1. **FUNCŢII DEFINITE DE UTILIZATOR**
   1. **SCOPUL LUCRĂRII**

Scopul lucrării este de a prezenta una dintre armele cele mai puternice ale programării procedurale, anume aceea a utilizării, alături de funcţiile predefinite din biblioteci, a funcţiilor definite de către utilizator.

Se exersează comparativ transmiterea parametrilor prin valoare şi prin referinţă.

Aplicaţiile din exemple şi din problemele propuse spre rezolvare fac apel la variabilele locale şi globale

* 1. **BREVIAR TEORETIC**

Pentru utilizarea funcţiilor în C++, este necesară cunoaşterea următoarelor aspecte de bază:

a) Funcţiile grupează setul de operatori pentru îndeplinirea unei sarcini concrete.

b) Programul principal apelează la funcţie, adresîndu-se la numele ei, după care urmează paranteze rotunde.

c) După terminarea prelucrării informaţiei, majoritatea funcţiilor întorc programului principal valori de tipuri concrete, care pot fi olosite în calcule.

d) Programul principal transmite funcţiilor parametrii (informaţia iniţială), inclusă în paranteze rotunde, care urmează după numele funcţiei.

e) Limbajul C++ foloseşte prototipuri de funcţie pentru determinarea tipului valorii returnate de câtre funcţie, a cantităţii si tipurilor parametrilor transmişi funcţiei.

În timpul creării programului, e necesar să se rezerve fiecare funcţie pentru rezolvarea unei sarcini. Fiecare funcţie creată trebuie să  primească un nume unic.

Ca şi în cazul variabilelor, numele unei funcţii este un identificator şi e de dorit să corespundă cu sensul logic al sarcinei pe care o îndeplineşte.

Funcţiile din C++ se aseamănă structural cu funcţia principală main().

În faţa numelui funcţiei se indică ***tipul*** ei, iar după numele funcţiei urmează ***lista de parametri*** descrişi înăuntrul parantezelor rotunde.

***Corpul funcţiei*** (compus din operatori) este amplasat după descrierea parametrilor, între acolade.

Sintaxa descrierii unei funcţii:

*tip\_f nume\_f (lista parametri) {declarare de variabile; operatori;}*

unde

tip\_f  - tipul funcţiei sau tipul valorii returnate de funcţie,

nume\_f  - numele funcţiei.

Dacă funcţia nu întoarce valori, tipul ei este void.

***1. Transmiterea parametrilor în funcţie.***

Dacă funcţia foloseşte parametri, ei trebuie descrişi în cadrul descrierii funcţiei.

În timpul descrierii parametrelor funcţiei se indică numeleşi tipul fiecărui parametru.

*tip\_parametru nume\_parametru;*

Dacă funcţia conţine mai mulţi parametri, ei vor fi descrişi împreună între parantezele rotunde după numele funcţiei, despărţiţi prin virgulă

*tip\_funcţie nume\_funcţie (tip\_parametru1 nume\_parametru1,*

*tip\_parametru2 nume\_parametru2*

*….*

*tip\_parametruN nume\_parametruN );*

În unele surse de descriere a limbajului de programare C/C++, parametrii ce se transmit din program în funcţie se numesc ***actuali***, iar parametrii ce sunt declaraţi în antetul funcţiei şi cărora li se atribuie valorile parametrilor actuali, se numesc parametri ***formali***.

În momentul folosirii parametrilor în funcţie, este necesară cunoaşterea următoarelor aspecte:

- Dacă funcţia foloseşte parametri, ea trebuie să indice numele unic şi tipul fiecărui parametru.

- Când programul apelează funcţia, compilatorul atribuie valoarea parametrilor de la stânga la dreapta

- Valorile transmise din program în funcţie, trebuie să coincidă ca număr, loc şi tip cu parametrii din funcţie.

***2. Întoarcerea valorilor din funcţie.***

Obiectivul oricărei funcţii este îndeplinirea unei sarcini concrete. În majoritatea cazurilor, funcţiile vor efectua calcule. După aceasta, funcţia va întoarce rezultatul fie funcţiei din care a fost apelată, fie funcţiei principale main. În momentul când funcţia întoarce o valoare, trebuie să fie cunoscut tipul ei. Tipul valorii returnate de funcţie se indică în momentul descrierii funcţiei, înainte de numele ei. ***Tipul valorii returnate*** se mai numeşte şi ***tipul funcţiei***.

Funcţiile folosesc operatorul ***return*** pentru a întoarce valori funcţiilor din care au fost apelate.

Când compilatorul întâlneşte operatorul ***return***, el întoarce valoarea dată şi încheie executarea funcţiei curente, controlul executării programului fiind cedat funcţiei din care a fost chemată funcţia curentă. Dacă după operatorul return, în funcţie mai există şi alţi operatori, ei vor fi ignoraţi, funcţia terminîndu-se odată cu îndeplinirea operatorului return.

Valoarea întoarsă de funcţie poate fi folosită în orice loc al programului, unde e posibilă folosirea unei valori de tip identic cu valoarea returnată. Când funcţia întoarce o valoare, această valoare poate fi atribuită unei variabile de acelaşi tip, folosind operatorul de atribuire. Valoarea mai poate fi folosită în cadrul instrucţiunilor de afişare, în cadrul instrucţiunilor de decizie sau a instrucţiunulor de ciclare.

De asemenea, ea poate fi folosită ca parametru în cadrl apelului altei funcţii.

***3. Prototipul funcţiei.***

Înainte de apelul unei funcţii, compilatorul C++ trebuie să cunoască tipul valorii returnate, cantitatea şi tipul parametrilor folosiţi de funcţie.

Există însă situaţii cînd unele funcţii în program sînt apelate reciproc. În aceste cazuri, este posibilă situaţia cînd o funcţie va fi apelată înaintea descrierii sale.

În acest caz, se folosesc prototipuri ale funcţiilor. ***Prototipul unei funcţii*** este amplasat la începutul programului şi conţine informaţia despre tipul valorii returnate, cantitatea şi tipul parametrilor folosiţi de funcţie.

Odată declarat prototipul unei funcţii, înainte de a fi început corpul programului, descrierea funcţiei poate fi făcută după acolada de închidere a programului principal.

***4. Variabile localeşi domeniul de vizibilitate.***

Apare adeseori necesitatea folosirii în funcţii a variabilelor proprii.Variabilele declarate în cadrul funcţiei e numesc ***variabile locale***. Numele şi valoarea unei variabile locale sunt cunoscute numai funcţiei în care ea a fost declarată. Chiar faptul că variabila locală există este cunoscut numai de către funcţia în are ea a fostd eclarată. Declararea variabilelor are loc la începutul funcţiei, imediat după acolada care deschide corpul acesteia. Numele variabilei locale trebuie să fie unic în funcţia în care a fost declarată. O variabilăse numeşte locală, din cauză că este văzută numai din funcţia în care a fost descrisă.

Sintaxa de declarare a unei variabile locale:

*tip\_f numele\_f (lista parametrilor) {tip\_vl numele\_vl;}*

unde:

tip\_f  - tipul funcţiei;

nume\_f - numele funcţiei;

tip\_vl - tipul variabilei;

numele\_vl - numele variabilei;

Principiile de declarare şi folosire a unei variabile locale oricărei funcţii sunt identice cu principiile de declarare şi utilizare a unei variabile declarate în corpul funcţiei principalemain( );

O variabilă declarată în corpul funcţiei main() este şi ea locală acestei funcţii.

În general, tot ceea ce este valabil pentru a fost spus despre variabilele declarate în funcţia main( ) - tipurile, numele, principiile de utilizare ş.a.- rămâne valabil şi pentru o variabilă locală, din orice altă funcţie.

***5. Variabile globale***

Numim ***variabilă globală*** o variabilă pentru care numele şi valoarea sunt cunoscute pe parcursul întregului program, orice funcţie din acest program.

Pentru a crea o variabilă globală, se foloseşte declararea ei la începutul programului, în afara oricărei funcţii. Orice funcţie (inclusiv funcţia main), care va urma după această declarare, poate folosi această variabilă globală.

Declararea unei variabile globale:

*# include<…>*

*# include<…>*

*…*

*# include<…>*

*tip\_vg nume\_vg;*

*eventuale declaraţii de funcţii*

*void main (void){…}*

unde ***tip\_vg***  este tipul variabilei globale, iar ***nume\_vg*** – numele variabilei globale.

Fiind declarată o variabilă globală, valoarea ei nu numai că e cunoscută oricărei funcţii din program, dar şi poate fi şi schimbată de către oricare dintre funcţiile prezente în program.

Cu toate că prezenţa variabilelor globale în program adaogă noi posibilităţi, este de dorit să nu se facă abuz de folosirea lor. Din cauză că orice funcţie din program poate schimba valoarea variabilei globale, este dificil de urmărit toate funcţiile care ar putea schimba această valoare, ceea ce conduce la un control dificil asupra execuţiei programului

***6. Conflicte dintre variabile localeşi globale***

În cazul în care un program trebuie să folosească o variabilă globală, poate apărea o situaţie de conflict între numele variabilei globale şi numele unei variabile locale. În aceste cazuri, limbajul C++ oferă prioritate variabilei locale.

Dacă există o variabilă globală cu acelaşi nume ca o variabila locală, compilatorul consideră, că orice apel al variabilei cu acest nume este un apel al variabilei locale.

Există situaţii, când apare necesitatea de a se adresa o variabilă globală care se află în conflict cu o variabilă locală. În acest caz, se poate folosi operatorul global de acces ***::*** .

* 1. **EXEMPLE**

1. ***Transmiterea parametrilor prin valoare şi prin referinţă***

*#include <conio.h>*

*#include <iostream.h>*

*#include <math.h>*

*float f1(int a,int b,int c){*

*a=a\*a;*

*b=b\*b;*

*c=c\*c;*

*return sqrt(a+b+c);*

*}*

*// Varianta 1*

*float f2(int &a,int &b,int &c){*

*a=a\*a;*

*b=b\*b;*

*c=c\*c;*

*return sqrt(a+b+c);*

*}*

*/\**

*\*/*

*/\**

*//Varianta 2*

*float f2(int \*a,int \*b,int \*c){*

*\*a=\*a\* \*a;*

*\*b=\*b\* \*b;*

*\*c=\*c\* \*c;*

*return sqrt(\*a+\*b+\*c);*

*}*

*\*/*

*void main(){*

*int x,y,z;*

*cout<<"Introduceti pe x : "; cin>>x;*

*cout<<"Introduceti pe y : "; cin>>y;*

*cout<<"Introduceti pe z : "; cin>>z;*

*cout<<"Rezultatul aplicarii lui f1 : "<<f1(x,y,z);*

*cout<<"x="<<x<<endl;*

*cout<<"y="<<y<<endl;*

*cout<<"z="<<z<<endl;*

*// Varianta 1*

*cout<<"Rezultatul aplicarii lui f2 : "<<f2(x,y,z);*

*/\**

*//Varianta 2*

*cout<<"Rezultatul aplicarii lui f2 : "<<f2(&x,&y,&z);*

*\*/*

*cout<<"x="<<x<<endl;*

*cout<<"y="<<y<<endl;*

*cout<<"z="<<z<<endl;*

*getch();*

*}*

1. ***Se înlocuieşte fiecare element al unui vector prin oglinditul său***

*#include <conio.h>*

*#include <iostream.h>*

*#include <string.h>*

*unsigned long int oglinda(unsigned long int n){*

*unsigned long int m=0;*

*int c;*

*while(n){*

*m=m\*10+n%10;*

*n=n/10;*

*}*

*return m;*

*}*

*void main(){*

*clrscr();*

*unsigned long int v[100];*

*int n, i;*

*cout<<"n=? "; cin>>n;*

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

*cout<<"v["<<i<<"]=? ";*

*cin>>v[i];*

*}*

*for(i=1; i<=n; i++) cout<<v[i]<<" ";*

*cout<<endl;*

*for(i=1; i<=n; i++) v[i]=oglinda(v[i]);*

*for(i=1; i<=n; i++) cout<<v[i]<<" ";*

*cout<<endl;*

*getch();*

*}*

1. ***Din vectorul v[] se construieşte vectorul w[], care conţine pe poziţia i valoarea 1, dacă v[i] este prim şi valoarea 0, în caz contrar***

*#include <conio.h>*

*#include <iostream.h>*

*#include <string.h>*

*#include <math.h>*

*int prim(unsigned long int n){*

*int i;*

*for(i=2; i<=sqrt(n); i++) if(n%i==0) return 0;*

*return 1;*

*}*

*void main(){*

*clrscr();*

*unsigned long int v[100];*

*int n, i, w[100];*

*cout<<"n=? "; cin>>n;*

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

*cout<<"v["<<i<<"]=? ";*

*cin>>v[i];*

*}*

*for(i=1; i<=n; i++) cout<<v[i]<<" ";*

*cout<<endl;*

*for(i=1; i<=n; i++) w[i]=prim(v[i]);*

*for(i=1; i<=n; i++) cout<<w[i]<<" ";*

*getch();*

*}*

1. ***Cmmdc al tuturor elementelor unui vector***

*#include <conio.h>*

*#include <iostream.h>*

*unsigned long int cmmdc(unsigned long int a, unsigned long int b){*

*if(a==b) return a;*

*while(a!=b) if(a<b) b=b-a;*

*else a=a-b;*

*return a;*

*}*

*void main() {*

*clrscr();*

*unsigned long int v[100], div;*

*int n, i;*

*cout<<"n=? "; cin>>n;*

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

*cout<<"v["<<i<<"]=? "; cin>>v[i];*

*}*

*div=cmmdc(v[1],v[2]);*

*for(i=3; i<=n; i++)*

*div=cmmdc(div,v[i]);*

*cout<<div;*

*getch();*

*}*

1. ***Aflarea unei rădăcini unice dintr-un interval***

Se pune problema găsirii unei rădăcini ***unice*** a unei ecuaţii de forma ***f(x) = 0***, într-un interval ***(a,b)*** dat. Se garantează (din considerente externe, care nu au importanţă în rezolvarea problemei) că în intervalul dat rădăcina ***există*** şi ***este unică***. Fără această presupunere, algorimul prezentat nu este valabil.

Se poate observa că, în aceste condiţii, semnul funcţiei ***f(x)*** este diferit în punctul ***a***, de cel al funcţiei în punctul ***b*** (graficul funcţiei taie axa ***x*** o singură dată) => ***f(a)f(b) < 0***.

Atunci, împărţind intervalul ***(a,b)*** în două părţi egale, obţinem un punct ***m = (a+b)/2***, iar graficul funcţiei va tăia axa ***x***, fie în punctul ***m*** (caz în care algoritmul s-a încheiat), fie în intervalul ***(a,m)*** – dacă ***f(a)f(m) < 0***, fie în intervalul ***(m,b)*** – dacă ***f(m)f(b) < 0***.

Continuăm procesul până când obţinem o precizie dorită (***|f(x)| < ɛ***) unde ***ɛ*** este o valoare introdusă de la tastatură.

*#include <conio.h>*

*#include <iostream.h>*

*#include <math.h>*

*float f(float x){*

*return x\*x-3\*x+2;*

*}*

*void main(){*

*clrscr();*

*float a, b, mijloc, epsi, i=0;*

*cout<<"a=";cin>>a;*

*cout<<"b=";cin>>b;*

*cout<<"epsi=";cin>>epsi;*

*mijloc=(a+b)/2;*

*cout<<"punctul initial "<<mijloc<<" f(mijloc)= "<<f(mijloc)<<endl;*

*while(fabs(f(mijloc))>epsi){*

*i++;*

*if(f(a)\*f(mijloc)<0) b=mijloc; else a=mijloc;*

*mijloc=(a+b)/2;*

*cout<<"iteratia "<<i<<" mijloc="<<mijloc<<" f(mijloc)="<<f(mijloc)<<endl;*

*}*

*cout<<"radacina este "<<mijloc;*

*getch();*

*}*

1. ***Maxime şi minime pe liniile şi coloanele unei matrice***

*#include <iostream.h>*

*#include <conio.h>*

*void maxmin(int v[10],int n,int &maxim,int &minim){*

*int i;*

*maxim=v[1];*

*minim=v[1];*

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

*if (maxim<v[i]) maxim=v[i];*

*if (minim>v[i]) minim=v[i];*

*};*

*};*

*void main(){*

*// Maxime si minime in matrice*

*/\* Semnificatia variabilelor:*

*mxln[] - vectorul maximelor pe linii*

*mnln[] - vectorul minimelor pe linii*

*mxcl[] - vectorul maximelor pe coloane*

*mncl[] - vectorul minimelor pe coloane*

*mxmxln - maximul din maximele pe linii*

*mxmnln - maximul din minimele pe linii*

*mnmxln - minimul din maximele pe linii*

*mnmnln - minimul din minimele pe linii*

*mxmxcl - maximul din maximele pe coloane*

*mxmncl - maximul din minimele pe coloane*

*mnmxcl - minimul din maximele pe coloane*

*mnmncl - minimul din minimele pe coloane \*/*

*clrscr();*

*int a[10][10], ajut[10], mxln[10], mnln[10], mxcl[10], mncl[10];*

*int n, m, i, j,k, mxmxln ,mxmnln, mnmxln, mnmnln, mxmxcl, mxmncl, mnmxcl, mnmncl;*

*cout<<"Introduceti numarul de linii : ";*

*cin>>m;*

*cout<<"Introduceti numarul de coloane : ";*

*cin>>n;*

*cout<<"Introduceti elementele matricii\n\n";*

*for (i=1;i<=m;i++)*

*for (j=1;j<=n;j++) {*

*cout<<"a["<<i<<"]["<<j<<"]=";*

*cin>>a[i][j];*

*};*

*cout<<endl;*

*for (i=1;i<=m;i++){*

*for (j=1;j<=n;j++)*

*cout<<a[i][j]<<" ";*

*cout<<endl;*

*};*

*cout<<endl;*

*for (i=1;i<=m;i++) {*

*for (j=1;j<=n;j++) ajut[j]=a[i][j];*

*maxmin(ajut,n,mxln[i],mnln[i]);*

*cout<<"linia "<<i<<" : "<<"max="<<mxln[i]<<" min="<<mnln[i]<<endl;*

*}*

*cout<<endl;*

*maxmin(mxln,m,mxmxln,mnmxln);*

*cout<<"Maximumul din maximele pe linii este "<<mxmxln<<endl;*

*cout<<"Minimumul din maximele pe linii este "<<mnmxln<<endl;*

*maxmin(mnln,m,mxmnln,mnmnln);*

*cout<<"Maximumul din minimele pe linii este "<<mxmnln<<endl;*

*cout<<"Minimumul din minimele pe linii este "<<mnmnln<<endl;*

*cout<<endl<<"Pentru continuare, apasati orice tasta!";*

*getch();*

*cout<<endl<<endl;*

*for (j=1;j<=n;j++) {*

*for (i=1;i<=m;i++) ajut[i]=a[i][j];*

*maxmin(ajut,m,mxcl[j],mncl[j]);*

*cout<<"coloana "<<j<<" : "<<"max="<<mxcl[j]<<" min="<<mncl[j]<<endl;*

*}*

*cout<<endl;*

*maxmin(mxcl,n,mxmxcl,mnmxcl);*

*cout<<"Maximumul din maximele pe coloane este "<<mxmxcl<<endl;*

*cout<<"Minimumul din maximele pe coloane este "<<mnmxcl<<endl;*

*maxmin(mncl,n,mxmncl,mnmncl);*

*cout<<"Maximumul din minimele pe coloane este "<<mxmncl<<endl;*

*cout<<"Minimumul din minimele pe coloane este "<<mnmncl<<endl;*

*getch();*

*}*

* 1. **TEME DE LABORATOR**

1. Să se realizeze subprogramele ***prim***, ***invers***, şi ***suma***, care primesc fiecare. prin intermediul parametrului ***n***, valoarea unui număr întreg lung. Subprogramul ***prim*** returnează valoarea ***1***, dacă numărul ***n*** este prim şi ***0*** în caz contrar; subprgramul ***invers*** returnează “oglinditul” numărului primit prin intermediul parametrului, iar subprogramul ***suma*** returnează suma cifrelor numărului primit ca parametru. Programul principal citeste de la tastatură un număr ***n***, care reprezintă numărul de elemente ale unui vector ***v***, ale cărui elemente se vor citi de la tastatură. Cu ajutorul subprogramelor ***prim***, ***invers*** şi ***suma***, elementele prime ale vectorului ***v*** se vor înlocui cu oglinditele lor, iar cele care nu sunt prime cu suma cifrelor lor. Vectorul nou format se va tipări pe monitor.
2. Cu ajutorul subprogramelor ***prim*** şi ***invers*** de la punctul anterior, să se afişeze pe monitor toate numerele prime de două cifre, care citite invers sunt tot prime.
3. Subprogramul ***aparcifra*** are doi parametri: ***numar*** – un număr întreg lung; ***cifra*** – un număr întreg de o singură cifră. Programul returnează numărul de apariţii ale cifrei ***cifra*** în numărul ***numar***.

Programul principal citeste de la tastatură un număr ***n***, care reprezintă numărul de elemente ale unui vector ***v***, ale cărui elemente se vor citi de la tastatură, şi o cifră – ***c***. Cu ajutorul subprogramului ***aparcifra***, elementele vectorului ***v*** se vor înlocui fiecare cu numărul de apariţii ale cifrei ***c*** în respectivul element. Vectorul nou format se va tipări pe monitor.

1. Utilizând subprogramul ***aparcifra*** de la punctul anterior, să se calculeze câte cifre pare are un număr n, citit de la tastatură.
2. Utilizând subprogramul ***aparcifra*** de la punctul anterior, să se calculeze câte cifre distincte are un număr n, citit de la tastatură
3. Subprogramul ***elimincifra*** are doi parametri: ***numar*** – un număr întreg lung; ***cifra*** – un număr întreg de o singură cifră. Subprogramul returnează numărul primit, din care s-au eliminat toate apariţiile cifrei cifra.

Programul principal citeste de la tastatură un număr ***n***, care reprezintă numărul de elemente ale unui vector ***v***, ale cărui elemente se vor citi de la tastatură. Programul va înlocui fiecare element din vectorul ***v*** cu numărul obţinut prin eliminarea tuturor cifrelor impare din număr; vectorul nou format se va tipări pe monitor.

1. Cu ajutorul subprogramului prim, de la problema 1, să se calculeze cele mai apropiate numere prime ***a*** şi ***b***, cu ***a < n <b*** (***n*** citit de la tastatură)
2. Să se realizeze subprogramul ***ordon***, care primeşte ca parametri:

* un vector ***v*** de numere întregi;
* un număr natural ***n***, reprezentând numărul de elemente ale vectorului ***v***;

Subprogramul returnează doi vectori, ***a*** şi ***b***, conţinând vectorul v ordonat crescător, respectiv descrescător; vectorii ***a***, ***b***, ***v*** ***NU*** sunt variabile globale

1. Să se realizeze subprogramul ***doivect***, care primeşte ca parametri:

* un vector ***v*** de numere întregi;
* un număr natural ***n***, reprezentând numărul de elemente ale vectorului ***v***;
* un vector ***w*** de numere întregi;
* un număr natural ***m***, reprezentând numărul de elemente ale vectorului ***w***.

Subprogramul returnează un vector a, cuprinzând elementele comune din vectorii ***v*** şi ***w.*** Vectorii ***v***, ***w***, şi ***a*** ***NU*** sunt variabile globale