molnar1956@yahoo.com

Afişarea unui text folosind subprograme.

```
#include <iostream>
using namespace std ;
void afiseazaText ()
{
        for (int i = 0; i < 5; i++)
        cout << "Subprograme" <<endl ;
}

int main ()
{
        afiseazaText ();
        return 0;
}
```

molnar1956@yahoo.com

Adunarea numerelor folosind subprograme.

```
#include <iostream>
using namespace std ;
int suma (int , int );
int main ()
{
int x, y;
       cout << "Dati doua numere intregi:" <<endl ;
       cout << "x= " ; cin >> x;
       cout << "y= " ; cin >> y;
       cout << "Suma lor este " << suma (x, y);
return 0;
}

int suma (int a, int b)
{
       int rezultat = a + b;
       return rezultat ;
}
```

molnar1956@yahoo.com

Aria cercului cu diferite aproximări.

```
#include <iostream>
using namespace std ;

double arieCerc (double raza , double PI = 3.14 )
    {
     return 2 * PI * raza * raza ;
     }

int main ()
{
cout << "Aria cercului de raza 2 este " << arieCerc (2.0 ) << '\n' ; // PI == 3.14
cout << "Aria cercului de raza 2 este " << arieCerc (2.0 , 3.141592 );
return 0;
}
```

molnar1956@yahoo.com

Supraîncărcarea (Overloading) funcţiilor

Supraîncărcaţi o funcţie atunci când definiţi mai multe versiuni ale aceleiaşi funcţii. Funcţia trebuie să aibă acelaşi nume, dar lista de parametri trebuie să difere prin numărul parametrilor sau prin tipul parametrilor (sau ambele). Tipul funcţiei poate fi diferit, dar nu este necesar să fie aşa. Versiunile diferă unele de altele numai prin lista de parametri!

molnar1956@yahoo.com

Aria pătratului şi dretunghiului.

```
#include <iostream>
using namespace std ;

// Aria patratului
double arie (double lungime )
{
return lungime * lungime ;
}

// Aria dreptunghiului
double arie (double lungime , double latime )
{
return lungime * latime ;
}

int main ()
{
cout << "Aria patratului de L = 2 este " << arie (2.0 ) << '\n' ;
cout << "Aria dreptunghiului de L = 3 si l = 4 este " << arie (3.0 , 4.0 );
return 0;
}
```

molnar1956@yahoo.com

Transmiterea prin valoare şi prin referinţă

Atunci când transmiteţi argumente (şi sunt variabile) unei funcţii, transmiteţi de fapt o copie a acelor variabile.

Aceasta este transmiterea prin valoare (*pass by value*).

Orice modificare a parametrilor unei funcţii este vizibilă numai în acea funcţie.

Variabilele - folosite ca argumente - rămân nemodificate.

```
#include <iostream>
using namespace std;
void modifica(int a)
{
a = a + 5;
cout << "a are valoarea: " << a << '\n';
}
int main()
{
int x = 1;
cout << "x inainte de apel: " << x << '\n';
modifica(x);
cout << "x dupa apel: " << x;
return 0;
}
```

molnar1956@yahoo.com

```
#include <iostream>
using namespace std ;

void modifica (int & a) // Nu uitati de ampersand !
{
        a = a + 5;
        cout << "a are valoarea: " << a << '\n' ;
}

int main ()
{
int x = 1;
cout << "x inainte de apel: " << x << '\n' ;
        modifica (x);
cout << "x dupa apel: " << x;
return 0;
}
```

molnar1956@yahoo.com

Vectorii se transmit prin referinţă!

```
#include <iostream>
using namespace std ;
void modifica (int v[], int l)
{
        for (int i = 0; i < l; i++)
        v[i] += 5; // v[i] = v[i] + 5;
}

int main ()
{
int w[] = {1, 2, 3, 4}, k = 4;
        modifica (w, k);
        for (int i = 0; i < k; i++)
        cout << w[i] << ' ' ;
return 0;
}
```

molnar1956@yahoo.com

Funcţii recursive

O funcţie care se autoapelează se numeşte *recursivă*.
Aveţi grijă ca funcţia să aibă o condiţie de terminare, altfel puteţi crea o repetiţie infinită.

Parametrii funcţiilor şi variabilele locale sunt încărcate, stocate, pe Stivă (o regiune din memorie structurată pe principiul stivei, LIFO), iar în cazul unei funcţii recursive infinite, Stiva se poate umple repede (*stack overflow*) cauzând un crash al programului.

Factorialul unui număr poate fi calculat cu o funcţie recursivă (deşi se poate face acelaşi lucru şi cu un loop).

molnar1956@yahoo.com

Factorialul unui număr poate fi calculat cu o funcţie recursive.

```
#include <iostream>
using namespace std ;
int fact (int n)
{
if (n == 0) // Conditia de terminare
        return 1;
    else
        return n * fact (n - 1);
}

int main ()
{
        cout << fact (0) << '\n' ;
        cout << fact (3) << '\n' ;
        cout << fact (8) << '\n' ;
return 0;
}
```