Instrucțiunile limbajului de programare C++

Atribuirea

Operatorii de atribuire: **simplă =**

cu operator ***=, /=, %=, +=, -=, <<=, >>=, &=, ^=, |=***

Operatorul de atribuire, în forma cea mai simplă, se notează prin caracterul "**=**". El se utilizează în expresii de forma:

v=expresie;

Principiul de execuţie:

- se evaluează expresia
- variabilei v i se atribuie valoarea obţinută(dacă este cazul se efectuează conversia respectivă)

În general o expresie de atribuire are forma:

vn=vn-1=vn-2=...=v2=v1=expresie;

Principiul de execuţie:

- se evaluează expresia
- variabilei v1 i se atribuie valoarea obţinută (dacă este cazul se efectuează conversia respectivă)
- conţinutul variabilei v1 i se atribuie variabilei v2 (dacă este cazul se efectuează conversia respectivă)

……

- conţinutul variabilei vn-1 i se atribuie variabilei vn (dacă este cazul se efectuează conversia respectivă)

Pentru atribuire, în afara semnului = se mai poate folosi şi atribuirea cu operator:

operator=

unde operator poate fi unul din operatorii: * / % + -

O atribuire de forma:

variabila operator=expresie este echivalentă cu **variabila =variabila operator expresie**

Exemplu:

s=s+i se mai poate scrie s+=i

p=p*i se mai poate scrie p*=i

Pentru a schimba valoarea unei variabile pot folosi citirea. Daca totusi doresc ca variabila sa primeasca valoarea unei expresii calculate pe parcursul algoritmului, atunci am nevoie de atribuire:

- **Sintaxa**: variabila<- expresie;
- **Efect**: In "cutia" variabilei se memoreaza valoarea expresiei;
- **Exemplu**:
  - *intreg a; a<- 10;*
  - *real b; fie b<- 3.14;*
  - *sir c; fie c<- 'totul e ok'*;

- **Observatie**:
  - noua valoare se memoreaza peste vechea valoare care se va pierde; adica, se inloceste vechea valoare cu cea noua
  - in fiecare din exemple in variabila se memoreaza o valoare de acelasi tip cu variabila; nu putem memora intr-o variabila o valoare de alt tip:
    - secventa *intreg a; a<-3.14; scrie a;* nu este corecta; variabila a este de tip intreg si nu poate memora o valoare reala.
    - secventa *real a; a<- +10;* este corecta deoarece valoarea intreaga 10 este si valoare reala, conform incluziunii matematice.
    - greselile frecvente sunt cele in care valori obtinute in urma unor impartiri sau radicali (valori reale) sunt atribuite unor variabile intregi
  - operatiile cele mai de intalnite la atribuire sunt incrementarea (marirea cu 1 a valorii variabilei) si decrementarea (micsorarea cu 1 a valorii variabilei)
  - **incrementarea**: ***a<- a+1;***
    - asa cum stim , intai se calculeaza valaorea expresiei (cresterea cu 1 a lui a) si apoi se memoreaza in variabila, peste vechea valoare
  - **decrementarea**: ***a<- a-1;***
    - se calculeaza valoarea expresiei (scadereacu 1 a lui a) si apoi se memoreaza in variabila, peste vechea valoare

Exemplu: calculul vitezei, atunci cand stim valoarea distantei parcurse si timpul necesar;

```
#include<iostream>
using namespace std;
int main()
{ int distanta,timp;
  float viteza ;
  cout<<"distanta= "; cin>>distanta;
  cout<<"timp= "; cin>>timp;
  viteza = distanta/timp;
  cout<<"viteza = "<<viteza<<endl;
  return 0;
}
```

Exemplu (interschimbarea a doua variabile folosind auxiliar). Fie doua variabile intregi. Sa se interschimbe valorile variabilelor. Daca variabila a are valoarea 5 si variabila b are valoarea 7, dupa executarea algorimului, variabila a sa aiba valoare 7 si variabila b sa aiba valoare 5.

```
#include<iostream>
using namespace std;
int main()
{ int a,b,aux;
  cout<<"a= "; cin>>a;
  cout<<"b= "; cin>>b;
  aux=a; a=b ; b=aux ;
  cout<<"a= "<<a<<<<"b= "<<b;
  return 0;
}
```

- *observatie*: problema este similara cu urmatoarea problema schimbarii continutului a doua pahare : necesita folosirea unui pahar auxiliar.

Exemplu (interschimbarea valorii a doua variabile fara auxiliar). Avem aceeasi problema dar trebuie rezolvata fara o variabila in plus.

```
#include<iostream>
using namespace std;
int main()
{ int a,b;
  cout<<"a= "; cin>>a;
  cout<<"b= "; cin>>b;
  a=a+b; b=a-b ; a=a-b ;
  cout<<"a= "<<a<<<<"b= "<<b;
  return 0;
}
```

Fara instructiunea de decizie sau de test algorimul ar fi linear. Aceasta instructiune ne ajuta in cazul in care avem o intrebare privind natura datelor folosite, a valorii lor la un moment dat. Practic, trebuie sa punem intrebari la care algoritmul sa raspunda cu adevarat/fals.

Operatori relationali

“>” inseamna mai mare
a>b are valoarea adevarat numai daca valoarea variabilei a este mai mare decat valoarea variabilei b
“<” inseamna mai mic
a<b are valoarea adevarat numai daca valoarea variabilei a este mai mica decat valoarea variabilei b
“>=” inseamna mai mare egal
a>b are valoarea adevarat numai daca valoarea variabilei a este mai mare sau egala cu valoarea variabilei b
“<=” inseamna mai mic sau egal
a>b are valoarea adevarat numai daca valoarea variabilei a este mai mica sau egala cu valoarea variabilei b
” ==” inseamna egal
a==b are valoarea adevarat numai daca variabila a are aceeasi valoare cu variabila b
“!=” inseamna diferit
a!=b are valoarea adevarat numai daca variabila a NU are aceeasi valoare cu variabila b

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

int main()
{ int a=5, b=6,c=0;
  cout<<"a<b "<<(a<b)<<endl;     // se va afisa a<b 1
  cout<<"a>b "<<(a>b)<<endl;     // se va afisa a>b 0
  cout<<"a==b "<<(a==b)<<endl;   // se va afisa a==b 0
  cout<<"a!=b "<<(a!=b)<<endl;   // se va afisa a!=b 1
  cout<<"a==0 "<<(a==0)<<endl;   // se va afisa a==0 0
  cout<<"c==0 "<<(c==0)<<endl;   // se va afisa c==0 1
  cout<<"++a<=b "<<(++a<=b)<<endl; // se va afisa ++a<=b 1
```

```
  cout<<"a>b-- "<<(a>b--)<<endl;       // se va afisa a>b--0
  cout<<"a>b "<<(a>b)<<endl;     // se va afisa a>b 1
  return 0;
}
```

Operatori logici

Operatorii logici (cu care lucreaza si logica matematica) sunt **SI, SAU si NEGARE**:
Operatorul && (SI) este un operator binar, cu doi operanzi cu valoare logica. Rezultatul operatorului SI este adevarat numai daca ambii operanzi au valoarea ADEVARAT.

- "Mie imi place fata bruneta **si** cu ochi albastri!"
- daca fata nu este bruneta sau nu are ochi albastri nici nu ma uit la ea, ceea ce inseamna fals; daca macar una dintre conditii este falsa atunci intreaga expresie este falsa

Operatorul || (SAU) este un operator binar cu doi operanzi cu valoare logica. Rezultatul operatiei SAU este adevarat daca macar unul din operanzi are valoarea ADEVARAT.

- "Mie imi place fata bruneta SAU cu ochi albastri!"
- Fata nu trebuie sa aiba ambele calitati. Daca este bruneta imi place (ADEVARAT). Daca este cu ochi albastri imi place (ADEVARAT).
- Daca macar una din conditii este adevarata atunci intreaga expresie este adevarata

Operatorul de negatie (vom folosi semnul " ! ") schimba valoarea de adevar a expresiei:
!ADEVARAT==FALS
!FALS ==ADEVARAT

Exemple:
x apartine [-5, -2] se scrie **(x>=-5) && (X<=-2)**
x apartine (0, 10) se scrie **(x>0) && (x<10)**

- x nu apartine (-7,9] se scrie, in una din variantele următoare
  (x<=-7) || (x>9)
  !(x>-7) || !(x<=9)
  !((X>-7) && (x<=9))

Observatii:

- prioritatea operatorilor logici: negare, SI, SAU
- in cazul expresiilor logice se pot aplica regulile lui **De Morgan:**
  !(exp1 **si** exp2) = **!**exp1 **sau !**exp2
  !(exp1 **sau** exp2) = **!**exp1 **si !**exp2

Exemple:

```
#include<iostream>
using namespace std;
int main()
{ int a=5,b=6,c=0;
  cout<<"a&&b "<<(a&&b)<<endl;          // se va afisa 1
  cout<<"a&&c "<<(a&&c)<<endl;          // se va afisa 0
  cout<<"-a||c "<<(-a||c)<<endl;        // se va afisa 1
  cout<<"a&&!c "<<(a&&!c)<<endl;        // se va afisa 1
  return 0;
}
```

Instructiunea de decizie (de test sau alternativa)

Sintaxa:
if (exp_logica)
atunci instructiune1;
[altfel instructiune2;]

Efect: Se evalueaza expresia logica; daca valoarea logia ca expresiei este diferită de 0 (ADEVARAT) se executa intructiunea 1; Daca exista sectiunea ALTFEL si valoarea logica a expresiei esteegala cu 0 (FALS) se executa intructiunea 2; in ambele cazuri, dupa executarea intructiunii corespunzatoare si executa ce instructiune urmeaza dupa testul nostru.
Observatie. Un algoritm **trebuie sa fie clar** si sa poata fi inteles intr-un singur fel. Din acest punct de vedere, exista doua moduri de a scrie un algoritm:

- cate o instructiune pe linie si cu marcatori la sfarsitul instructiunilor de test si repetitive (cum cere manualul)
- mai multe instructiuni pe line, separate printr-un marcator; pentru cazul cand un grup de instructiuni trebuie executate impreuna pe unul din cazurile instructiunii de decizie (de exemplu), acestea pot fi incadrate cu acolade. In limbajul C++ cand o secventa de instructiuni trebuie executate una dupa alta, datorita logicii algoritmului, acea secventa o vom incadra in acolade (de exemplu: **{instructiune1; instructiune2; instructiune3; …}**);

Exemple:

- *Sa se afiseze maximul a doua valori citite.*

Pseudocod :

intreg a,b;
citeste a,b;
daca (a>b) **atunci scrie** a
altfel scrie b.

- *Sa se afiseze modulul (matematic) al unui numar intreg.*

```
#include <iostream>
using namespace std;
int main()
{ int x,modul;
  cout<<"x="; cin>>x;
  if (x>=0) modul=x;
    else modul=-x;
  cout<<"modulul="<<modul;
  return 0;
}
```

- *Sa memoram valoarea maxima intr-o variabila si sa o afisam.*

```
#include <iostream>
using namespace std;
int main()
{ float a,b,max;
  cout<<"a="; cin>>a;
  cout<<"b="; cin>>b;
  if (a>b) max=a;
    else max=b;
  cout<<"maximul="<<max;
  return 0;
}
```

- *Sa se citeasca o valoare intreaga si sa se stabileasca daca s-a citit o valoare para sau impara.*

```
#include <iostream>
using namespace std;
int main()
{ int x;
  cout<<"x="; cin>>x;
  if (x%2==0)
     cout<<"par";
   else cout<<"impar";
  return 0;
}
```

- *Se citește o literă mică. Să se verifice dacă este vocală sau consoană.*

```
#include <iostream>
using namespace std;
int main()
{ char x;
  cout<<"x="; cin>>x;
  if (x=='a' || x=='e' || x=='i'|| x=='o' || x=='u')
       cout<<"este vocala";
  else cout<<"este consoana";
  return 0;
}
```

- *Se citește un caracter. Să se afișeze dacă este litera mare, mică sau dacă nu e literă.*

```
#include <iostream>
using namespace std;
int main()
{ char x;
  cout<<"x=";cin>>x;
  if(x>='A'&&x<='Z')
     cout<<"este litera mare";
  else
    if(x>='a'&&x<='z')
       cout<<"este litera mica";
    else
       cout<<"nu este litera";
  return 0;
}
```

Operatori logici

Operatorii logici (cu care lucreaza si logica matematica) sunt SI, SAU si NEGARE:

Operatorul && (SI) este un operator binar, cu doi operanzi cu valoare logica. Rezultatul operatorului SI este adevarat numai daca ambii operanzi au valoarea ADEVARAT.

- "Mie imi place fata bruneta **si** cu ochi albastri!"
- daca fata nu este bruneta sau nu are ochi albastri nici nu ma uit la ea, ceea ce inseamna fals; daca macar una dintre conditii este falsa atunci intreaga expresie este falsa

Operatorul || (SAU) este un operator binar cu doi operanzi cu valoare logica. Rezultatul operatiei SAU este adevarat daca macar unul din operanzi are valoarea ADEVARAT.

- "Mie imi place fata bruneta SAU cu ochi albastri!"
- Fata nu trebuie sa aiba ambele calitati. Daca este bruneta imi place (ADEVARAT). Daca este cu ochi albastri imi place (ADEVARAT).
- Daca macar una din conditii este adevarata atunci intreaga expresie este adevarata

Operatorul de negatie (vom folosi semnul " **!** ") schimba valoarea de adevar a expresiei:

!ADEVARAT==FALS
!FALS ==ADEVARAT

Exemple:

x apartine **[-5, -2]** se **scrie** `(x>=-5) && (X<=-2)`

x apartine **(0, 10)** se scrie `(x>0) && (x<10)`

- x nu apartine (-7,9] se scrie, in una din variantele următoare

```
(x<=-7)   || (x>9)
!(x>-7)   ||  !(x<=9)
!((X>-7) && (x<=9))
```

Observatii:

- prioritatea operatorilor logici: **negare**, **SI, SAU**
- in cazul expresiilor logice se pot aplica regulile lui **De Morgan:**

!(exp1 **si** exp2) = **!**exp1 **sau !**exp2
!(exp1 **sau** exp2) = **!**exp1 **si !**exp2

Exemple:

```
#include<iostream>
using namespace std;
int main()
{ int a=5,b=6,c=0;
  cout<<"a&&b "<<(a&&b)<<endl;          // se va afisa 1
  cout<<"a&&c "<<(a&&c)<<endl;          // se va afisa 0
  cout<<"-a||c "<<(-a||c)<<endl;        // se va afisa 1
  cout<<"a&&!c "<<(a&&!c)<<endl;        // se va afisa 1
  return 0;
}
```

Instructiunea de decizie (de test sau alternativa)

Sintaxa:
if (exp_logica)
atunci instructiune1;
[altfel instructiune2;]

Efect: Se evalueaza expresia logica; daca valoarea logia ca expresiei este diferită de 0 (ADEVARAT) se executa intructiunea 1; Daca exista sectiunea ALTFEL si valoarea logica a expresiei esteegala cu 0 (FALS) se executa intructiunea 2; in ambele cazuri, dupa executarea intructiunii corespunzatoare si executa ce instructiune urmeaza dupa testul nostru.
Observatie. Un algoritm **trebuie sa fie clar** si sa poata fi inteles intr-un singur fel. Din acest punct de vedere, exista doua moduri de a scrie un algoritm:

- cate o instructiune pe linie si cu marcatori la sfarsitul instructiunilor de test si repetitive (cum cere manualul)
- mai multe instructiuni pe line, separate printr-un marcator; pentru cazul cand un grup de instructiuni trebuie executate impreuna pe unul din cazurile instructiunii de decizie (de exemplu), acestea pot fi incadrate cu acolade. In limbajul C++ cand o secventa de instructiuni trebuie executate una dupa alta, datorita logicii algoritmului, acea secventa o vom incadra in acolade (de exemplu: **{instructiune1; instructiune2; instructiune3; …}**);

Exemple:

- *Sa se afiseze maximul a doua valori citite.*

intreg a,b;
citeste a,b;
daca (a>b) **atunci scrie** a
altfel scrie b.

- *Sa se afiseze modulul (matematic) al unui numar intreg.*

```
#include <iostream>
using namespace std;
int main()
{ int x,modul;
  cout<<"x="; cin>>x;
  if (x>=0) modul=x;
     else modul=-x;
  cout<<"modulul="<<modul;
  return 0;
}
```

- *Sa memoram valoarea maxima intr-o variabila si sa o afisam.*

```
#include <iostream>
using namespace std;
int main()
{ float a,b,max;
  cout<<"a="; cin>>a;
  cout<<"b="; cin>>b;
  if (a>b) max=a;
    else max=b;
  cout<<"maximul="<<max;
  return 0;
}
```

- *Sa se citeasca o valoare intreaga si sa se stabileasca daca s-a citit o valoare para sau impara.*

```
#include <iostream>
using namespace std;
int main()
{ int x;
  cout<<"x="; cin>>x;
  if (x%2==0)
      cout<<"par";
    else cout<<"impar";
  return 0;
}
```

- *Se citește o literă mică. Să se verifice dacă este vocală sau consoană.*

```
#include <iostream>
using namespace std;
int main()
{ char x;
  cout<<"x="; cin>>x;
  if (x=='a' || x=='e' || x=='i'|| x=='o' || x=='u')
        cout<<"este vocala";
  else cout<<"este consoana";
  return 0;
}
```

- *Se citește un caracter. Să se afișeze dacă este litera mare, mică sau dacă nu e literă.*

```
#include <iostream>
using namespace std;
int main()
{ char x;
  cout<<"x=";cin>>x;
  if(x>='A'&&x<='Z')
      cout<<"este litera mare";
  else
     if(x>='a'&&x<='z')
        cout<<"este litera mica";
     else
        cout<<"nu este litera";
  return 0;
}
```

Structuri repetitive

Exista trei instructiuni (structuri) repetitive folosite in toate limbajele:

- instructiunea repetitiva cu test initial CAT TIMP (WHILE) (se foloseste cand numarul de repetitii este nedefinit)
- instructiunea repetitiva cu test final REPETA-PANA CAND/ EXECUTA CAT TIMP (DO WHILE) (se foloseste cand numarul de repetitii este nedefinit)
- instructiunea repetitiva cu un numar cunoscut de pasi PENTRU (FOR) (se foloseste cand numarul de repetitii este cunoscut)

Pentru a prelucra un sir de numere trebuie sa stim fie numarul de valori in discutie (sir cu numar cunoscut de valori), fie sa avem o valoare care sa marcheze sfarsitul sirului (de obicei – zero).

Instructiunea WHILE

Sintaxa:

while (expresie_logica)

instructiunea;

Efect:

- se stabileste valoarea de adevar a expresiei logice
- daca valoarea expresiei logice este ADEVARAT atunci se executa instructiunea si se reia evaluarea expresiei logice
- daca valoarea expresiei logice este FALS atunci se continua cu instructiunea de dupa while

Observatii:

- while este repetitiva conditionata anterior deoarece intai se evaluaeza conditia si apoi se executa instructiunea
- Practic, succesiunea de etape este
- **exp logica**, *instructiune*, **exp logica**, *instructiune*, **exp logica**, *instructiune*,…**exp logica;**

  succesiunea se incheie cu exp logica in momentul in care valoarea expresiei este FALS.
- daca pe cazul ADEVARAT trebuie sa scriem mai multe instructiuni, acestea vor fi grupate cu acolade

Exemplul 1

- *Fie a si b doua valori naturale. Se se simuleze inmultirea a*b prin adunari repetate.*
  - P=a*b=a+a+….+a de b ori
  - Expresia de mai sus spune ca **a** trebuie adunat la **P** de **b** ori; adica, la fiecare adunare la **P** a lui **a** putem sa scadem un 1 din b (*decrementam*), pentru a pastra numarul de adunari ce mai trebuie efectuat; cand**b** va fi zero, se va adunat **a** de **b** ori

```
#include <iostream>
using namespace std;
int main()
{ int a,b,p;
  cout<<"a=";cin>>a;
  cout<<"b=";cin>>b;
  p=0;
  while (b>0)
  { p=p+a;
    b=b-1;
  }
  cout<<p;
  return 0;
}
```

Observatii:

- pentru fiecare adunare a lui a la p, scadem un 1 din b
- aceste doua operatii trebuie executate pentru fiecare caz in care b este pozitiv; de aceea au fost grupate cu acolade
- este algoritmul corect? Sa verificam cazurile cu zero:
  - daca a este zero, atunci la P se aduna zero de b ori; practic P ramane la valoarea zero. CORECT!
  - daca b este zero, conditia din CAT TIMP este falsa si atunci nu se executa instructiunile; in consecinta se afiseaza direct valoarea lui P,adica zero; CORECT!

Exemplul 2

- *Fie **a** si **b** doua valori naturale. Sa se simuleze impartirea lui a la b prin scaderi repetate si sa se afiseze catul si restul.*
  - vom scadea din a valoarea lui b de mai multe ori, numarand intr-o variabila **contor** de cate ori am facut scaderea (practic variabila *contor* va fi catul impartirii)
  - operatia se va repeta cat timp din **a** se mai poate scadea un **b**, adica **cat timp** a mai mare decat b
  - Ce reprezinta valoarea ramasa in **a**? O valoare mai mica decat **b**? Evident, restul impartirii.

```
#include <iostream>
using namespace std;
int main()
{ int a,b,contor;
  cout<<"a=";cin>>a;
  cout<<"b=";cin>>b;
  contor=0;
  if (b==0)
      cout<<"nu se poate realiza impartirea la 0";
  else
   { while (a>=b)
     { a=a-b;
       contor=contor+1;
     }
     cout<<"catul este = "<<contor<<"restul este = "<<a;
   }
  return 0;
}
```

Observatii:

- am pus intre acolade secventa in care numaram de cate ori am scazut pe b din a; erau doua instructiuni; de asemenea pe varianta “else” este instructiunea while si o afisare, care de asemenea au fost puse intre acolade
- este corect algoritmul? Daca incercati cazurile normale, va merge. Sa verificam cazurile speciale:
  - daca **a** este mai mic decat **b** atunci cat timp nu va functiona si se va afisa direct **contor** (zero) si **a**(restul); CORECT!
  - daca **a** este zero, atunci nu se executa *cat timp* si se afiseaza **contor** si **a** (0 si 0) CORECT!
  - daca b este zero, se afiseaza mesajul de eroare si algoritmul se incheie CORECT!

Exemplul 3.

- *Sa se calculeze cel mai mare divizor comun a doua valori a si b naturale.*
  - asa cum spune definitia, cel mai mare divizor comun trebuie sa fie o valoare, care sa divida atat pe a cat si pe b; aceasta valoare poate fi in cel mai bun caz a sau b

```
#include <iostream>
using namespace std;
int main()
{ int a,b,cmmdc;
  cout<<"a=";cin>>a;
  cout<<"b=";cin>>b;
  cmmdc=a;
  while (a%cmmdc+b%cmmdc>0)
     cmmdc=cmmdc-1;
  cout<<"cmmdc="<<cmmdc;
  return 0;
}
```

Observatii:

- expresia *a%cmmdc +b%cmmdc !=0* este nula doar daca atat a%cmmdc cat si b%cmmdc sunt nule, adica cmmdc divide simultan a si b; atat timp cat aceasta conditie nu se realizeaza scadem cmmdc-ul
- ultima valoare panaa la care se poate scadea este 1, divizorul tuturor numerelor; in acest caz a si b se numesc numere prime intre ele.

Exemplul 4

- ***Ghiceste-mi numarul!*** *Eu imi aleg un numar intre 1 si 100. In cati pasi il poti ghici? La fiecare incercare raspund cu SUCCES, MIC sau MARE.*
  - prima varianta este sa intrebam aleator; cam multi pasi!
  - sa parcugem valorile de la 1 la 100; cam multi pasi!
  - sa gandim! vom testa intodeauna mijlocul intervalului; in acest fel, la fiecare intrebare, elimin jumatate din cazuri.

```
#include <iostream>
using namespace std;
int main()
{ int x,m1,m2,mij,pasi=0;
  cout<<"x=";cin>>x;
  m1=1; m2=100;
  mij=50;
  while (mij!=x)
    { if (x<mij) m2=mij;
         else m1=mij;
      mij=(m1+m2)/2;
      pasi=pasi+1;
    }
  cout<<"nr pasi = "<<pasi;
  return 0;
}
```

Observatie:

- dupa fiecare test fara succes intervalul in care cautam se ajusteaza la stanga (daca **mij** este mai mic decat**x**) sau la dreapta (daca **mij** este mai mare decat **x**)

Exemplul 5.

- *Fie N un numar natural. Sa se calculeze suma cifrelor lui N.*
  - vom initializa o variabila **suma** cu zero
  - trebuie ca delimitam pe rand cifrele care formeaza numarul N; putem determina rapid ultima cifra: n%10; daca o stergem (n=n/10) putem deternima penultima cifra; s.a.m.d.
  - cand ne oprim? … cand N nu mai are cifre; deci , cand N este zero.

```
#include <iostream>
using namespace std;
int main()
{ int n,s=0;
  cout<<"n=";cin>>n;
  while (n!=0)
  { s=s+n%10;
    n=n/10;
  }
  cout<<"suma cifrelor="<<s;
  return 0;
}
```

Observatie:

- instructiunea *suma<-suma+n%10* creste **suma** cifrelor deja obtinute cu valoarea ultimei cifre a lui **n**.
- pentru cazul in care n este nul, cat timp nu se mai executa si se afiseza suma cu valoarea 0; Suma cifrelor lui 0 este 0. CORECT!

Instructiunea DO … WHILE

Sintaxa:

```
do{   instructiunea;

    }while (expr_logica);
```

Efect:

1. se executa instructiunea
2. se stabileste valoarea de adevar a expresiei logice
3. daca valoarea conditiei este ADEVARAT atunci se revine la executarea instructiunii.
4. daca valoarea conditiei este FALSA atunci se continua cu instructiunea de dupa do {… } while

Observatii:

- instructiunea do {… } while este o instructiune repetitiva conditionata posterior (sau cu test final)
- intai executa instructiunea de repetat si apoi verifica necesitatea repetarii; instrutiunea se executa macar o data
- secventa de operatii este: instructiune**,** expr_logica, instructiune, … , expr_logica, instructiune, expr_logica

Exemplu.

- *Fie N un numar natural. Sa se calculeze suma cifrelor lui N.*
  - vom initializa o variabila **suma** cu zero
  - trebuie ca delimitam pe rand cifrele care formeaza numarul N; putem determina rapid ultima cifra: n%10; daca o stergem (n=n/10) putem deternima penultima cifra; s.a.m.d.
  - cand ne oprim? … cand N nu mai are cifre; deci , cand N este zero.

```
#include <iostream>
using namespace std;
int main()
{ int n,s=0;
  cout<<"n=";cin>>n;
  do { s=s+n%10;
       n=n/10;
     } while (n!=0);
  cout<<"suma cifrelor="<<s;
  return 0;
}
```

Aplicatii:

- sa se afiseze cifrele numarului
- sa se numere cate cifre are N
- sa se stabileasca de cate ori apare o cifra anume
- sa se determine cifra maxima/minima din numar
- sa se creeze oglinditul numarului N (N=1234 => 4321)
- sa se stabileasca daca N estre palindrom (egal cu oglinditul sau). Ex: 12321, 11, 121, …

- *Se citesc numere reale pozitive de la tastatură, până la întâlnirea numărului 0. Să se afişeze suma numerelor citite.*

varianta WHILE

```
#include <iostream>
using namespace std;
int main()
{ float S=0,x;
  cout<<"x="; cin>>x;
  while (x!=0)
  { S+=x;
    cout<<"x="; cin>>x;
  }
  cout<<"S="<<S;
  return 0;
}
```

varianta DO WHILE

```
#include <iostream>
using namespace std;
int main()
{ float S=0,x;
  do{ cout<<"x="; cin>>x;
      S+=x;
    }while (x!=0);
  cout<<"S="<<S;
  return 0;
}
```

Instructiunea FOR

Sintaxa:
for(contor=exp_ini; contor!=exp_fin; modificare_contor)
instructiune;

Efect: pentru fiecare valoare a contorului intre expresia initiala si expresia finala se executa instructiunea;
Exemplu:

```
for(i=1;i<=10;i++)
   cout<< ” Nu ma prinzi!”;
```

- pentru fiecare valoarea a variabile i, de la 1 la 10, se afiseaza ” Nu ma prinzi!”
- de 10 ori se afiseaza ” Nu ma prinzi!”
- daca secventa ce trebuie repetata contine mai multe instructiuni, acestea se vor grupa cu acolade

Observatii:

- practic, pentru fiecare valoare a lui i, intai se testeaza daca nu s-a depasit valoarea finala 10 si apoi se executa instructiunea;
- algoritmic, propozitia de mai sus este :

```
i=1;
while(i<=10)
 {   cout<< ” Nu ma prinzi!”;
     i=i+1;
}
```

- practic , secventa de mai sus ne explica faptul ca instructiunea **FOR** este echivalenta cu instructiuniw **WHILE**.
- instructiunea este “ceruta” daca descrierea algorimului spune “de la valoarea X la valoarea Y”, “pentru primele X valori”, “de X ori”, …

Exemplul 1.

- *Sa se afiseze numerele pare pana la o valaore N, naturala.*

```
#include <iostream>
using namespace std;
int main()
{ int n,i;
  cin>>n;
  for(i=0;i<=n;i++)
     if (i%2==0)
          cout<<i<<"  ";
  return 0;
}
```

Observatii:

- algoritmul ia fiecare valoare intre 0 si **n** si o testeaza daca este para (restul impartirii lui **i** la 2 sa fie nul : *i%2==0*)
- se efectueaza n pasi din care jumatate sunt gresiti; trebuie o varianta mai buna

Exemplul 2.

- *Aceeasi problema dar incercam sa mergem din doi in doi*

```
#include <iostream>
using namespace std;
int main()
{ int n,i;
  cin>>n;
  for(i=0;i<=n;i=i+2)
     if (i%2==0)
          cout<<i<<"  ";
  return 0;
}
```

- *Aceeasi problema dar incercam sa mergem pana la n/2*

```
#include <iostream>
using namespace std;
int main()
{ int n,i;
  cin>>n;
  for(i=0;i<=n/2;i++)
     if (i%2==0)
          cout<<i*2<<"  ";
  return 0;
}
```

Observatii:

- in primul caz, 2-ul de dupa n (*i<-0,n,2*) stabileste cresterea lui i cu 2 si nu cu 1 asa cum este implicit
- in al doilea caz, ne folosim de faltul ca valorile cautate sunt pare, divizibile cu 2;

Exemplul 3.

- Sa se calculeze suma primelor N numere naturale.
  - evident, stim formula n*(n+1)/2 dar sa incercam un algoritm;
  - va trebui sa adunam, la o **suma** , toate valoarile de la 1 la n

```
#include <iostream>
using namespace std;
int main()
{ int n,i,suma=0;
  cin>>n;
  for(i=0;i<=n;i++)
     suma=suma+i;
  cout<<suma;
  return 0;
}
```

- *Se citeste un sir de N valori intregi. Sa se determine cea mai mare valoare citita (valoarea maxima dintr-un sir).*

```
#include <iostream>
using namespace std;
int main()
{ int n,i,max,val;
```

```
  cout<<”n=”; cin>>n;
  cout<<”valoare:”;cin>>val;
  max=val;
  for(i=2;i<=n;i++)
     { cout<<” valoare:”;cin>>val;
       if (val>max)
         max=val;
     }
  cout<<max;
  return 0;
}
```

- *Se citesc n numere întregi. Să se calculeze suma numerelor pozitive şi câte numere impare sunt.*

```
varianta FOR
#include <iostream>
using namespace std;
int main()
{ int n,i,S=0,contor=0,x;
  cout<<"n="; cin>>n;
  for(i=1;i<=n;i++)
   { cout<<"x="; cin>>x;
     if (x>0) S+=x;
     if (x%2) contor++;
   }
  cout<<"S="<<S<<endl;
  cout<<"numarul numerelor impare="<<contor;
  return 0;
}
varianta WHILE
#include <iostream>
using namespace std;
int main()
{ int n,i,S=0,contor=0,x;
  cout<<"n="; cin>>n;
  i=1;
  while(i<=n)
  { cout<<"x="; cin>>x;
    if (x>0) S+=x;
    if (x%2) contor++;
    i++;
  }
  cout<<"S="<<S<<endl;
  cout<<"numarul numerelor impare="<<contor;
  return 0;
}
varianta DO WHILE
#include <iostream>
```

```
using namespace std;
int main()
{ int n,i,S=0,contor=0,x;
  cout<<"n="; cin>>n;
  i=1;
  do{ cout<<"x="; cin>>x;
      if (x>0) S+=x;
      if (x%2) contor++;
      i++;
    }while(i<=n);
  cout<<"S="<<S<<endl;
  cout<<"numarul numerelor impare="<<contor;
  return 0;
}
```

- *Se citeste un numãr natural n si un numãr real x. Se cere sã se calculeze suma: $S=1+x+x^2+x^3+x^4+...+x^n$*

varianta cu calculul puterilor lui x cu functia pow

```
#include<iostream>
#include<cmath>
using namespace std;
int main()
{ int i,n;
  float x,s=0;
  cout<<"n="; cin>>n;
  cout<<"x="; cin>>x;
  for(i=0;i<=n;i++)
     s=s+pow(x,i);
  cout<<"suma este:"<<s;
  return 0;
}
```

varianta cu calculul puterilor lui x pe parcurs

```
#include<iostream>
using namespace std;
int main()
{ int i,n;
  float x,s=0,px=1; //px calculeaza puterile lui x
  cout<<"n="; cin>>n;
  cout<<"x="; cin>>x;
  for(i=0;i<=n;i++)
   {  s=s+px;
      px=px*x;
   }
  cout<<"suma este:"<<s;
  return 0;
}
```

- *Să se afişeze numerele mai mici decât n care sunt pătrate perfecte.*

```
#include<iostream>
#include<cmath>
using namespace std;
int main()
{
  int x,n;
  cout<<"n="; cin>>n;
  for(x=1;x<=n;x++)
    if (sqrt(x)==(int)sqrt(x))
       cout<<x<<" ";
  return 0;
}
```

Aplicatii:

- Se citeste un număr natural n. Se cere să se afişeze numerele de la n la 1 în ordine descrescătoare.
- Se citeşte un număr natural n. Se cere să se afişeze numerele de la n la 1 în ordine descrescătoare din 2 in 2.
- Să se afişeze cuburile perfecte mai mici decât un număr n citit de la tastatură.
- Se citeşte un număr natural n. Se cere să se afişeze produsul primelor n numere naturale.
- Se citeşte un număr natural n. Se cere să se afişeze suma primelor n numere naturale.
- Se citeşte un număr natural n şi apoi n numere. Se cere să se calculeze suma numerelor impare şi pozitive citite.
- Se citeşte un număr natural n. Se cere să se afişeze suma pătratelor primelor n numere naturale.
- Se citeşte un număr natural n. Se cere să se calculeze suma: S=1+1/2+1/3+1/4+...+1/n
- Se citeşte un număr natural n şi un număr real x. Se cere să se calculeze suma: $S=1-x+x^2-x^3+x^4-...+(-1)^n x^n$
- Se citeşte un număr natural n. Se cere să se calculeze suma factorialelor primelor n numere naturale:

  $$S=1+1*2+1*2*3+1*2*3*4+\ldots+1*2*3*\ldots*n$$
- Se citesc n perechi de numere naturale. Să se afişeze câte prechi au proprietatea că ambele numere din prereche sunt pare.
  Exemplu: pt n=3 şi perechile de numere (12 13), (12 14) (6 7) va afişa 1 deoarece perechea 12 14 îndeplineşte coditiile cerute