
int *p;
p = &x;
```

اگر فرض کنیم x در ادرس ۳۰۰۰ واقع است، پس فعلاً مقدار p برابر با ۳۰۰۰ خواهد بود. اکنون اگر دستور p=p+5; اجرا شود، مقدار p برابر خواهد شد با ۳۰۱۰، زیرا هدف از دستور فوق این است که p به اندازه ۵ عدد int جلوتر برود و می‌دانیم هر int دو بایت است.

استفاده نادرست از اشاره گرها:

اگر اشاره‌گری تعریف کنیم ولی آدرس متغیر خاصی را در آن قرار ندهیم، درصورتی‌که محتوای محل مورد اشاره آن را مقداردهی کنیم، نادرست خواهد بود.

مثال:

```
int *p;
*p = 241;
```

توضیح مثال فوق: در متغیر اشاره‌گری p یک مقدار تصادفی ممکن است وجود داشته باشد، به عبارت دیگر p به محل نامشخصی از حافظه اشاره کند. دستور انتساب باعث می‌شود که در آن محل نامشخص عدد ۲۴۱ قرار گیرد.

کاربردهای اشاره‌گرها

۱- کاربرد اشاره‌گرها در تغییرپذیرکردن پارامتر توابع:

در حالت عادی (بدون استفاده از اشاره‌گرها)، وقتی یک تابع فراخوانی می‌شود، پارامترهایی که در اختیار آن قرار می‌گیرد، در واقع فقط مقدار پارامترهای اصلی تابع فراخواننده را دارند و به عبارت دیگر کپی پارامترهای واقعی در اختیار تابع فراخوانی‌شده قرار می‌گیرد. در نتیجه اگر تابع فراخوانی‌شده مقدار جدیدی در پارامترهای خود قرار دهد، این تغییرات فقط روی کپی پارامترهای واقعی اعمال خواهد شد و پارامترهای واقعی تغییر نخواهند کرد. به این حالت تحویل پارامتر به تابع، فراخوانی با تحویل مقدار (Call by value) می‌گویند.

مثال ۱- خروجی برنامه زیر چیست؟

```
void  test ( int a )
{
        a ++;
        cout << "In function test: " << a;
}
```

```
void main()
{
        int x = 8;
        test ( x );
        cout << "\n After function: " << x;
}
```

جواب:

اما درمواردی که بخواهیم تابع بتواند مقدار پارامتر حقیقی را تغییر دهد، باید به جای مقدار (کپی)، آدرس (اشاره‌گر) پارامتر را در اختیار تابع قرار دهیم. دراین صورت تابع به محل واقعی پارامتر اصلی مراجعه می‌کند و می‌تواند اصل آن را دستکاری کند. به این حالت تحویل پارامتر به تابع، فراخوانی با ارجاع یا فراخوانی با تحویل آدرس (Call by reference) می‌گویند.

مثال۲- خروجی برنامه زیر چیست؟

```
Void  temp ( int a, int *b )
{
        a += 5;
        *b += 5;
        cout << "In function: first number is " << a << " second is " << *b;
}
void main()
{
        int x=2, y=2;
        int *p;
            p = &y;
            temp ( x, p);
            cout << "\n After function: first number is " << x << " second is " << y;
}
```

جواب:

نکته: در کامپایلرهای جدیدتر برای تحویل پارامترها به تابع به‌صورت اشاره‌گری، کافی است که فقط در تعریف تابع، قبل از نام پارامتر، علامت & قرار گیرد و در فراخوانی، پارامتر بصورت عادی ذکر می‌شود. مثلاً برنامه فوق به صورت زیر می‌تواند نوشته شود:

```
void temp ( int a, int &b )
{
        a += 5;
        b += 5;
        cout << "In function: first number is " << a << " second is " <<  b;
}
void main()
{
        int x=2, y=2;
            temp ( x, y );
            cout<<"\n After function: first number is " << x << " second is " << y;
}
```

مثال۳- می‌خواهیم برنامه‌ای بنویسیم که یک عدد را دریافت کند و به کمک یک تابع، هم مجموع و هم حاصل‌ضرب رقم‌های آن را تعیین کند.

```
. . .
void Process ( int N, int &s, int &m )
{
        s =0;    m =1;
        while ( N )
        {
                s = s + ( N%10 );
                m = m * ( N%10 );
                N /= 10;
        }
}
```

```
void main()
{
        int number, sum, mult;
        cin >> number;
        Process ( number, sum, mult );
        cout << "For the number " << number << " sum: " << sum << " mult: " << mult;
}
```

همان‌طور که مثال اخیر نشان می‌دهد، با استفاده از اشاره‌گرها می‌توانیم تابع را طوری بنویسیم که بیشتر از یک مقدار برگرداند (البته به طور غیر مستقیم، نه با استفاده از return).

۲- کاربرد اشاره‌گرها در تخصیص حافظه به صورت پویا:

با استفاده از اشاره‌گرها می‌توانیم آرایه‌هایی با اندازه پویا تعیین کنیم به این معنی که در زمان اجرای برنامه ابتدا اندازه آرایه مورد نظر از کاربر دریافت شود و یا به طریق دیگری توسط برنامه تعیین شود و سپس در حین اجرا برای آرایه به این اندازه حافظه تخصیص دهیم.

برای تخصیص حافظه به صورت پویا (در حین اجرای برنامه) از دستور new استفاده می‌کنیم. این دستور حافظه مورد نظر را تخصیص می‌دهد و آدرس (اشاره‌گر) ابتدای فضای تخصیص‌یافته را در اختیار برنامه قرار می‌دهد. البته این امکان وجود دارد که تخصیص حافظه با شکست مواجه شود که در این صورت، دستور new اشاره‌گر NULL (تهی) را برمی‌گرداند.

مثال- در این برنامه، ابتدا تعداد عناصر یک لیست از کاربر دریافت می‌شود. سپس یک آرایه پویا به این اندازه ایجاد می‌شود و عناصر آن از کاربر گرفته می‌شود. برنامه باید تعیین کند که آیا این لیست، متقارن است یا نه.

```
. . .
void main()
{
        int *A;
        int n;
        cin >> n;
        A = new int ( n );      // یا: A = new int [ n ];
        if ( A == NULL )
```

```
        {
                cout << "Sorry! Cannot allocate memory.";
                exit ( 1 );
        }
        for ( int i = 0; i < n; i++ )
                cin >> A[i];
        int m = 1;
        for ( int i=0, j=n-1; i < j; i++, j--)
                if ( A[i] != A[j] )
                        m = 0;
        if( m )
                cout << "Yes ";
        else
                cout << "No";
}
```

لیست پیوندی

برای ذخیره سازی یک لیست دو روش وجود دارد:

۱) آرایه (ذخیره سازی ترتیبی): عناصر آرایه در خانه‌های متوالی حافظه ذخیره می‌شوند. یک مزیت آرایه سادگی کار با آن است. مزیت دیگر این است که با داشتن اندیس یک عنصر، امکان مراجعه مستقیم به آن وجود دارد.

۲) لیست پیوندی: یک لیست پیوندی از تعدادی عنصر (نود) تشکیل می‌شود که هر عنصر یک ساختار است که در آن علاوه بر فیلدهای داده‌ای معمول، حداقل یک فیلد از نوع اشاره‌گر وجود دارد که آدرس عنصر بعدی لیست در حافظه را نشان می‌دهد. کافی است در برنامه، آدرس اولین عنصر را داشته باشیم، در این صورت می‌توانیم با دنبال‌کردن زنجیره اشاره‌گرها به همه عناصر لیست، دسترسی پیدا کنیم.

فیلد آدرس بعدی در آخرین عنصر برابر با NULL است.

یک مزیت لیست پیوندی، مدیریت حافظه به صورت پویا است به این ترتیب که هر زمان که عنصر جدیدی لازم است، برای آن حافظه اختصاص می‌دهیم و با تنظیم اشاره‌گرها آن را در موقعیت مناسب در لیست درج می‌کنیم و هر زمان که به عنصری نیاز نباشد، با کنار گذاشتن آن از لیست، فضای مربوط به آن در حافظه را آزاد می‌کنیم.

مزیت دیگر لیست پیوندی این است که برای اضافه‌کردن یا حذف یک عنصر بر خلاف آرایه نیاز به شیفت‌دادن عناصر دیگر که زمانبر است، نیست، بلکه فقط کافی است که اشاره‌گرها به طور مناسبی تغییر کنند. مثلاً برای اضافه‌کردن عنصر جدیدی بین عناصر پنجم و ششم کافی است که آدرس عنصر ششم را در فیلد اشاره‌گر عنصر جدید قرار دهیم و آدرس عنصر جدید را در فیلد اشاره‌گر عنصر پنجم قرار دهیم. همچنین مثلاً برای حذف عنصر دهم کافی است که آدرس عنصر یازدهم را در فیلد اشاره‌گر عنصر نهم قرار داده و به این ترتیب عنصر دهم از لیست کنار گذاشته می‌شود.

انواع لیست پیوندی

لیست پیوندی ممکن است یک‌طرفه (تک‌پیوندی) یا دوطرفه (دوپیوندی) باشد. در لیست تک‌پیوندی هر عنصر فقط آدرس عنصر بعدی را نگهداری می‌کند و فیلد اشاره‌گر در آخرین عنصر NULL است، درحالیکه در یک لیست دوپیوندی هر عنصر هم آدرس عنصر قبلی و هم آدرس عنصر بعدی را نگهداری می‌کند و فیلد آدرس بعدی در آخرین عنصر و نیز فیلد آدرس قبلی در اولین عنصر NULL است. از طرف دیگر لیست‌های پیوندی می‌توانند ساده یا حلقوی باشند. در لیست‌های حلقوی، فیلد آدرس آخرین عنصر به جای آنکه NULL باشد، به‌صورت حلقوی به اولین عنصر اشاره می‌کند.

لیست تک‌پیوندی ساده

لیست دوپیوندی ساده

لیست تک‌پیوندی حلقوی

لیست دوپیوندی حلقوی

از طرف دیگر ممکن است عناصر یک لیست مرتب یا نامرتب باشند که این مسأله در اضافه‌کردن و جستجوی عناصر تأثیر دارد.

برنامه پیاده‌سازی یک لیست تک‌پیوندی ساده مرتب

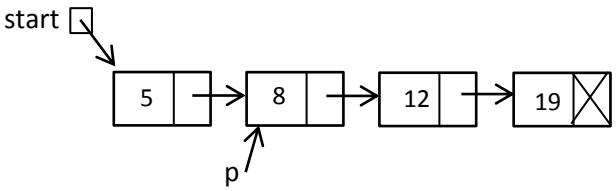
در اینجا یک لیست تک‌پیوندی ساده پیاده‌سازی می‌کنیم که هر عنصر آن ساختاری با دو فیلد است: یکی فیلد data از نوع عدد صحیح و دیگری فیلدی به نام next از نوع اشاره‌گر که برای نگهداری آدرس عنصر بعدی در نظر گرفته می‌شود.

یک متغیر اشاره‌گری به نام start برای نگهداری آدرس سر لیست (آدرس اولین عنصر) در نظر گرفته می‌شود که در ابتدا که لیست خالی است، مقدار NULL دارد.

قبل از بررسی برنامه به این نکته توجه کنید که اگر p اشاره‌گری به یک ساختار باشد، برای نام‌بردن از یک فیلد از ساختاری که p به آن اشاره می‌کند، از عملگر <- استفاده می‌کنیم.

چند مثال از کاربرد عملگر <- :

با توجه به لیست پیوندی زیر:



start -> data برابر است با:

p -> data برابر است با:

p -> next برابر است با:

p -> next -> data برابر است با:

دستور p = p -> next; باعث می‌شود که . . .

در برنامه زیر قرار است کاربر بتواند اعمال زیر را انجام دهد:

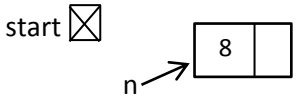
- مشاهده همه عناصر لیست
- اضافه کردن عنصر جدید به لیست
- حذف یک عنصر مورد نظر از لیست
- خروج از برنامه

```
. . .
struct node
{
        int data;
        node *next;
};
node *start = NULL;
```

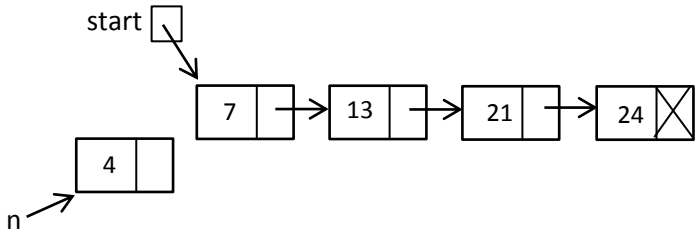
```
void Display()
{
        node *p;
        for ( p=start; p!=NULL; p=p->next )
            cout << p->data << "\n";
 }
```

حالت‌های اضافه‌کردن در لیست تک‌پیوندی

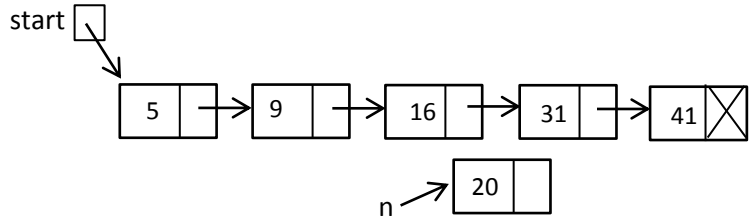
۱) در لیست خالی



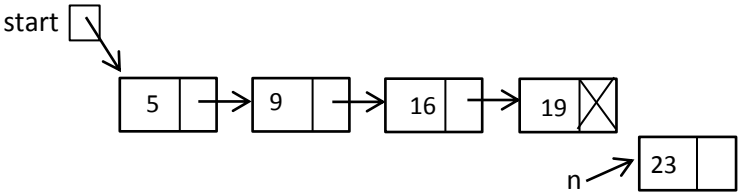
۲) در ابتدای لیست



۳) در وسط لیست



۴) در انتهای لیست

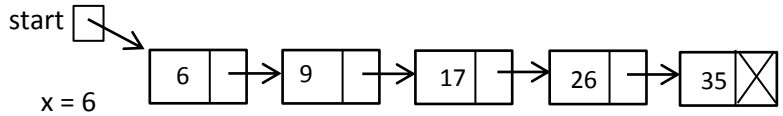



```
void Insert()
{
        node *p, *q, *n;
        int x;
          n = new node;
          if( !n )
          {
                cout << "\n Sorry! Cannot allocate memory.\n";
                return;
          }
```

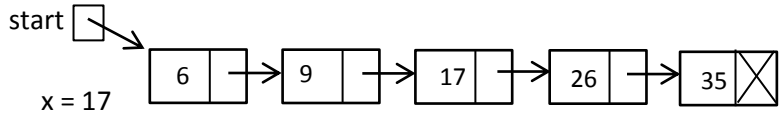
```
        cout << "\n Please enter x to be inserted:";
        cin >> x;
        n ->  data = x;
        if ( start == NULL ||  x < start -> data )
        {
                n -> next = start;
                start = n;
                return;
        }
        p = start -> next;
        q= start;
        while ( p!= NULL && p -> data < x )
        {
                p = p->next;
                q = q->next;
        }
        q -> next = n;
        n -> next = p;
}
```

حالت های حذف از لیست تک پیوندی

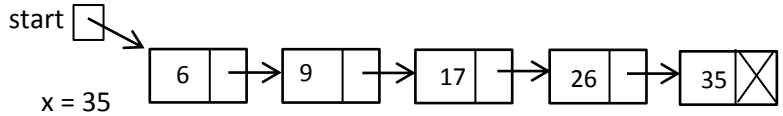
۱) حذف عنصر اول



۲) حذف از وسط لیست



۱) حذف عنصر آخر




```
void Remove()
{
        node *p, *q, *temp;
        int x;
        cout << "\n Please enter x to be removed:";
        cin >> x;
```

```
        if ( start == NULL ||  x < start ->data )
        {
                cout << "\n List is empty or x not found.\n";
                return;
        }
        if ( x == start -> data)
        {
                temp = start;
                start = start-> next;
                delete temp;
                return;
        }
        p = start -> next;
        q = start;
        while ( p != NULL && p -> data < x )
        {
                p = p -> next;
                q = q -> next;
        }
        if ( p == NULL  ||  p -> data > x )
        {
                cout << "\n Not found.\n";
                return;
        }
        q -> next = p-> next;
        delete p;
}
void main()
{
        while ( 1 )
        {
                cout <<"\n (D)isplay, (I)nsert, (R)emove, (E)xit ? \n";
                char x = toupper ( getche() );
                if ( x == 'D' )  Display();
```

```
            if ( x == 'I' )  Insert();
            if ( x == 'R' )  Remove();
            if ( x == 'E' )  exit( 0 );
        }
        getch();
}
```

لیست دوپیوندی

ساختار نود مناسب برای یک لیست دوپیوندی می‌تواند به‌صورت زیر باشد:

```
struct node
{
        int data;
        node *left, *right;
 };
```

پردازش روی لیست دوپیوندی تا حدی با لیست تک‌پیوندی فرق دارد. مثلاً برای درج عنصر جدیدی با اشاره گر n بعد از عنصری با اشاره‌گر p می‌توانیم دستوراتی به‌صورت زیر بنویسیم:

```
n -> left = p;
n -> right = p -> right;
if ( p-> right != NULL )
      p -> right -> left = n;
p -> right = n;
```

به عنوان یک نمونه دیگر، حذف اولین عنصر از یک لیست دوپیوندی می‌تواند به‌صورت زیر باشد:

```
temp = start;
start = start -> right;
if( start != NULL )
    start -> left = NULL;
delete temp;
```

توابع بازگشتی(Recursive Function)

در بعضی از زبان‌ها از جمله C این امکان وجود دارد که تابعی خود را فراخوانی کند (بازگشتی مستقیم)، یا این که دو تابع متقابلاً یکدیگر را فراخوانی کنند (بازگشتی غیرمستقیم). مثلاً بصورت زیر:

بازگشتی مستقیم:

```
void f(...)
{
        .
        .
        f(...);
        .
        .
}
```

بازگشتی غیرمستقیم:

```
void f1(...)
{
        .
        .
        f2(...);
        .
        .
}
void f2(...)
{
        .
        .
        f1(...);
        .
        .
}
```

تقریباً هر برنامه‌ای را که با حلقه‌های تکرار می‌توان نوشت، با روش بازگشتی هم می‌توان نوشت. اما درمواردی که ماهیت مساله بازگشتی است، استفاده از توابع بازگشتی پیاده‌سازی را ساده‌تر و خلاصه‌تر می‌کند.

اما یک عیب استفاده از توابع بازگشتی این است که فراخوانی‌های بازگشتی مکرر منجر به پرشدن سریع پشته سیستم می‌شود.

مثال۱) محاسبه فاکتوریل به روش بازگشتی

```
long int fact( int n )
{
        if ( n == 0 )
            return 1;
        //else
            return n * fact ( n-1 );
}
```

مثال۲) تعیین مجموع رقم‌های یک عدد

```
int sum( int A )
{
        if( A<10 )
            return A;
        int temp = sum ( A/10 );
        return temp + A%10;
}
```

مثال۳) محاسبه توان

روش اول:

```
float power ( float a, int b )
{
        if ( b == 0 )
                return 1;
        //else
                return a * power ( a, b-1 );
}
```

روش دوم:

```
float power ( float a, int b )
{
        if ( b == 0 )
                return 1;
        float temp = power ( a, b/2 );
        if ( b%2 == 0 )
                return temp * temp;
        else
                return temp * temp * a;
}
```

مثال۴) خروجی هریک از دو تابع زیر اگر با پارامتر 1 فراخوانی شوند چیست؟

```
void f1 ( int k )
{
        if ( k > 100 )
                return;
        cout << k << "   ";
        f1 ( 2*k );
}
```

```
void f2 ( int k )
{
	if ( k > 100 )
		return;
	f2 ( 2*k );
	cout  << k << "   ";
}
```

جواب: خروجی ;(1) f1:

خروجی ;(1) f2:

مثال۵) جستجوی دودویی با پیاده‌سازی بازگشتی

```
int bsearch ( float A [], float x, int start, int finish)
{
	if ( start <= finish)
	{
		int mid = ( start + finish ) / 2;
		if ( x == A[mid] )
			return mid;
		if ( x < A[mid] )
			return bsearch ( A, x, start, mid-1 );
		// if ( x > A[mid] )
			return bsearch ( A, x, mid+1, finish);
	}
	return -1;
}
```

برنامه‌های سری دوم

۱) برنامه‌ای بنویسید که هربار ضرایب یک معادله درجه دوم را از کاربر بگیرد و آن را حل کند. برای حل معادله یک تابع بنویسید که شش پارامتر داشته باشد: a، b، c (ضرایب معادله)، n (پارامتر تغییرپذیر برای تعداد ریشه‌ها)، x1 و x2 (پارامترهای تغییرپذیر برای ریشه‌های معادله). خروجی باید در تابع main() چاپ شود.

۲) برنامه‌ای بنویسید که عناصر یک آرایه را دریافت کند و به کمک یک تابع بازگشتی، بیشترین مقدار در آرایه را تعیین کند.

برنامه نویسی شی گرا با #C

مقدمه

این جزوه با هدف آموزش برنامه‌سازی شیءگرا با استفاده از زبان #C در سه بخش تنظیم شده است:

بخش اول شما را با #C و مفاهیم وابسته به آن از جمله .NET framework آشنا می‌کند.

بخش دوم به طور خلاصه تولید برنامه‌های کاربردی ویندوزی را توضیح می‌دهد. البته هدف اصلی این جزوه، آموزش برنامه‌نویسی شیءگرا است اما چون قرار است این کار با استفاده از محیط #C انجام شود ابتدا در این بخش، تکنیک‌های اساسی برنامه‌نویسی ویژوال در این محیط آموزش داده می‌شود.

بخش سوم به برنامه‌سازی شیءگرا و کار با کلاس‌ها و اشیا و... اختصاص دارد.

این جزوه برای آموزش درس برنامه نویسی شیءگرا تدوین شده است و چون دانشجویان در درس های قبلی با زبان ++C آشنا شده اند در این جزوه فرض شده است که قسمت هایی از #C که با ++C یکسان است، نیاز به توضیح ندارد.

بخش اول: آشنایی با #C و مفاهیم وابسته

برای تولید یک برنامه ویندوزی از یک محیط ویژوال استفاده می‌شود. درچنین محیطی بخش زیادی از کار تولید برنامه (به ویژه user interface برنامه یعنی ظاهر برنامه که کاربر از طریق آن با برنامه ارتباط برقرار می‌کند) با استفاده از امکانات بصری موجود و بدون نیاز به کدنویسی امکان‌پذیر است و در قسمت‌هایی هم که نیاز به کدنویسی است، امکانات و راهنمایی‌های مؤثری به برنامه‌نویس ارائه می‌شود. مایکروسافت یک محیط تولید برنامه ویژوال به نام Visual Studio عرضه کرده است. برنامه‌نویس در این محیط می‌تواند به یکی از چند زبان موجود برنامه‌نویسی کند که یکی از آنها #C است.

این زبان ها همه تحت پوشش چارچوب .NET. هستند. .NET. یک جزء نرم‌افزاری قابل اضافه‌شدن به ویندوز است که حل بسیاری از نیازهای نرم‌افزاری رایج را بصورت آماده در قالب یک کتابخانه کلاس ارائه می‌کند. این کتابخانه، مجموعه مفصلی از کلاس‌ها و متدهای آماده در زمینه‌های مختلف (رابط کاربر، ورودی/خروجی، شبکه، پایگاه داده و....) را دربردارد که این امکانات در همه زبان‌هایی که مبتنی بر .NET. هستند (از جمله #C)، قابل استفاده است.

کلاس های موجود در .NET. در قالب دسته هایی که «فضای نام»[1] نامیده می‌شوند، دسته‌بندی شده‌اند. هر فضای نام مجموعه‌ای از کلاس‌های مربوط به موضوع خاصی را شامل می‌شود. البته در مواردی هر فضای نام خود به چند فضای نام فرعی‌تر تقسیم می‌شود.

بخش دوم: تولید برنامه‌های کاربردی ویندوزی

۲–۱ مفاهیم مشترک در برنامه‌های کنسولی و برنامه‌های مبتنی بر فرم

[1] namespace

زبان #C از نظر دستور زبان شباهت‌های قابل توجهی با ++C دارد. مثلا بلاک‌ها، نوع‌داده‌ها، تعریف متغیرهای ساده، دستورات پایه انتساب، انتخاب و حلقه‌ها و عملگرها در دو زبان مشابه هستند، اما فرق‌های زیادی هم بین این دو مثلاً در استفاده از آرایه‌ها و رشته‌ها وجود دارد و مخصوصاً توابع کتابخانه‌ای ++C با کلاس‌ها و متدهای موجود در کتابخانه کلاس #C اساساً متفاوت است.

درباره مفهوم کلاس به طور مفصل در بخش سوم بحث خواهد شد. فعلاً به صورت خلاصه یک کلاس را به عنوان توصیف برنامه‌نویسی از یک مفهوم در نظر بگیرید که خصوصیت‌های نمونه‌های آن مفهوم و عملیات مربوط به آنها را پیاده‌سازی می‌کند. مثلاً کلاس Form برای ارائه مفهوم یک فرم ویندوزی استفاده می‌شود که خصوصیت‌های مختلف آن از قبیل موقعیت، اندازه و کنترل‌های روی آن و نیز عملیات مختلف از قبیل جابجایی، بستن و... را پیاده‌سازی می‌کند.

برای استفاده از امکانات کلاس‌های کتابخانه سی‌شارپ باید یا نام فضای نام دربردازنده آن را به‌طور کامل قبل از نام کلاس ذکر کنیم یا این که با استفاده از using این فضا را در ابتدای برنامه معرفی کنیم که در این صورت هنگام استفاده فقط ذکر نام کلاس کافی است.

مثال: در کتابخانه NET. کلاسی به نام MessageBox وجود دارد که از آن برای نمایش یک پیغام کوتاه فوری استفاده می‌کنیم. این کلاس در فضای نام Systems.Windows.Forms تعریف شده است. بنابراین یا باید به شکل کامل زیر از این کلاس استفاده کنیم:

```
System.Windows.Forms.MessageBox.Show(“The number is incorrect!”);
```

یا این که در بالای برنامه داشته باشیم:

```
using System.Windows.Forms;
```

که در این صورت کافی است به شکل خلاصه به صورت زیر از کلاس استفاده کنیم:

```
MessageBox.Show(“The number is incorrect!”);
```

در سی‌شارپ علاوه بر اینکه نوع‌داده‌های معمول ++C از قبیل int، float و... قابل استفاده هستند، نوع مستقلی بنام string وجود دارد. مثلا می توانیم به‌صورت زیر یک متغیر رشته‌ای بنام s تعریف کنیم:

```
string s;
```

به‌طور معمول حتی وقتی قرار است از ورودی، عدد دریافت شود، ابتدا ورودی به‌صورت رشته دریافت می‌شود و سپس به عدد تبدیل می‌شود. این کار به کمک متدی بنام Parse صورت می‌گیرد.

مثال:

```
a = int.Parse( str );
```

(رشته str به معادل عددی آن تبدیل و در متغیر عددی a قرار می‌گیرد.)

برعکس در مواردی لازم است یک عدد به رشته تبدیل شود. این کار با استفاده از متدی بنام ToString انجام می‌شود. مثلاً:

```
str = a.ToString();
```

هم چنین در سی‌شارپ برای مقادیر منطقی، نوعی بنام bool وجود دارد و بر خلاف C++ لازم نیست از اعداد برای بیان مقادیر منطقی استفاده کنیم.

به کمک سی‌شارپ برنامه‌های مختلفی می‌توان ایجاد کرد از جمله برنامه‌های کنسولی و برنامه‌های کاربردی ویندوزی (مبتنی بر فرم). در دو قسمت بعدی به این دو نوع پروژه می‌پردازیم.

۲–۲ برنامه‌های کنسولی

خروجی یک برنامه کنسولی در کنسول است که صفحه ساده و سیاه‌رنگی شبیه به صفحه DOS است. برای اهداف خاصی از جمله شروع برنامه‌نویسی با C# نوشتن یک برنامه کنسولی مناسب به نظر می‌رسد.

کلاسی به نام Console شامل متدهای مفیدی برای استفاده در برنامه‌های کنسولی است. از جمله با استفاده از متد ReadLine می‌توانیم از صفحه کلید یک ورودی (رشته‌ای) دریافت کنیم وبه کمک WriteLine ویا Write می‌توانیم چیزی را در خروجی چاپ کنیم.

مثال۱: در این مثال می‌خواهیم طرز کار با متغیرهایی از انواع مختلف، ورودی/خروجی، تبدیل نوع و دستور if را ببینیم. می‌خواهیم قبض مصرفی برق را برای یک مشترک نمایش دهیم. کد جنسیت و نام و میزان مصرف دریافت می‌شود وبر اساس هر کیلووات ساعت ۸۲.۵ ریال قبض صادر می‌شود.

ابتدا قسمت New Project را انتخاب کنید. نوع پروژه را Console Application انتخاب کنید و در همان صفحه نامی هم برای پروژه انتخاب کنید مثلا Electrical. با انتخاب OK، صفحه تولید برنامه به صورت زیر باز می‌شود:

```
using System;
using System.Collections.Generic;
using System.Text;
namespace Electrical
{
        class Program
        {
                static void main (string[] args)
                {
                }
        }
}
```

اکنون داخل متد Main را به صورت زیرکامل کنید:

```
static void Main (string[] args)
 {
        int consume;
```

```
        char code;
        string name , consumeStr, message;
        float pay;
        Console.Write ( "Enter code:" );
        Code = Console.Readkey().keychar;
        Console.WriteLine();
        Console.Write ( "Enter name:" );
        name = Console.ReadLine();
        Console.Write ( "Enter consume:" );
        consumeStr = Console.ReadLine();
        Consume = int.Parse ( consumeStr );
        pay = consume * 82.5f;
        if ( code == 'f' )
                message = "Mr. ";
        else
                message = "Ms. ";
        message = message + name;
        Console.WriteLine ( message + "    " + pay.ToString() );
}
```

برنامه را با Ctrl F5 اجرا کنید.

۲-۳ برنامه‌های ویندوزی مبتنی بر فرم

در برنامه‌های ویندوزی، رابط کاربر به صورت گرافیکی و به صورت یک فرم (پنجره) است. در این فرم امکاناتی وجود دارد که کنترل نامیده می‌شوند. مثلاً دکمه[۱]، برچسب[۲]، کادر متنی[۳]، دکمه رادیویی[۴] و چک‌لیست[۵] نمونه‌هایی ازکنترل هستند.

با ایجاد یک پروژه ویندوزی، یک فرم به‌صورت آماده و خالی ظاهر می‌شود و با استفاده از جعبه ابزار[۶] می‌توانیم کنترل‌های موردنظر را روی آن قرار دهیم. این کارها در پنجره‌ای به نام پنجره Design صورت می‌گیرد.

هم خود فرم و هم هریک از کنترل‌ها «خاصیت»های[۱] مختلفی دارند از قبیل نام، متن، اندازه، موقعیت، رنگ نوشته و... که می‌توانیم با راست‌کلیک‌کردن روی آن، پنجره properties را باز کنیم و این خاصیت‌ها را مشاهده و یا به‌طور دلخواه تنظیم کنیم. بعضی از خاصیت‌های کنترل‌ها هم به‌صورت ویژوال و با ماوس قابل تغییر هستند مثل اندازه و مکان.

---

[1] Button
[2] Label
[3] Textbox
[4] Radiobutton
[5] Checkbox
[6] Toolbox

قسمتی از کارها باید با کدنویسی کامل شود. به روش‌های مختلف می‌توانیم پنجره Code را باز کنیم و کد موردنظر برای موارد مختلف را در آنجا بنویسیم. مثلاً با دابل‌کلیک‌کردن روی یک دکمه، صفحه کد باز می‌شود و می‌توانیم متدی را که به هنگام کلیک کاربر روی این دکمه قرار است اجرا شود، در آنجا بنویسیم.

به‌طور پیش‌فرض، نام فرم و نیز کنترل‌ها به‌صورت شماره‌دار توسط خود محیط درنظر گرفته می‌شود مثلاً Form1، Form2، ... و یا Textbox1، Textbox2،...، label1 و... . بهتر است همان اول از پنجره properties نام مناسبی برای آنها انتخاب کنیم. مخصوصاً توصیه می‌شود که از روش موسوم به نامگذاری مجارستانی برای این منظور استفاده شود. دراین روش، نام هر کنترل شامل یک پیشوند است که نوع کنترل را نشان می‌دهد و ادامه نام، کار کنترل را نشان می‌دهد مثلاً btnOk، lblEnterNumber، rdbAdd، txtTeacherInfo.

۲-۳-۱ ایجاد یک برنامه ساده مبتنی بر فرم

مثال ۲: هدف از این مثال، ایجاد یک برنامه فرمی ساده است که در آن کنترل‌های بسیار رایج، استفاده می‌شود. برنامه قرار است برای یک فروشگاه، فاکتور فروش کامپیوتر صادر کند. قیمت کامپیوتر ۳۰۰۰۰۰، مانیتور CRT ۱۵۰۰۰۰ و مانیتور LCD ۲۰۰۰۰۰ تومان فرض می‌شود.

نام مشتری در یک Textbox دریافت می‌شود. همچنین از طریق دکمه‌های رادیویی، کاربر تعیین می‌کندکه مشتری، مونیتور CRT انتخاب کرده است یا LCD. از طرف دیگر برای مشتریانی که عضو این فروشگاه هستند ۵ درصد تخفیف در نظر گرفته می‌شود. عضویت مشتری در فروشگاه از طریق یک Checkbox تعیین می‌شود.

مراحل کار: برای تولید این برنامه، New Project و سپس پروژه نوع Windows Application را انتخاب کنید و نام پروژه را (مثلاً) Computer Store انتخاب کنید. با بازکردن پنجره properties[1]، خاصیت Text فرم را به "Computer Store" تغییر دهید. با باز کردن Toolbox یک Label به روی فرم منتقل کنید. سپس خاصیت Text آن را به "Enter customer name:" تغییر دهید.

یک Textbox به فرم اضافه کنید. خاصیت Name آن را به txtName تغییر دهید.

دو دکمه رادیویی به داخل فرم منتقل کنید. درمورد اولی خاصیت Name آن را به rdbCRT و خاصیت Text آن را به "CRT" و نیز خاصیت Checked آن را به True تغییر دهید. درمورد دومی، خاصیت Name آن را به rdbLCD و خاصیت Text آن را به "LCD" تغییر دهید.

یک CheckBox به فرم اضافه کنید. خاصیت Name آن را به chkMember و خاصیت Text آن را به "Member of Store" تغییر دهید.

یک TextBox برای نمایش فاکتور اضافه کنید و خاصیت Name آن را به txtShow تغییر دهید. همچنین در صورت تمایل، قبل از این Textbox، یک label اضافه کنید و خاصیت Text آن را به "Total Price:" تغییر دهید.

---

[1] properties

دو دکمه ایجاد کنید. درمورد اولی، خاصیت Name آن را btnShow و خاصیت Text آن را "Show Factor" قرار دهید. برای دکمه دوم خاصیت Name آن را به btnOk و خاصیت Text آن را به "OK" تغییر دهید.

اکنون طراحی رابط کاربر کامل شده است و باید با کدنویسی مشخص کنیم که برنامه چه کاری باید انجام دهد. روی دکمه OK دوبار کلیک کنید. پنجره کد باز می‌شود و آماده است تا کد متد مربوط به کلیک کاربر روی این دکمه را بنویسید. آن را به‌صورت زیر کامل کنید:

```
private void btnOK_Click(object sender, EventArgs e)
{
        this.Close();
}
```

سپس دوباره به پنجره design برگردید و این بار روی دکمه Show Factor دوبارکلیک کنید و متد مربوط به این دکمه را بصورت زیر کامل کنید:

```
private void btnShow_Click(object sender, EventArgs e)
{
        if (txtName.Text == "")
            MessageBox.Show("Please enter name");
        else
        {
           float price = 300000;
           if (rdbCRT.Checked)
               price += 150000;
           else
               price += 200000;
           if (chkMember.Checked)
               price = price - 0.05f * price;
           txtShow.Text = price.ToString();
        }
}
```

برنامه را با Ctrl F5 اجرا کنید.

۲-۳-۲ رویداد - استثنا

درحین اجرای برنامه برای هر کنترل چندین رویداد[1] مختلف ممکن است رخ دهد که می‌توانیم در برنامه مشخص می‌کنیم که در مورد هریک از آنها کار خاصی صورت گیرد. مثلاً برای یک دکمه، رویدادهای متنوعی وجود دارد مانند کلیک روی دکمه،

[1] Event

بردن ماوس روی دکمه، خارج‌کردن ماوس از روی دکمه، زدن Enter روی دکمه و .... برای یک دکمه رادیویی برخی از رویدادها عبارتند از تغییر وضعیت تیک آن، Enter روی آن و ...

برای برنامه‌ریزی متد مربوط به کلیک روی یک دکمه کافی است که در پنجره Design روی آن دابل‌کلیک کنیم تا متد مربوط نظر برای تکمیل باز شود. برای برنامه‌ریزی رویدادهای دیگر ابتدا باید کنترل مورد نظر را انتخاب کنیم، سپس از قسمت بالایی پنجره Properties، آیکون Events را انتخاب کنیم تا لیست کل رویدادهای ممکن برای آن کنترل، نمایش داده شود. سپس می‌توانیم روی هر رویداد دابل‌کلیک کنیم تا متد مربوط به آن رویداد برای تکمیل باز شود.

مفهوم دیگری که در این بخش به آن خواهیم پرداخت، استثنا[1] است. یک استثنا خطایی است که در زمان اجرای برنامه پیش می‌آید. استثناها بسیار متنوع هستند. مثلاً ممکن است به جای یک متغیر صحیح، کاراکترهای نامعتبر وارد شود (FormatException) یا این که سعی شود تقسیم بر صفر صورت گیرد (DivideByZeroException).

بهتر است برنامه را طوری بنویسیم که در صورت بروز یک استثنا برخورد مناسبی صورت گیرد، مثلا یک پیام خطا صادر شود یا کار مناسب دیگری انجام شود. به این مدیریت و برخورد مناسب اصطلاحا Exception Handling می‌گویند. یک استثنای مدیریت‌نشده، اغلب به قطع برنامه منجر می‌شود که بسیار نامناسب است. برای مدیریت استثنا می‌توانیم بخشی از برنامه را که ممکن است در آن استثنا بروز کند، داخل یک بلاک try قرار دهیم و بعد از این بلاک، یک یا چند بلاک catch قرار دهیم که درصورتی‌که هریک از استثناهای مختلف پیش آمده باشد، دستورات مشخص‌شده در بلاک catch مربوط به آن اجرا شود. جزئیات بیشتر در این مورد را در قالب مثال زیر خواهیم دید.

مثال ۳: هدف از این مثال، تمرین برنامه‌نویسی رویدادها و نیز مدیریت استثنا است. برنامه‌ای تولید خواهیم کرد که به نوعی شبیه یک ماشین‌حساب عمل می‌کند. کاربر دوعدد را در دو TextBox وارد می‌کند. همچنین از یک مجموعه دکمه‌های رادیویی، یکی از گزینه‌های جمع، تفریق، ضرب یا تقسیم را انتخاب می‌کند. برنامه باید عمل خواسته‌شده را روی دو عدد داده شده انجام دهد و نتیجه را در یک TextBox نمایش دهد. می‌خواهیم پس از اعلام نتیجه، چنانچه کاربر، عمل مورد نظر را تغییر دهد، مجدداً (بدون نیاز به کلیک روی دکمه محاسبه)، محاسبه براساس عملگر جدید صورت گیرد. همچنین می‌خواهیم حداقل دو خطای تقسیم بر صفر و فرمت، مدیریت شود و در صورت بروز آنها یک پیغام مناسب نمایش داده شود.

مراحل کار: یک پروژه از نوع Windows Application ایجاد کنید و نام Calculator را برای آن انتخاب کنید.

خاصیت Text فرم ایجادشده را به «ماشین حساب» تغییر دهید و با ماوس، اندازه فرم را در هر دو راستا قدری بزرگ کنید.

به ترتیب کنترل‌هایی به صورت زیر به فرم کلید اضافه کنید:

یک label برای پیغام واردکردن عدد اول. خاصیت Text آن را به Enter first number تغییر دهید.

یک TextBox برای دریافت عدد اول. خاصیت Name آن را به txtFirst تغییر دهید.

یک label برای پیغام واردکردن عدد دوم. خاصیت Text آن را به Enter second number تغییر دهید.

[1] Exception

یک TextBox برای دریافت عدد دوم. خاصیت Name آن را txtSecond تغییر دهید.

چهار دکمه رادیویی به ترتیب با نام های rdbAdd، rdbSub، rdbMult و rdbDivide. خاصیت Text آنها را به ترتیب به «جمع»، «تفریق»، «ضرب» و «تقسیم» تغییر دهید.

یک Label برای توضیح نتیجه. خاصیت Text آن را به «نتیجه:» تغییر دهید.

یک Textbox برای نمایش نتیجه. خاصیت Name آن را به txtResult تغییر دهید.

دو دکمه یکی برای انجام محاسبه و دیگری برای پایان‌دادن به برنامه به ترتیب با نام‌های btnCalculate وbtnClose. خاصیت Text آنهارا به ترتیب برابر با «محاسبه» و «خروج» قرار دهید.

اکنون طراحی فرم کامل شده است.

پنجره code را بازکنید. پس از متدی که به صورت زیر تعریف شده است:

```
public Form1()
{
      InitializeComponent();
}
```

متدی به صورت زیر برای انجام محاسبه اضافه کنید:

```
private void calculate()
{
       int a, b, c = 0;
       try
       {
           a = int.Parse(txtFirst.Text);
           b = int.Parse(txtSecond.Text);
           if (rdbAdd.Checked)
               c = a + b;
           if (rdbSub.Checked)
               c = a - b;
           if (rdbMult.Checked)
               c = a * b;
           if (rdbDivide.Checked)
               c = a / b;
           txtResult.Text = c.ToString();
       }
       catch (DivideByZeroException)
       {
           MessageBox.Show("b cannot be 0");
       }
       catch (FormatException)
       {
           MessageBox.Show("Invalid characters");
       }
}
```

به پنجره Design برگردید. با دابل‌کلیک روی دکمه «محاسبه» کد متد مربوط به کلیک روی این دکمه را به صورت زیر کامل کنید (کافی است که فقط فراخوانی متد calculate را داخل متد بنویسید).

```
private void btnCalculate_Click(object sender, EventArgs e)
{
     calculate();
}
```

دوباره به پنجره Design برگردید. روی دکمه رادیویی «جمع» کلیک کنید. در پنجره Properties آیکون Events را انتخاب کنید و از لیست رویدادها CheckedChanged را انتخاب و روی آن دابل‌کلیک کنید. متد مربوط به تغییر این دکمه رادیویی برای تکمیل باز می‌شود. دقیقاً مشابه با مورد قبل، فقط فراخوانی متد calculate را داخل آن قرار دهید:

```
private void rdbAdd_CheckedChanged(object sender, EventArgs e)
{

       calculate();
}
```

کار مشابهی برای سه دکمه رادیویی دیگر هم انجام دهید.

با دابل‌کلیک روی دکمه Close متد مربوط به آن را بصورت زیر کامل کنید:

```
private void btnClose_Click(object sender, EventArgs e)
{
    this.Close();
}
```

برنامه را با Ctrl F5 اجرا کنید.

بخش سوم: برنامه‌نویسی شیءگرا

۳-۱ کلاس، شیء، اعضای اختصاصی و عمومی، سازنده

یک سیستم از موجودیت‌های مختلفی تشکیل شده است که هریک «ویژگی‌ها» و «رفتارهایی» دارند. مثلاً در سیستم انتخاب واحد دانشگاه، مهم‌ترین موجودیت‌ها عبارتند از: دانشجویان، درس‌های ارائه‌شده، اساتید و مدیر آموزش. وقتی بخواهیم با استفاده از روش شیءگرا یک سیستم را به صورت یک برنامه کامپیوتری پیاده‌سازی کنیم، هریک از موجودیت‌های آن سیستم را به‌عنوان یک **شیء**[1] در برنامه تعریف می‌کنیم و ویژگی‌ها و رفتارهای آن شیء و ارتباط آن با دیگر اشیا را پیاده‌سازی می‌کنیم. دقت کنید که در اینجا هر موجودیتی حتی اگر جاندار باشد، یک شیء محسوب می‌شود. بنابراین مثلاً در یک برنامه شیءگرا که برای پیاده‌سازی کامپیوتری انتخاب واحد طراحی می‌شود، هر یک از دانشجویان، درس‌ها، اساتید و مدیر آموزشی به صورت یک شیء تعریف می‌شوند. مفهوم کلی که همه اشیاء از یک نوع را نشان می‌دهد، یک **کلاس** نامیده می‌شود. مثلاً «دانشجو» یک کلاس است و هریک از دانشجویان که نمونه[2]هایی از کلاس دانشجو هستند، یک شیء از این کلاس محسوب می‌شوند. همچنین «درس» یک کلاس است و «مبانی کامپیوتر»، «برنامه‌سازی پیشرفته» و ... هر کدام یک شیء از این کلاس هستند.

درمورد هر کلاس، دو مفهوم باید پیاده‌سازی شود:

۱- ویژگی‌های هر شیء از کلاس که **خاصیت**[3] نامیده می‌شوند. مثلاً ویژگی‌های هر دانشجو عبارتند از نام، شماره دانشجویی، تعداد واحدهای گذرانده و...

[1] object
[2] instance
[3] property

۲- عملیات مربوط به هر شیء از کلاس که در قالب **متد[1]ها** (توابع) پیاده‌سازی می‌شوند. مثلاً در سیستم انتخاب واحد، یک دانشجو ممکن است درس خاصی را انتخاب کند، مرخصی تحصیلی بگیرد یا سیستم برای او بررسی کند که آیا پیش‌نیازی را رعایت کرده است و...

خاصیت‌ها و متدهای مربوط به یک کلاس، **اعضای**[2] کلاس نامیده می‌شوند.

دریک برنامه شیءگرا معمولاً کلاس‌های زیادی تعریف شده است که باهم ارتباط دارند. اعضای یک کلاس را می‌توان طوری تعریف نمود که فقط متدهای عضو همان کلاس به آن دسترسی داشته باشند (private) و یا این که متدهای کلاس‌های دیگر هم به آن دسترسی داشته باشند (public). توصیه می‌شود که تا حد امکان اعضای یک کلاس به ویژه خاصیت‌ها (فیلدها) به‌صورت private تعریف شوند و حتی اگر کلاس‌های دیگر قرار است به این مقادیر دسترسی پیدا کنند، این کار را به کمک متدهای کلاس مورد نظر انجام دهند نه به‌طور مستقیم. به این اصل **کپسوله‌سازی**[3] می‌گویند. دلایل مختلفی برای این کار وجود دارد که دو مورد از آنها در دو مثال زیر آمده است:

۱–فیلد معدل مربوط به یک دانشجو نمی‌تواند توسط یک کلاس دیگر مقداردهی شود چراکه اصلاً معنی ندارد که معدل یک دانشجو بدون محاسبه برابر با مقدار خاصی قرار گیرد. معدل یک دانشجو از روی نمرات دانشجو به شکل خاصی تعیین می‌شود که مسلماً این کار جزو وظایف خودِ کلاس دانشجو است. اما می‌توانیم فیلد معدل دانشجو را بصورت «فقط خواندنی» در اختیار کلاس‌های دیگر قرار دهیم. مثلاً کلاس دانشجو یک فیلد اختصاصی بنام Average و یک متد عمومی به نام ReportAverage داشته باشد. هر کلاس دیگر اگر می‌خواهد معدل دانشجو را بداند، متد ReportAverage کلاس دانشجو را فراخوانی می‌کند و این متد، مقدار معدل را به آن تحویل می‌دهد.

۲– فرض کنید که یک کلاس دیگر می‌خواهد به فیلد تعداد واحدهای دانشجو (unit) مقداردهی کند. حتی اگر این اجازه داده شود، کلاس دانشجو باید خود، مسئول درستی مقدار پیشنهادی باشد. بنابراین بهتر است بازهم فیلد unit اختصاصی تعریف شود اما یک متد عمومی بنام SetUnit درنظر گرفته شود که هرگاه یک کلاس دیگر این متد را فراخوانی کرد و مقداری به‌عنوان «تعداد واحد» به‌صورت پارامتر به آن تحویل داد، آن را در فیلد unit قرار دهد. نکته مهم این است که متد SetUnit می‌تواند طوری نوشته شود که مقدار پارامتر را بررسی کند وفقط درصورتی‌که معتبر باشد (مثلاً بین ۱۲ تا ۲۰ باشد) آن را در فیلد unit قراردهد وگرنه خطا بگیرد.

دریک برنامه شیءگرا ابتدا یک کلاس تعریف می‌شود. سپس هر زمان که لازم باشد، از آن کلاس(نوع)، شیء جدیدی ایجاد (new) می‌شود.

اغلب لازم است که به محض ایجاد یک شیء، یک سری کارهای اولیه فورا برای آن انجام شود. مثلاً در سیستم انتخاب واحد به محض ایجاد یک شیء از نوع «دانشجو» باید شماره دانشجویی، نام و... برای او مشخص شود. معمولاً برای این منظور یک متد

[1] method
[2] members
[3] encapsulation

نوشته می‌شود. چنین متدی **سازنده** (constructor) نامیده می‌شود. نام متد سازنده، هم‌نام کلاس است و برای آن هیچ نوع مقدار برگشتی تعریف نمی‌شود (حتی void).

درمثال زیر مفهوم کلاس، شیء، اعضای اختصاصی و عمومی و متد سازنده را تمرین خواهیم کرد.

مثال ۴– دراین مثال، کلاسی بنام Student برای ارائه مفهوم «دانشجو» تعریف می‌کنیم. این کلاس دارای پنج خاصیت خواهد بود که عبارتند از نام دانشجو، حداقل واحد مجاز، حداکثر واحد مجاز، تعداد واحد انتخابی و معدل (معدل ترم قبل که در انتخاب واحد، مبنای حداکثر تعداد واحد مجاز است). برای تمرین و بررسی تفاوت خاصیت‌های خصوصی و عمومی، خاصیت «نام» را public و بقیه را private تعریف می‌کنیم (در اصل، در این برنامه، همه خاصیت‌ها بهتر است خصوصی تعریف شوند).

همچنین برای این کلاس، چند متد تعریف می‌کنیم: یکی متد سازنده که نام ومعدل دانشجو پارامتر آن است و این دو را در فیلدهای مربوط قرار می‌دهد و همچنین براساس معدل، حداکثر تعداد واحد مجاز را هم تنظیم می‌کند و حداقل واحد مجاز را هم برابر با ۱۲ قرار می‌دهد. دو متد دیگر هم یکی برای گزارش‌کردن معدل و دیگری برای مقداردهی به فیلد تعداد واحد انتخابی خواهیم نوشت.

مراحل کار: یک پروژه جدید بنام University ایجاد کنید. در فرم ایجاد شده سه TextBox ایجاد کنید و خاصیت نام آنها را به ترتیب txtName ، txtAverage و txtUnit قرار دهید. همچنین می‌توانید در کنار هریک از این سه، یک Label قرار دهید که فیلد Text آنها را به ترتیب به صورت "Student name is:"، "Student average is:" و "Please enter the units:" باشد.

همچنین یک دکمه ایجاد و خاصیت Name آن را به btnTestUnit و خاصیت Text آن را به Test unit تغییر دهید.

برای ایجاد کلاس Student، دکمه Add New Item را روی نوار ابزار پیدا و کلیک کنید (یا کلیدهای Ctrl Shift A را بزنید). از این طریق، کلاسی به‌نام Student ایجاد کنید. در پنجره‌ای که باز می‌شود، باید تعریف کلاس را به صورت زیر کامل کنید:

```
class Student
{
    public string name;
    private int maxUnit, minUnit, unit;
    private double average;
    public Student (string nam, double avg)
    {
           name=nam;
           average=avg;
           if (average >= 17)
               maxUnit=24;
           else
               maxUnit=20;
           minUnit=12;
     }
     public double ReportAverage()
     {
           return average;
      }
      public void SetUnit(int u)
      {
          if (u < minUnit || u > maxUnit)
              System.Windows.Forms.MessageBox.Show("Number of units is wrong");
```

```
            else
            {
                unit = u;
                System.Windows.Forms.MessageBox.Show("Number of units is OK");
            }
        }
    }
```

اکنون به پنجره طراحی فرم برگردید و روی خود فرم دابل‌کلیک کنید. صفحه مربوط به متد Form1_Load باز می‌شود که این متد به محض بازشدن فرم اجرا خواهد شد. ابتدا چند خط بالاتر بروید و در ابتدای کلاس Form1 (قبل از public Form1()، خط زیر را اضافه کنید:

```
Student s;
```

سپس متد Form1_Load را به‌صورت زیر کامل کنید:

```
private void Form1_Load(object sender, EventArgs e)
{
     s = new Student("Amini", 14.5);
     txtName.Text = s.name;
     txtAverage.Text = s.ReportAverage().ToString();
}
```

بعد متد مربوط به کلیک روی دکمه Test Unit را با دابل‌کلیک روی این دکمه باز و به صورت زیر کامل کنید:

```
private void btnTestUnit_Click(object sender, EventArgs e)
{
      s.SetUnit(int.Parse(txtUnit.Text));
}
```

۳-۲ ارث‌بری و Overriding

دیدیم که یک کلاس یک نوع را توصیف می‌کند. گاهی از یک کلاس، کلاس(های) دیگری مشتق می‌شود. مثلاً اگر برای مفهوم «کارمند» یک کلاس تعریف کنیم، می‌توانیم از آن، کلاس دیگری به نام «مدیر» مشتق کنیم که بسیاری از ویژگی‌های آن، مانند کلاس «کارمند» است (به این حالت ارث‌بری[1] گفته می‌شود). اما کلاس مدیر برخی ویژگی‌های خاص خود را هم دارد (مثلاً حقوق مبنای مدیر فرق می‌کند و...). به کلاس «کارمند» دراین مثال، کلاس پایه[2] و به کلاس «مدیر» کلاس مشتق‌شده[3] یا زیرکلاس[4] می‌گویند.

مثال ۵- دراین مثال، کلاسی بنام Employee تعریف می‌کنیم که مفهوم «کارمند» را توصیف می‌کند. خاصیت‌های مربوط به کارمند عبارتند از نام کارمند(name) و حقوق پایه او (baseSalary). میزان حقوق پایه یک کارمند عادی دو میلیون تومان در نظر گرفته می‌شود. علاوه بر متد سازنده، یک متد دیگر هم برای این کلاس تعریف می‌کنیم که GiveSalary نام دارد و میزان حقوق کارمند را برمی‌گرداند.

سپس یک زیرکلاس بنام Manager از کلاس پایه Employee مشتق می‌کنیم که کارمند از نوع «مدیر» را توصیف می‌کند.

---

[1] inheritance
[2] Base class
[3] Derived class
[4] Subclass

کلاس Manager فیلدهای نام و حقوق پایه را از کلاس Employee به ارث می‌برد اما علاوه بر آنها یک فیلد دیگر هم دارد که تعداد کارمندان تحت مدیریت او را نشان می‌دهد(numberOfEmployees).

حقوق پایه یک مدیر برخلاف کارمندان عادی سه میلیون تومان است و علاوه بر این به ازای هر کارمند تحت مدیریتش ده هزار تومان به حقوقش افزوده می‌شود. بنابراین متد GiveSalary برای این کلاس باید متفاوت با کلاس کارمند باشد.

مراحل کار: یک پروژه بنام Office (اداره) ایجاد کنید. روی فرم دو TextBox ایجاد کنید و خاصیت نام آنها را به txtEmpSalary و txtManagerSalary تغییر دهید. همچنین می‌توانید درکنار هر کدام از TextBoxها یک Label برای توضیح، اضافه کنید که فیلد Text آنها به ترتیب "Employee's salary" و "Manager's salary:" باشد.

از طریق Add New Item یک کلاس به نام Employee ایجاد کنید و تعریف آن را به صورت زیر کامل کنید.

```
class Employee
{
     protected string name;
     protected long baseSalary;
     public Employee (string n)
     {
         baseSalary = 2000000;
         name = n;
     }
     public virtual long GiveSalary()
     {
         return baseSalary;
     }
}
```

سپس کلاس دیگری به نام Manager ایجاد کنید و خط اول تعریف آن را به صورت زیر تغییر دهید تا مشخص شود که زیرکلاسی از Employee است:

```
class Manager: Employee
```

تعریف کلاس Manager را به صورت زیر کامل کنید (دقت کنید که فقط مواردی نوشته می‌شود که نسبت به کلاس پایه یعنی Employee اضافه شده یا تغییر کرده است).

```
class Manager: Employee
{
    int numberOfEmployees;
    public Manager( string n, int number ): base( n )
    {
        baseSalary = 3000000;
        numberOfEmployees = number;
    }
    public override long GiveSalary()
    {
        return baseSalary + numberOfEmployees * 10000;
    }
}
```

اکنون به صفحه Design برگردید و روی فرم دابل‌کلیک کنید و بعد متد مربوط به بازشدن فرم را به صورت زیر تکمیل کنید:

```
private void Form1_Load(object sender, EventArgs e)
{
       Employee emp = new Employee("Razavi");
       txtEmpSalary.Text = emp.GiveSalary().ToString();
```

```
        Manager man = new Manager("Saeedi", 20);
        txtManagerSalary.Text = man.GiveSalary().ToString();
}
```

۳-۳ سربارگذاری متدها

می‌توانیم نسخه‌های مختلف از یک متد بنویسیم که تعداد و/یا نوع پارامترهای آنها با هم فرق داشته باشد و طرز کار آنها هم اگر چه شبیه به هم ولی تا حدی متفاوت باشد . به این کار **سربارگذاری**[1] یا **بارگذاری اضافی** متد می‌گویند.

به عنوان یک نمونه فرض کنید در یک لیست، اطلاعات دانشجویان نگهداری می‌شود که شامل شماره دانشجویی، نام، نام خانوادگی و... است و بر حسب شماره دانشجویی مرتب است. ممکن است کاربر، شماره دانشجویی یک کاربر را بدهد و از برنامه بخواهد که اطلاعات کامل این دانشجو را استخراج کند. از طرفی ممکن است کاربر شماره دانشجویی را نداند و به جای آن نام و نام خانوادگی دانشجو را بدهد. می‌توانیم دو نسخه از متدی به نام Search بنویسیم: یکی یک پارامتر از نوع long (شماره دانشجویی) دریافت کند و با جستجوی دودویی، دانشجوی با این شماره دانشجویی را پیدا کند. دیگری دو پارامتر از نوع string (نام و نام خانوادگی) دریافت کند و با جستجوی خطی، دانشجوی دارای این نام و نام خانوادگی را پیدا کند. با این که نام هر دو متد Search است، در صورتی که متد با یک پارامتر عددی فراخوانی شود، نسخه اول آن و در صورتی که با دو پارامتر رشته‌ای فراخوانی شود، نسخه دوم آن فعال خواهد شد.

مثال ۵ نمونه دیگری از سربارگذاری متدهارا نشان می دهد.

مثال ۶- موضوع این مثال هم اداره و کارمند است اما با مثال قبل فرق‌هایی دارد. یک کلاس به نام Employee تعریف می‌کنیم که مانند مثال قبل دو فیلد name و bseaSarlary دارد. مقدار baseSalary برای هرکارمند باز هم ۲۰۰۰۰۰۰ تومان فرض می‌شود. متدی به نام GiveSalary تعریف می‌کنیم که بدون پارامتر است و در صورت فراخوانی، مقدار حقوق پایه کارمند را برمی‌گرداند. یک نسخه دیگر از متد GiveSalary تعریف می‌کنیم که یک پارامتر از نوع صحیح می‌گیرد که تعداد ساعت‌های اضافه‌کاری کارمند است. این متد مبلغ مربوط به اضافه‌کاری کارمند را از قرار هر ساعت ۵۰۰۰ تومان با حقوق پایه جمع می‌کند و حقوق کامل کارمند را برمی‌گرداند. باز هم نسخه سومی از متد GiveSalary تعریف می‌کنیم که دو پارامتر دارد: یکی از نوع عدد صحیح که تعداد ساعت‌های اضافه‌کاری کارمند را نشان می‌دهد و دیگری از نوع double که نرخ مالیات را نشان می‌دهد. متد اخیر جمع کل حقوق پایه و اضافه کاری را با کسر مالیات نشان می‌دهد.

اکنون برحسب این که متد GiveSalary با هیچ یا یک یا دو پارامتر فراخوانی شود، به ترتیب نسخه اول، دوم یا سوم متد اجرا خواهد شد. مثلاً GiveSalary(); مقدار ۲۰۰۰۰۰۰، GiveSalary(20); مقدار ۲۱۰۰۰۰۰ و GiveSalary(20, 0.05); مقدار ۱۹۹۵۰۰۰ را برخواهد گرداند.

مراحل کار: برای تولید این برنامه، یک پروژه به نام Office2 ایجادکنید. روی فرم برنامه سه TextBox ایجاد کنید و خاصیت نام آن را به ترتیب به txtMainSalary، txtMainOvertimeSalary و txtSalaryTax تغییر دهید. همچنین در صورت تمایل در کنار

[1] Overloading

هریک از این سه، یک label ایجاد کنید که متن آنها به ترتیب بصورت "Main salary:" و "Main salary + overtime:" و "Salary after tax:" باشد. یک کلاس به نام Employee ایجاد کنید و تعریف آن را به صورت زیر کامل کنید:

```
class Employee
{
    string name ;
    double baseSalary;
    public Employee( string n )
    {
        name = n;
        baseSalary = 2000000;
    }
    public double GiveSalary()
    {
        return baseSalary;
    }
    public double GiveSalary ( int hours )
    {
        return baseSalary + hours * 5000;
    }
    public double GiveSalary( int hours, double tax)
    {
        double s = baseSalary + hours * 5000;
        return s - tax * s;
    }
}
```

به پنجره Design برگردید و روی فرم دابل‌کلیک کنید تا متد Form1_Load برای تکمیل باز شود. قبل از تکمیل این متد در ابتدای کلاس Form1 (قبل از public Form1()، خط زیر را اضافه کنید:

```
Employee emp;
```

اکنون متد Form1_load را به صورت زیر کامل کنید:

```
private void Form1_Load(object sender, EventArgs e)
{
      emp = new Employee("Habibi");
      txtMainSalary.Text = emp.GiveSalary().ToString();
      txtMainOvertimeSalary.Text = emp.GiveSalary(20).ToString();
      txtSalaryTax.Text = emp.GiveSalary(20, 0.05).ToString();
}
```

امتحان میان‌ترم برنامه سازی پیشرفته – فروردین ۹۴ – مدرس: حسن شاکری – وقت: ۷۵ دقیقه

۱) برنامه‌ای بنویسید که با دریافت n و r و با استفاده از یک تابع برای محاسبه فاکتوریل، مقدار رابطه زیر را محاسبه و اعلام کند.

$$C_{n,r} = \frac{n!}{r!\,(n-r)!}$$

۲) برنامه‌ای بنویسید که عناصر یک آرایه دوبعدی ۱۰×۸ را دریافت کند و میانگین عناصر هر سطر آن را تعیین کند.

۳) برنامه‌ای بنویسید که اطلاعات بیست دانشجو شامل نام، شماره دانشجویی و معدل را به صورت ساختار دریافت کند و اطلاعات کامل مربوط به دانشجوی دارای بیشترین معدل را تعیین کند.

۴) خروجی برنامه زیر را با توضیح مختصر تعیین کنید:

```
void main()
{
    char S[20];
    int k;
    strcpy( S, "Good morning");
    k = strlen( S );
    cout << k <<"\n";
    S[0] = tolower( S[0] );
    cout << S[0] << "\n";
    if( !strcmp( S, "Hello" )
       cout << "message 1";
    else
       cout << "message 2";
}
```

موفق باشید

امتحان میان‌ترم برنامه سازی پیشرفته – ۹۵/۲/۸ – مدرس: حسن شاکری – وقت: ۷۵ دقیقه

۱) برنامه‌ای بنویسید که یک تاریخ شامل روز و ماه را دریافت کند و به کمک یک تابع بررسی‌کننده مشخص کند که آیا این تاریخ معتبر است یا نه. به عنوان نمونه تاریخ‌های ۴/۳۱/ و ۱۰/۶/ معتبر هستند ولی تاریخ‌های ۷/۳۵/ و ۱۴/۲۱/ و ۳/۰/ و ۸/۳۱/ نامعتبر هستند.

۲) برنامه‌ای بنویسید که یک رشته را دریافت کند و تعداد کلمات آن و نیز تعداد اعداد موجود در آن را تعیین کند. مثلا رشته "Sum of 725 and 63 is 788." دارای هفت کلمه و سه عدد است.

۳) برنامه‌ای بنویسید که یک چندجمله‌ای را به صورت آرایه‌ای از ساختارهای ضریب و توان دریافت کند و سپس مقدار این چندجمله‌ای را به ازای $x=2$ تعیین کند. مثلا چندجمله‌ای $3x^5 - 2x^2 + 10x + 3$ به صورت زیر نمایش داده می‌شود:

3	-2	10	3
5	2	1	0
۰	۱	۲	۳

4
n

و حاصل آن به ازای $x=2$ برابر است با 111.

۴) خروجی برنامه زیر را با توضیح مختصر تعیین کنید:

```
void test( int N, int &M )
{
        N = N + 3;
        M = N + 3;
}
void main()
{
      int x=10, y=10;
      char S[31];
      char *p;

      p = new int [100];
      for( int  i = 0;  i < 100; i++)
                p[i] = 5 * i;
     cout << *(p + 3) << "\n";

      test( x, y );
      cout << x << "     " << y << "\n";

      strcpy( S, "Good morning";
      if( !strcmp( S, "Hello" ))    cout << "Message 1";     else    cout << "Message 2";
}
```

موفق باشید

امتحان پایان‌ترم برنامه‌سازی پیشرفته- ۸ خرداد ۹۲ – مدرس: حسن شاکری – وقت: دو ساعت

۱) برنامه‌ای بنویسید که دو عدد را دریافت کند و همه مضرب‌های پنج بین این دو عدد را به ترتیب نزولی چاپ کند. برای نمایش مضرب‌ها از یک تابع استفاده کنید.

۲) برنامه‌ای بنویسید که هر بار نام یک کاربر را دریافت کند تا این که '***' وارد شود. برنامه باید حرف اول هر نام را به حالت بزرگ و بقیه را به حالت کوچک تبدیل کند و نمایش دهد.

۳) یک تابع بازگشتی بنویسید که تعداد رقم‌های یک عدد را تعیین کند.

۴) مفهوم، کاربرد و مزایای لیست پیوندی را در سه تا چهار سطر توضیح دهید. همچنین نحوه اضافه کردن یک عنصر به وسط لیست پیوندی را با شکل نمایش دهید.

۵) یک کلاس برای ارائه مفهوم دانشجو بنویسید که دارای دو خاصیت شماره دانشجویی و معدل باشد. متد سازنده را طوری بنویسید که شماره دانشجویی و معدل را به عنوان پارامتر دریافت و در این خاصیت‌ها قرار دهد. یک متد دیگر بنویسید که حداکثر واحد مجاز را برحسب معدل برگرداند (معدل زیر ۱۲: ۱۴ واحد، معدل ۱۲ تا کمتر از ۱۷: ۲۰ واحد، معدل ۱۷ یا بیشتر: ۲۴ واحد).
یک زیرکلاس از کلاس فوق با عنوان دانشجوی ترم آخر تعریف کنید که نیازی به سازنده مجزا ندارد ولی متد اعلام حداکثر واحد مجاز باید در این زیرکلاس بازنویسی (override) شود به طوری که بدون توجه به معدل، عدد ۲۴ برگردانده شود.

۶) با توضیح مختصر خروجی برنامه زیر را تعیین کنید:

```
void  f( int a, int *b)
{
      a++;
      b=*a;
      cout << a << *b;
}
void main()
{
      int m=10, n=10;
      f ( m, &n);
      cout << m << n ;
}
```

موفق باشید

امتحان پایان‌ترم برنامه سازی پیشرفته – خرداد ۹۵ – مدرس: حسن شاکری – وقت: دو ساعت

۱- یک لیست تک‌پیوندی را در نظر بگیرید که هر عنصر آن دارای دو فیلد data (داده) و next (اشاره‌گر به گره بعد) است. یک تابع برای نمایش مقادیر عناصر لیست و یک تابع برای اضافه‌کردن عنصر جدیدی در انتهای لیست بنویسید.

۲- یک تابع بازگشتی بنویسید که مجموع رقم‌های یک عدد صحیح مثبت را برگرداند.

۳- اگر تابع بازگشتی زیر با پارامتر 2 فراخوانی شود، چه خروجی تولید خواهد شد؟ (به اختصار دلیل خود را ذکر کنید)

```
void  P ( int a )
{
        if( a>20 )
                return;
        P ( a+3 );
        cout << a << "   ";
}
```

۴- یک کلاس برای پیاده‌سازی مفهوم «کارمند» بنویسید. فرض کنید که برای هر کارمند اطلاعاتی شامل نام و حقوق پایه در نظر گرفته می‌شود. مقدار حقوق پایه برای هر کارمند 300000 فرض می‌شود. همچنین علاوه بر متد سازنده‌ی کلاس کارمند، یک متد برای گزارش‌کردن مقدار حقوق کارمند لازم است.

سپس یک زیرکلاس از کلاس کارمند به نام «مدیر» بنویسید. این کلاس علاوه بر خاصیت‌ها و متدهای کلاس کارمند دارای یک خاصیت (داده) اضافی است که تعداد کارکنان تحت مدیریت مدیر را نشان می‌دهد. از طرف دیگر حقوق پایه مدیر برخلاف کارمندان عادی 400000 است و ضمناً به ازای هر کارمند تحت مدیریت مدیر مبلغ 5000 به حقوق او افزوده می‌شود.

موفق باشید