MINISTERUL EDUCAȚIEI AL REPUBLICII MOLDOVA

UNIVERSITATEA TEHNICĂ A MOLDOVEI

**RAPORT**

LUCRAREA DE LABORATOR NR. 3

LA PROGRAMAREA CALCULATOARELOR

Elaborat: st. gr. TI – 211, Popa Ecaterina

Verificat: Stratulat Ștefan

UTM, Chișinău, 2021

**Scopul lucrării**

Scopul lucrării este de a familiariza studentul cu scrierea și utilizarea funcțiilor și a recursiei, utilizînd limbajul C.

**Probleme pentru utilizarea funcțiilor:**

1. Scrieți un program C pentru a găsi cubul oricărui număr folosind funcția.

#include <stdio.h>

#include <math.h>

void main () {

int a;

printf ("Numarul este ");

scanf ("%d", &a);

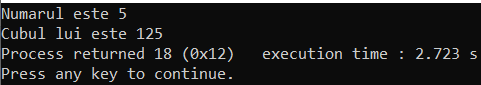
printf ("Cubul lui este %d", cub(a));

}

int cub (int a) {

return pow (a, 3);

}



1. Scrieți un program C pentru a găsi diametrul, circumferința și aria cercului folosind funcțiile.

#include <stdio.h>

#include <math.h>

double dim (double);

double circ (double);

double aria (double);

void main () {

double r;

printf ("Raza este ");

scanf ("%lf", &r);

printf ("Diametrul este %.2lf \n", dim(r));

printf ("Circumferinta este %.2lf \n", circ(r));

printf ("Aria este %.2lf \n", aria(r));

}

double dim (double r) {

double rez=2\*r;

return rez;

}

double circ (double r) {

const double pi=3.14;

return 2\*pi\*r;

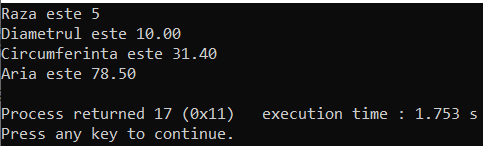
}

double aria (double r) {

const double pi=3.14;

return pi\*pow(r, 2);

}



1. Scrieți un program C pentru a găsi maxim și minim între două numere folosind funcții.

#include <stdio.h>

void main () {

int a, b;

printf ("a = ");

scanf ("%d", &a);

printf ("b = ");

scanf ("%d", &b);

printf ("Maxim = %d \n", max(a, b));

printf ("Minim = %d", min(a, b));

}

int max (a, b) {

if (a>b) {

return a;

} else {

return b;

}

}

int min (a, b) {

if (a<b) {

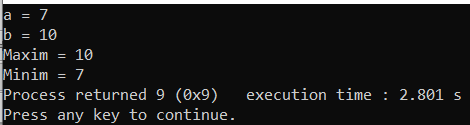
return a;

} else {

return b;

}

}



1. Scrieți un program C pentru a verifica dacă un număr este par sau impar folosind funcțiile.

#include <stdio.h>

void main () {

int n;

printf ("Numarul este ");

scanf ("%d", &n);

par(n);

}

void par (int n) {

if (n%2==0) {

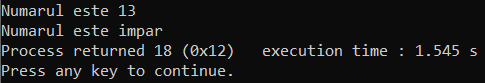
printf ("Numarul este par");

} else {

printf ("Numarul este impar");

}

}



1. Scrieți un program C pentru a verifica dacă un număr este prim, Armstrong sau un număr perfect folosind funcții.

#include <stdio.h>

#include <math.h>

int main()

{

int x, res=0;

printf("Introduceti n: ");

scanf("%d",&x);

res=prim(x);

if (res==0) {

printf("Este numar prim \n");

}

else {

printf("Nu este numar prim \n");

}

arm(x);

perfect(x);

}

int prim(int n) {

int i;

for(i=2; i<=n/2; i++) {

if(n%i!=0)

continue;

else

return 1;

}

return 0;

}

void arm(int n) {

int temp, r, sum=0;

temp=n;

while (n>0) {

r=n%10;

sum=sum+pow(r, 3);

n=n/10;

}

if (temp==sum)

printf("Este numar Armstrong",temp);

else

printf("Nu este un numar Armstrong",temp);

}

void perfect(int n){

int i, sum=0;

for(i=1; i<n; i++) {

if(n%i==0) {

sum=sum+i;}

}

if (sum==n) {

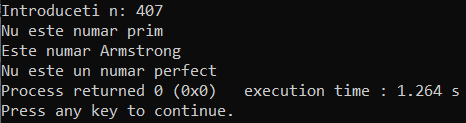
printf("\nNumar perfect",n); }

else {

printf("\nNu este un numar perfect", n);

}

}



1. Scrieți un program C pentru a găsi toate numerele prime între intervalul dat folosind funcțiile.

#include <stdio.h>

#include <math.h>

//Fie intervalul de la a la b

void main()

{

int a, b;

printf("Introduceti primul nr: ");

scanf("%d",&a);

printf("Introduceti al doilea nr: ");

scanf("%d",&b);

prim(a, b);

}

int prim(int a, int b)

{

int i,j;

for (i=a; i<=b; i++)

{

int k;

for(k=i; k<=b; k++)

{

int c=0;

for(j=1; j<=k; j++)

{

if(k%j==0)

{

c++;

}

}

if(c==2)

{

printf("%d ",k);

}

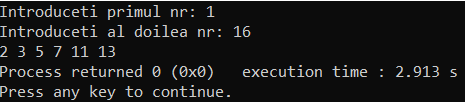
break;

}

}

return 0;

}



1. Scrieți un program C pentru a imprima toate numerele puternice între intervalul dat folosind funcțiile.

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

int puternic(int n)

{

while (n % 2 == 0)

{

int putere = 0;

while (n % 2 == 0)

{

n /= 2;

putere++;

}

if (putere == 1)

return 0;

}

for (int factorial = 3; factorial <= sqrt(n); factorial += 2)

{

int putere = 0;

while (n % factorial == 0)

{

n = n / factorial;

putere++;

}

if (putere == 1)

return 0;

}

return (n == 1);

}

int main()

{

int a, b;

printf("Introduceti inceputul si sfarsitul sirului: ");

scanf("%d%d", &a, &b);

for (int i = a; i <= b; i++)

{

if (puternic(i))

{

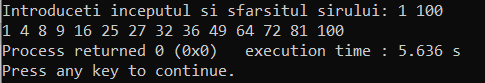
printf("%d ", i);

}

}

return 0;

}



1. Scrieți un program C pentru a imprima toate numerele Armstrong între intervalul dat folosind funcțiile.

#include <stdio.h>

int Armstrong(int n);

void printArmstrong(int inceput, int sfarsit);

int main()

{

int inceput, sfarsit;

printf("Introdu inceputul si sfarsitul sirului: ");

scanf("%d%d", &inceput, &sfarsit);

printf("Toate numerele Armstrong intre %d si %d sunt: \n", inceput, sfarsit);

printArmstrong(inceput, sfarsit);

return 0;

}

int Armstrong(int n)

{

int orig, ultimacifra, suma;

orig = n;

suma = 0;

while(orig != 0)

{

ultimacifra = orig % 10;

suma += ultimacifra \* ultimacifra \* ultimacifra;

orig /= 10;

}

if(n == suma)

return 1;

else

return 0;

}

void printArmstrong(int inceput, int sfarsit)

{

while(inceput <= sfarsit)

{

if(Armstrong(inceput))

{

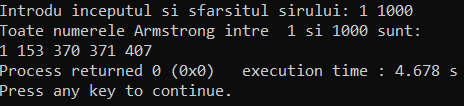
printf("%d ", inceput);

}

inceput++;

}

}



1. Scrieți un program C pentru a imprima toate numerele perfecte între intervalul dat folosind funcțiile.

#include <stdio.h>

int Perfect(int n);

void printPerfect(int inceput, int sfarsit);

int main()

{

int inceput, sfarsit;

printf("Introdu inceputul si sfarsitul sirului: ");

scanf("%d%d", &inceput, &sfarsit);

printf("Toate numerele perfecte de la %d pana la %d sunt: \n", inceput, sfarsit);

printPerfect(inceput, sfarsit);

return 0;

}

int Perfect(int n)

{

int i, suma;

suma = 0;

for(i=1; i<n; i++)

{

if(n % i == 0)

{

suma += i;

}

}

if(suma == n)

return 1;

else

return 0;

}

void printPerfect(int inceput, int sfarsit)

{

while(inceput <= sfarsit)

{

if(Perfect(inceput))

{

printf("%d ", inceput);

}

inceput++;

}

}

**Probleme pentru utilizarea funcțiilor recursive:**

1. Scrieți un program C pentru a găsi puterea oricărui număr folosind recursivitatea.

#include <stdio.h>

int putere(int x, int y);

int main()

{

int baza, p, rezultat;

printf("Baza: ");

scanf("%d", &baza);

printf("Puterea: ");

scanf("%d", &p);

rezultat = putere(baza, p);

printf("%d^%d = %d", baza, p, rezultat);

return 0;

}

int putere(int baza, int p)

{

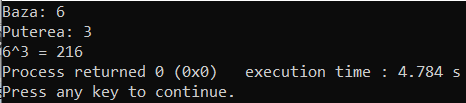
if (p != 0)

return (baza \* putere(baza, p - 1));

else

return 1;

}



1. Scrieți un program C pentru a imprima toate numerele naturale cuprinse între 1 și n folosind recursivitatea.

#include<stdio.h>

void printNumar(int sfarsit);

int main()

{

int sfarsit;

printf("Ultimul termen al sirului: \n");

scanf("%d", &sfarsit);

printf("Numerele naturale de la 1 pana la %d sunt:", sfarsit);

printNumar(sfarsit);

return 0;

}

void printNumar(int n)

{

if(n)

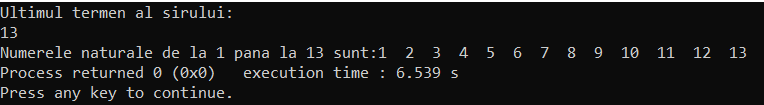
printNumar(n-1);

else

return;

printf("%d ", n);

}



1. Scrieți un program C pentru a imprima toate numerele pare sau impare în intervalul dat folosind recursivitatea.

#include <stdio.h>

#include <stdlib.h>

void printParImpar (int n, int limita);

int main()

{

int inceput, sfarsit;

printf("Introdu inceputul sirului\n");

scanf("%d",&inceput);

printf("Introdu sfarsitul sirului\n");

scanf("%d",&sfarsit);

printf("Numerele pare/impare de la %d pana la %d sunt: ",inceput,sfarsit);

printParImpar(inceput,sfarsit);

return 0;

}

void printParImpar(int n, int limita)

{

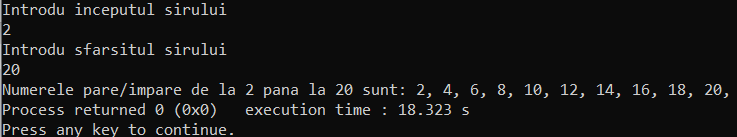
if(n>limita)

return;

printf("%d, ",n);

printParImpar(n + 2, limita);

}



1. Scrieți un program C pentru a găsi suma tuturor numerelor naturale cuprinse între 1 și n folosind recursivitatea.

#include <stdio.h>

int suma(int n);

int main() {

int sfarsit;

printf("Introdu sfarsitul sirului: ");

scanf("%d", &sfarsit);

printf("Suma numerelor de la 1 la %d este %d", sfarsit,suma(sfarsit));

return 0;

}

int suma(int n) {

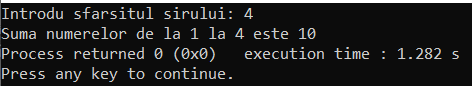
if (n != 0)

return n + suma(n - 1);

else

return n;

}



1. Scrieți un program C pentru a găsi suma tuturor numerelor pare sau impare în intervalul dat folosind recursivitatea.

#include <stdio.h>

int ParImpar(int inceput, int sfarsit);

int main()

{

int inceput, sfarsit, suma;

printf("Primul termen: ");

scanf("%d", &inceput);

printf("Ultimul termen: ");

scanf("%d", &sfarsit);

printf("Suma numerelor pare/impare de la %d pana la %d = %d\n", inceput, sfarsit, ParImpar(inceput, sfarsit));

return 0;

}

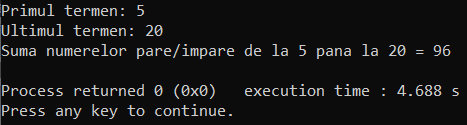
int ParImpar(int inceput, int sfarsit)

{

if(inceput > sfarsit)

return 0;

else

return (inceput + ParImpar(inceput + 2, sfarsit));

1. Scrieți un program C pentru a găsi inversarea oricărui număr folosind recursivitatea.

( Ex: 2021 -> 1202)

#include<stdio.h>

int main()

{

int n,invers;

printf("Introdu un numar:");

scanf("%d",&n);

invers=inversiune(n);

printf("Inversul lui %d este %d",n,invers);

return 0;

}

int suma=0,rest;

inversiune(int n)

{

if(n)

{

rest=n%10;

suma=suma\*10+rest;

inversiune(n/10);

}

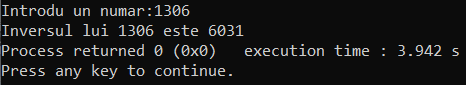
else

return suma;

return suma;

}

}



1. Scrieți un program C pentru a verifica dacă un număr este palindrom folosind recursivitatea.

#include <stdio.h>

#include <math.h>

int invers(int n);

int palindrom(int n);

int main()

{

int n;

printf("Numarul: ");

scanf("%d", &n);

if(palindrom(n) == 1)

{

printf("%d este un palindrom.\n", n);

}

else

{

printf("%d nu este un palindrom.\n", n);

}

return 0;

}

int palindrom(int n)

{

if(n == invers(n))

{

return 1;

}

return 0;

}

int invers(int n)

{

int cifra = (int)log10(n);

if(n == 0)

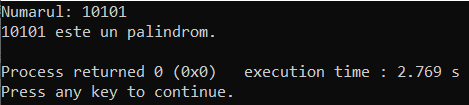
{

return 0;

}

return ((n%10 \* pow(10, cifra)) + invers(n/10));

}



1. Scrieți un program C pentru a găsi suma cifrelor unui număr dat folosind recursivitatea.

#include <stdio.h>

int sumacifrelor(int n);

int main()

{

int n, suma;

printf("Introduceti un numar: ");

scanf("%d", &n);

suma = sumacifrelor(n);

printf("Suma cifrelor numarului %d este %d", n, suma);

return 0;

}

int sumacifrelor(int n)

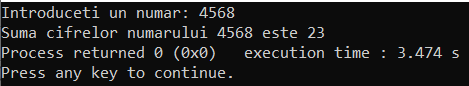
{

if(n == 0)

return 0;

return ((n % 10) + sumacifrelor(n / 10));

}



1. Scrieți un program C pentru a găsi factorialul oricărui număr folosind recursivitatea.

#include<stdio.h>

int factorial(int n);

int main() {

int n;

printf("n #= ");

scanf("%d",&n);

printf("%d! este %d", n, factorial(n));

return 0;

}

int factorial(int n)

{

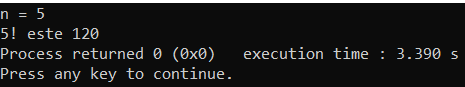
if (n>=1)

return n\*factorial(n-1);

else

return 1;

}



1. Scrieți un program C pentru a genera al n-lea termen Fibonacci folosind recursivitate.

#include <stdio.h>

int fib(int n);

int main()

{

int n;

long int fibonacci;

printf("n = ");

scanf("%d", &n);

fibonacci = fib(n);

printf("Al %d termen fibonacci este %ld", n, fibonacci);

return 0;

}

int fib(int n)

{

if(n == 0)

return 0;

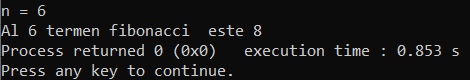
else if(n == 1)

return 1;

else

return fib(n-1) + fib(n-2);

}



**Rezultatul lucrării:**

1. Ce reprezintă o funcție în limbajul C?

O funcție în limbajul C reprezintă un modul independent al programului, o diviziune a unui program mai complex, folosit pentru a pune în evidență anumite sarcini concrete în cadrul acestora.

1. Cum se declară o funcție în limbajul C?

Declararea funcției în limbajul C are loc prin declararea prototipului funcției, fără corpul acesteia: se declară tipul de date returnat, numele funcției și argumentele acesteia, necesare la apelarea ulterioară a funcției.

<tip\_date\_returnat> <nume\_functie> ([lista\_parametrilor\_formali]);

1. Cum se apelează o funcție în limbajul C?

Apelul unei funcții în limbajul C are loc prin identificarea numelui funcției și, a parametrilor actuali, între paranteze rotunde.

1. Ce scop are instrucțiunea **return** pentru funcții în limbajul C?

Scopul funcției return în limbajul C este de a returna o expresie identică ca tip cu tipul funcției declarate.

1. Ce reprezintă lista de parametri formali pentru o funcție în limbajul C?

Lista de parametri formali reprezintă parametrii definiți în antetul unei funcții și utilizate în cadrul acestei funcții.

1. Ce înseamnă o funcție recursivă?

O funcție recursivă reprezintă o funcție care se auto-apelează: permite apelarea ei însăși în interiorul aceluiași program.

1. Cum se realizează în limbajul C o funcție recursivă?

În limbajul C deosebim recursivitate directă – în definiția funcției regăsim un apel al funcției însăși, și indirectă – când în corpul funcției există un apel al altei funcții ce conține la rândul ei un apel al funcției recursive.

1. Care este diferența între o funcție simplă și o funcție cu recursie?

Funcția simplă este o componentă independentă a programului, cu corpul ei, parametrii de intrare și ieșire; pe când funcția cu recursie permite rezolvarea problemelor prin auto-apelarea ei însăși.

1. De ce este o practică bună de a utiliza funcții în program ?

E o practică bună de a utiliza funcțiile în program, deoarece ne permit să împărțim o sarcină maii dificilă în multe mai mici, ce permit o rezolvare mai rapidă a acesteia.