RO: Funcţii recursive.

EN: Recursive functions.

Obiective:

- Abilitatea de a proiecta algoritmi recursivi simpli şi de a-i implementa în limbajul C/C++;
- Înţelegerea mecanismului de încărcare şi eliberare a stivei la apelul funcţiilor recursive.

Rezumat:

O funcţie este recursivă, dacă executarea ei implică cel puţin un apel către ea însăşi (autoapel). Autoapelarea se poate realiza:

- direct, prin ea însăşi (*recursivitate directă*);
- fie prin intermediul altor funcţii care se apelează circular (*recursivitate indirectă*).

Caracteristici:

- Autoapelul se face pentru o dimensiune mai mică a problemei. Există o dimensiune a problemei pentru care nu se mai face autoapel, ci se revine din funcţie (eventual, cu returnarea unui rezultat).
- Pentru evitarea situaţiei în care funcţia se apelează pe ea însăşi la infinit, este obligatoriu ca autoapelul să fie legat de îndeplinirea unei condiţii care să asigure oprirea din acel ciclu de autoapeluri.
- Orice apel de funcţie provoacă în limbajul C/C++ memorarea pe stivă a parametrilor acelei funcţii şi a adresei de revenire la prima instrucţiune de după apel. La apeluri repetate, spaţiul ocupat pe stivă poate creşte înafara limitelor admise. De aceea se urmărește ca funcțiile recursive să aibă un număr minim de parametri, în acest caz recomandându-se a folosi parametrii globali dacă e posibil.
- La apelul recursiv al unei funcţii se creează o nouă instanţă de calcul pentru acea funcţie. La fiecare apel - pe stiva locală funcţiei se pun:
    - un nou set de variabile locale (inclusiv parametri formali), cu aceleași nume dar diferite ca zonă de memorie și cu valori diferite la fiecare apel
    - în fiecare moment e accesibilă doar variabila din vârful stivei

Utilizarea tehnicilor recursive nu conduce de obicei la soluţii optime, fiind recomandabilă analiza eficienţei soluţiei iterative (nerecursive) şi a celei recursive, alegându-se soluţia cea mai avantajoasă. Soluţia recursivă duce însă la soluţii mai elegante şi mai uşor de urmărit.

Exemple de algoritmi recursivi simpli:

- calculul factorialului;
- calculul şirului lui Fibonacci;
- funcţia Ackermann;
- cel mai mare divizor comun;
- calculul combinărilor;
- calculul sumei cifrelor unui numar, etc.

Exemple:

Atenţie:

- **Urmăriţi la fiecare program de mai jos valorile parametrilor de apel (cu QuickWatch, de exemplu: *fact (int)* si *n* pentru programul care calculeaza recursiv factorialul) şi stiva, rulând pas cu pas programul (cu tasta *F11*)!**
- **Faceţi modificările necesare pentru ca programul să verifice valorile parametrilor de apel sau valoarea returnată de funcţie aşa încât să se comporte bine şi pentru valori nepotrivite.**
- **Scrieţi aceeaşi funcţie în varianta iterativă.**
- **Contorizaţi numărul de paşi necesari în cazul recursiv, respectiv în cazul nerecursiv şi faceţi o comparaţie**

```
//*************************************************************************
//implementeaza functia factorial in mod recursiv
#include <iostream>
using namespace std;

int factorial(int n);

int main() {
int step;
cout << "\nEnter the step: "; cin >> step;
cout << "\nFactorial of step = " << step << endl<<factorial(step) << endl;
cout << "address 1";
}//main

int factorial(int n) {
cout << "\nStep = " << n;
if (n == 0) return 1;
else
return n * factorial(n - 1);  //address 2
}

//*************************************************************************
//Implementeaza functia de creare a valorilor lui Fibonacci in mod recursiv
#include <stdio.h>

int fib(int);

int main( ){
        printf("Fibonacci de 7 =%d",  fib(7));          //apelul functiei pentru al 7-lea element
        }//main

int fib(int n){
        if (n < 2)
                return n;//adica fib(1)=1, fib(0)=0
        else
                return (fib(n-1) + fib(n-2));           //apel recursiv
        }//fib

//*************************************************************************
//Solutia iterativa pentru generarea sirului lui Fibonacci pana la N dorit
#include <iostream>
using namespace std;

int main( ) {
        int l = 1, j = 0, k, n;
        cout << "Wished number is: ";
        cin >> n;
        cout<<"f[0]= 0\n";
        for (k = 1; k < n; k++)
        {
                j = l + j;
                l = j - l;
                cout <<"f[" <<k <<"]= " <<j <<endl;
        }//end for
}//main

//*************************************************************************
//Suma cifrelor unui numar in baza 10
//V(n)=0, daca n=0,
//V(n/10)+n%10
```

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

int suma_cifre_recursiv(int);

int main( ){
	int n;
	cout<<"Numarul ale carui cifre vor fi insumate: ";
	cin>>n;
	cout<<"Suma cifrelor este: " << suma_cifre_recursiv(n);
}

int suma_cifre_recursiv(int n){
	 if (n==0) return 0;
	 else return suma_cifre_recursiv(n/10)+n%10;
}
//*************************************************************************
//Media elementelor pare dintr-un sir de numere intregi in doua variante
#include<iostream>
using namespace std;
#define DIM 20
double mediaPare(int a[], int n);
int sumaPare(int a[], int n);

static int k;

int main( ) {
	int a[DIM], n, s;
	cout << "\nCate elemente? (n>0, n<20): ";
	cin >> n;
	for (int i = 0; i < n; i++)
		cin >> a[i];
	cout << "\nMedia elementelor pare (recursiv)= " << mediaPare(a, n);
	k = 0;
	s = sumaPare(a, n);
	cout << "\nSuma elementelor pare (recursiv)= " << s << " nr. pare = " << k <<" Media cu
suma= "<<s/(double)k;
}//main

double mediaPare(int a[], int n) {

	//int k=0; nu pastreaza contorul
	if (n == 0) return 0;
	else if (a[n - 1] % 2 == 0) { k++; return mediaPare(a, n - 1) + (double)a[n - 1] / k; }
	else	return mediaPare(a, n - 1);
}//mediaPare

int sumaPare(int a[], int n) {

	//int k=0; nu pastreaza contorul
	if (n == 0) return 0;
	else if (a[n - 1] % 2 == 0) { k++; return sumaPare(a, n - 1) + a[n - 1]; }
	else	return sumaPare(a, n - 1);
}//sumaPare
```

Teme:

1. Construiţi o funcţie recursivă care calculează A_n^k, unde n, k sunt citite de la tastatură ($A_n^k = n \cdot A_{n-1}^{k-1}$; la k=1 A_n^k este n). Verificati rezultatul folosind si metoda bazata pe factorial.

2. Calculati C_n^k, *n* si *k* fiind preluate de la tastatura, utilizând o funcţie recursivă ($C_n^k = \frac{n}{n-k} \cdot C_{n-1}^k$; pentru n=0, k=0 sau n=k, C_n^k este 1). Verificati rezultatul folosind si metoda bazata pe factorial.
3. Calculul celui mai mare divizor comun a două numere folosind o funcţie recursivă.
4. Se consideră recursivitatea (seria de medii arimetico-geometrice a lui Gauss):
   $a_n=(a_{n-1}+b_{n-1})/2$ şi
   $b_n=sqrt(a_{n-1}*b_{n-1})$, determinaţi a_n şi b_n, pentru n, a_0, b_0 introduse de la tastatură.
5. Citiţi un şir de caractere de la tastatură, caracter cu caracter, cu ajutorul unei funcţi bazate pe caracter. Afişaţi şirul în ordine inversă folosind o funcţie recursivă.
6. Determinaţi printr-o funcţie recursivă produsul scalar al doi vectori (tablouri unidimensionale de aceiasi lungime).
7. Să se calculeze suma numerelor impare dintr-un tablou unidimensional de numere întregi în mod recursiv, tablou citit dintr-un fişier unde, ca primă valoare, avem numărul de elemente ale tabloului.
8. Analog cu problema precedentă, dar se calculează produsul elementelor aflate pe poziţii impare într-un tablou unidimensional, respectiv să se calculeze suma numerelor prime din tablou.
9. Folosind o funcție recursivă, calculați suma valorilor introduse de la tastatură cu confirmare, adică cereți utilizatorului să indice dacă mai dorește să mai introducă o nouă valoare sau nu. Modificați funcția pentru a calcula și afișa, și media valorilor date de utilizator. Semnalați printr-un mesaj când suma valorilor depășește o anumită valoare prestabilită.
10. Considerați un șir de n (<=30) de valori întregi. Determinați în mod recursiv și nerecursiv numărul de apariții în șir ale unei valori întregi *x* cititte de la tastatură.
11. Considerați un număr *n* întreg pozitiv în baza 10 introdus de la tastatură. Folosind o funcție recursivă convertiți valoarea *n* într-o altă bază de numerație $1<b<10$ citită de la tastatură.
12. Fie ecuația de gradul 2: $x^2 - sx + p = 0$. Fără a calcula rădăcinile x_1 și x_2 determinați $S_n = x_1^n + x_2^n$, folosind reprezentarea recursivă a sumei: *Sum(n) ={ 2, dacă n=0; s, dacă n=1; s*Sum(n-1) – p*Sum(n-2), dacă n>1; }* unde *s* și *p* sunt valori reale iar *n* întreagă, introduse de la tastatură. Verificați dacă e posibil rezultatul obținut.
13. Scrieți un program care să calculeze în mod recursiv și în mod nerecursiv valoarea seriei armonice $s_n=1/1+1/2+1/3+\ldots 1/n$, unde *n* este un număr natural, cu două funcții diferite. Apelați cele două funcții cu diferite valori ale lui *n*.

Opţional activitate suplimentara:
-Aplicaţii folosind numărul lui Fibonacci, cu numărul de aur, secţiunea de aur, unghiul de aur. Semnificaţie, utilizare (inclusiv codare), optimalitate.
-Formula lui Moivre, ce face legătura între numerele complexe și trigonometrie. Daca avem un numar complex *z* e dat sub forma trigonometrica: $z=\rho(\cos\phi+i\sin\phi)$, atunci:
$z^n=[\rho(\cos\phi+i\sin\phi)]^n=\rho^n(\cos n\phi+i\sin n\phi)$. Formula lui Moivre poate folosita pentru a exprima $\cos n\phi$ si $\sin n\phi$ ca si puteri ale $\cos\phi$ si $\sin\phi$ in mod recursiv. Aplicații la transformata Fourier.
-Analiza corectitudinii unor expresii şi obţinerea formei poloneze postfixate, prefixate etc.
-Tehnici recursive de generare şi desenare a fractalilor, etc.

Homework:

1. Write a recursive function that calculates A_n^k, where n and k are read form the keyboard ($A_n^k = n \cdot A_{n-1}^{k-1}$; if k=1 A_n^k is n). Verify the result using the factorial definition.
2. Write a recursive function that calculates C_n^k, where *n* and *k* are read from the keyboard ($C_n^k = \frac{n}{n-k} \cdot C_{n-1}^k$; if n=0, k=0 or if n=k, C_n^k is 1). Verify the result using the factorial definition.
3. Calculate the greatest common divider of 2 numbers using a recursive function.
4. Considering the following recursive formulas (Gauss arithmetical-geometrical media):
   $a_n=(a_{n-1}+b_{n-1})/2$ and $b_n=sqrt(a_{n-1}*b_{n-1})$,
   determine a_n and b_n for n, a0 and b0 read from the standard input.

5. Read a string of caracters from the keyboard, one character at a time, using a character function. Reverse the string using a recursive function.
6. Determine the scalar product of 2 vectors using a recursive function (one-dimensional array same dimension).
7. Calculate the sum of the odd numbers from an array of integer values, using a recursive function. The numbers from the array are read from a file. The first value in the array represents the array's length.
8. Using the code developed for the previous problem, calculate the sum of the prime numbers in the array.
9. Using a recursive function, calculate the sum of a series composed of keyboard entered values. The values are read as long as the user desires so. Modify the function in order to determine and display the average value of the entered numbers. Print on the screen a significant message when the sum is greater than a predefined value.
10. Consider an array of *n* (<=30) integer_values. Determine (recursively and non-recursively) the number of times a certain value *x* read from the keyboard appears in the array.
11. Read from the keyboard a positive integer value *n* (base 10). Use a recursive function for converting *n* into another base 1< *b<10*, also read from the keyboard.
12. Consider the 2-nd degree equation $x^2 - sx + p = 0$. Without calculating the solutions x_1 and x_2 determine $S_n = x_1^n + x_2^n$ using this sum's recursive definition: *Sum(n) ={ 2, if n=0; s, if n=1; s*Sum(n-1) – p*Sum(n-2), if n>1; }*. *s* and *p* are float values, *n* is an integer value, all read from the keyboard. Verify the result if it is possible.
13. Write a program that calculates recursively and non-recursively (two distinct functions) the value of the harmonic series s_n=1/1+1/2+1/3+…1/n, where *n* is a natural number. Call the functions with 2 different values for *n*.