Laboratorul 6. INSTRUCȚIUNI (2)

Introducere

Prezentul laborator continuă prezentarea instrucțiunilor specifice limbajelor C/C++.

Sunt prezentate instrucțiunile repetitive care permit executarea în mod repetat a unor secvențe de program atât timp cât o condiție (expresie) este logic adevărată (while, do-while, for).

Pentru a forța întreruperea execuției structurilor secvențiale, alternative sau repetitive, se pot folosi instrucțiuni de salt (break, continue, return, goto) sau funcția exit().

Se poate întrerupe execuția unui bloc de instrucțiuni, a unei funcții sau chiar a programului în totalitatea sa.

6.1. Instrucțiuni de ciclare (repetitive)

6.1.1.

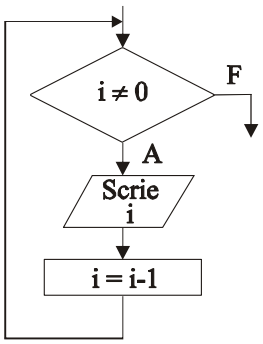
Sintaxa instrucțiunii este:

while (expresie)
instrucțiune;

unde expresie e orice expresie logică, iar instrucțiune, o instrucțiune simplă sau bloc de instrucțiuni.

La execuție, se testează valoarea expresiei. Dacă este logic adevărată (nenulă), se execută instrucțiune. Se evaluează expresia din nou şi dacă este în continuare adevărată se reia execuția instrucțiunii. Se repetă această succesiune de operații atât timp cât expresie rămâne adevărată. În momentul în care expresie a devenit falsă (zero), se întrerupe execuția instrucțiunii while şi se trece la instrucțiunea imediat următoare din program.

```
int i;
i = 5;
while (i)                 // se testează valoarea lui i; inițial are
{                         // valoarea 5, deci se va executa blocul de
                          // instrucțiuni; i este afişat şi decrementat;
   printf("\ni=d", i );   // se repetă afişarea şi decrementarea lui i
   i--;                   // până când acesta ia valoarea zero;
                          // în acel moment se întrerupe execuția
}                         // instrucțiunii while şi se trece la
 …                        // următoarea linie din program.
```



De obicei valoarea expresiei care constituie condiția de ciclare este modificată în interiorul secvenței care se repetă. Dacă valoarea acesteia nu este modificată, există riscul de repetare fără sfârşit a instrucțiunii, evoluția programului fiind blocată. Există instrucțiuni care pot forța ieşirea dintr-o instrucțiune repetitivă. Acestea vor fi prezentate în paragrafele următoare.

În exemplul următor se implementează, folosind instrucțiunea while, următoarea aplicație:

Se calculează media aritmetică a N valori citite de la tastatură. Valoarea N este, de asemenea citită de la tastatură.

```
#include <stdio.h>

void main()
{
   int i, N;                  // variabila i va fi folosită ca şi contor pentru numărarea
                              // valorilor citite; ea va controla evoluția secvenței repetitive
   float val;                 // variabila val se foloseşte pentru citirea valorilor de la tastatură
   float rez;                 // variabila rez este folosită pentru calculul mediei aritmetice
   printf("Introdu numărul de valori:");
   scanf("%d", &N);

   i = 1;                     // se fac inițializări pentru variabilele utilizate în secvența
   rez = 0;                   // repetitivă

   while (i <= N)             // condiția de execuție a ciclului - compară valorile i şi N
      {
         printf("Introdu o valoare:");
         scanf("%f", &val);
         rez = rez + val;
         i++;                 // modificarea valorii variabilei de control a evoluției ciclului
      }
   rez = rez/N;
   printf("\nMedia aritmetica a valorilor este: %10.2f\n", rez);
}
```

Observații

- Dacă în momentul de început al instrucțiunii while condiția nu este îndeplinită se trece direct la instrucțiunea imediat următoare acesteia.
  În exemplul anterior, dacă N primeşte valoare <1, instrucțiunea while nu va fi executată. Se trece la instrucțiunea rez = rez/N. Dacă N=0, se ajunge la o situație care produce eroare fatală, şi anume împărțirea la zero.

Pentru un ciclu care se repetă până la apăsarea tastei 'N', se poate scrie:

```
char ch ;
...
while( ch != 'N')
   {      ...
      ch=getch();
   }
```

sau:

```
char ch ;
...
while( ch=getch() != 'N')
   {
...
   }
```

Instrucțiunea care se repetă poate fi instrucțiunea vidă, ca în exemplul următor:

```
char ch ;
...
while( ch=getch() != 'X') ;
 ...
```

Efectul acestei instrucțiuni este că, pentru ca programul să poată continua, trebuie apăsată tasta 'X'.

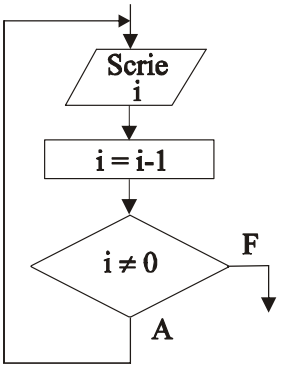
6.1.2. Instrucțiunea de ciclare cu test final (do-while)

Sintaxa generală a instrucțiunii este:

```
do
        instructiune;
while (expresie);
```

Instrucțiunea instructiune se execută atât timp cât expresie este nenulă. Evaluarea expresiei expresie se face după executarea instrucțiunii instructiune.

```
.........
i=5;
do
{
   printf("\ni=%d", i );
   i--;
}                       // expresia condițională se evaluează după
while (i);              // ce blocul de instrucțiuni a fost executat o
                        // dată; reluarea execuției lui se face numai
                        // dacă i ≠0
```



Se reia aplicația care calculează media aritmetică a N valori citite de la tastatură folosind pentru implementare instrucțiunea do-while.

```
#include <stdio.h>

void main()
{
   int i, N;            // variabila i va fi folosită ca şi contor pentru numărarea valorilor citite;
                        // ea va controla evoluția secvenței repetitive
   float val;           // variabila val se foloseşte pentru citirea valorilor de la tastatură
   float rez;           // variabila rez este folosită pentru calculul mediei aritmetice
   printf("Introdu numarul de valori:");
   scanf("%d", &N);

   i = 1;                       // se fac inițializări pentru variabilele utilizate în secvența
   rez = 0;                     // repetitivă

   do
      {  printf("Introdu o valoare:");
         scanf("%f",&val);
```

```
        rez = rez + val;
        i++;                    // modificarea valorii variabilei de control a evoluției ciclului
      }
   while (i<=N);                // condiția de execuție a ciclului - compară valorile i şi N

   rez = rez/N;
   printf("\nMedia aritmetica a valorilor este: %10.2f\n", rez);
}
```

Observații

- Blocul de instrucțiuni se execută cel puțin odată, indiferent de valorile variabilelor i şi N
- Dacă N primeşte valoare <1, instrucțiunea do-while va fi întreruptă, ca şi în cazul folosirii instrucțiunii while. Se trece la instrucțiunea rez = rez/N. Dacă N=0, se ajunge la o situație care produce eroare fatală, şi anume împărțirea la zero.
- În cazul celor două versiuni care folosesc instrucțiunea while, respectiv do-while, diferențele de cod nu sunt majore, problemele care pot apărea sunt similare. În funcție de aplicație însă, pot apărea situații în care una dintre instrucțiuni se pretează a fi implementată mai eficient.

```
do
{  printf("Introdu numarul de valori:");
   scanf("%d", &N);
}
while (N<1);
```

Observații

- Introducerea secvenței de verificare a valorii variabilei N poate evita situația ce generează eroare prin împărțirea la zero.
- În situația de față este de preferat să se folosească instrucțiunea do-while, având în vedere faptul că întâi trebuie introdusă valoare pentru N şi apoi se face verificarea valorii ei.

6.1.3. Instrucțiunea de ciclare cu contor (for)

Sintaxa generală a instrucțiunii for este:

for(expr1; expr2; expr3)
instrucțiune;

Instrucțiunea for se execută în următorii paşi:

- *expr1* se evaluează o singură dată, la intrarea în bucla for;
- *expr2* se evaluează înaintea fiecărei iterații şi reprezintă condiția de execuție a instrucțiunii; valoarea logic falsă a ei provoacă ieşirea din ciclu;
- *expr3* se evaluează la sfârşitul fiecărei iterații, pentru actualizarea parametrilor ciclului.

Instrucțiunea for oferă posibilitatea scrierii unui cod mai compact şi mai clar.

Observații

- Instrucțiunea for este o instrucțiune de ciclare cu test inițial, fiind echivalentă cu:

```
expr1;
while(expr2)
{       instructiune;
        expr3;
}
```

Exemplu de folosire a instrucțiunii for:

```
for( i = 5 ; i ; i-- )          // instrucțiunea înglobează inițializarea lui i, modificarea
                                // valorii lui i, testarea valorii lui i
   printf(”\ni=%d”, i );
```

Aplicația care calculează media aritmetică a N valori citite de la tastatură poate fi implementată cu folosirea instrucțiunii for, după cum urmează:

```
#include <stdio.h>

void main()
{
   int i, N;      // variabila i va fi folosită ca şi contor pentru numărarea valorilor citite; ea
                  // va controla evoluția secvenței repetitive
   float val;     // variabila val se foloseşte pentru citirea valorilor de la tastatură
   float rez;     // variabila rez este folosită pentru calculul mediei aritmetice
   printf("Introdu numarul de valori:");
   scanf("%d", &N);

   for (i=1, rez=0; i<=N ; i++)          //  se fac inițializări pentru variabilele i şi rez, se
                                         // verifică condiția de execuție a ciclului, se
      {                                  // modifică valoarea variabilei de control i
         printf("Introdu o valoare:");
         scanf("%f", &val);
         rez = rez + val;
      }
   rez = rez/N;
   printf("\nMedia aritmetica a valorilor este: %10.2f\n", rez);
}
```

Se observă că prin folosirea instrucțiunii for codul devine mai compact.

Observații

- Expresia de inițializare folosită inițializează ambele variabile, i şi rez.
- Din sintaxa instrucțiunii for poate lipsi oricare dintre expresii, dar semnul de punctuație punct şi virgulă (;) trebuie să fie folosit. De exemplu, se poate folosi o scriere similară cu folosirea instrucțiunii while:

```
i = 1;
rez = 0
```

```
for(  ; i<=N ; )
{       printf("Introdu o valoare:");
        scanf("%f", &val);
        rez = rez + val;
        i++;
}
```

Calcul lui N factorialul (N!) se poate face astfel:

```
int N, i, fact;
 ...
for ( i=N, fact=1 ; i>0 ; i--)
        fact *= i;
```

sau:

```
int N, i, fact;
// ...
for ( i=N, fact=1 ; i>0 ; fact *= i, i-- ) ;
```

Observații

- Se observă că a doua variantă ne oferă o scriere compactă, expresia de calcul a variabilei *fact* fiind inclusă în expr3. Instrucțiunea care trebuie executată repetitiv a devenit expresia vidă.

În sintaxa instrucțiunii for poate lipsi expr2, cea care descrie condiția de ieşire din ciclu. Aceasta poate duce la o ciclare fără sfârşit. Ca şi cazul instrucțiunilor while, do-while se poate forța ieşirea din ciclu cu instrucțiuni de salt.

```
for ( ;  ; )        // nu este precizată expresia care condiționează execuția instrucțiunii
{
   printf("\nSe executa un ciclu infinit !");
}
```

6.2. Instrucțiuni de salt

6.2.1. Instrucțiunea break

Instrucțiunea are forma:

break;

Instrucțiunea break se foloseşte:

- în interiorul instrucțiunii switch pentru a întrerupe execuția acesteia după executarea unei secvențe asociată cu un selector case (vezi paragraful 5.4.2).
- în instrucțiunile de ciclare (while, do-while, for) determinând ieşirea forțată din acestea.

În continuare este prezentat un exemplu de folosire a instrucțiunii break:

```
#define N 100
int i;
for( i=1 ; i<=N ; i++)
```

```
{
 if (!(i%7))
      {
       printf("\ni=%d - divizibil cu 7",i);
       getch();
       break;          // datorita lui folosirii instrucţiunii break ciclul for se va întrerupe când
                       // i devine divizibil cu 7 şi nu când i ajunge la valoarea 100; ca urmare,
                       // o sa se afişeze doar primele 7 valori si se iese din for;
      }
 printf("\ni=%d",i);
}
```

Instrucţiunea break nu poate fi inclusă într-o structură secvenţială sau structură alternativă realizată cu instrucţiunea if.

```
int a, b;
printf("\na=");
scanf("%d", &a);
printf("\nb=");
scanf("%d", &b);
if(a>b)
   {  printf("\n a>b");
      break;                  // eroare, break nu este inclus nici în instrucţiunea switch,
                              // nici în instrucţiune repetitivă
   }
// ....
```

În mod frecvent, instrucţiunea break se foloseşte în instrucţiuni repetitive fără condiţie de finalizare a acestora. De exemplu:

```
char ch;
//...
for ( ; ; )          // nu este precizată expresia care condiţionează ieşirea din ciclul for
{
 //  ...
  if ((ch=getch()) == 'X')     // la tastarea caracterului 'X' se forţează ieşirea din ciclul
      break;                   // for, prin instrucţiunea break
}
```

Dacă o secvenţă conţine mai multe instrucţiuni repetitive imbricate şi una dintre ele conţine instrucţiunea break, se va întrerupe doar ciclul care conţine break.

```
char ch;
int a, b;
//...
while (a<b)
{
//...
   for (  ;  ;  )   // nu este precizată expresia care condiţionează ieşirea din ciclul for
   {
     // ...
```

```
      if ((ch=getch()) == 'X')  // la tastarea caracterului 'X' se forțează ieşirea din ciclul
         break;                 // for, prin instrucțiunea break, dar se continuă while; la următoarea
   }                            // iterație a ciclului while se va lansa din nou instrucțiunea for care va
//...                           // putea fi întreruptă prin tastarea caracterului 'X'
}
```

6.2.2. Instrucțiunea continue

Instrucțiunea are forma:

continue;

Se foloseşte numai în instrucțiunile repetitive şi are ca efect trecerea la următoarea iterație într-un ciclu while, do-while sau for.

Nu are ca efect întreruperea instrucțiunii repetitive, ci vor fi ignorate instrucțiunile ce îi urmează în interiorul ciclului. Acesta se va continua cu următoarea iterație.

Exemplu de folosire a instrucțiunii continue:

```
int i;
for( i=1 ; i<=100 ; i++)
   {
    if (!(i%7))
        { printf("\ni=%d - divizibil cu 7", i);
          getch();
          continue;      // datorita lui continue, se va ignora linia următoare si se trece la
                         // următoarea iterație a lui for, ca urmare, se vor afişa toate
                         // valorile, de la 1 la 100, pentru cele divizibile cu 7 afişarea fiind
                         // diferită față de celelalte valori
        }
    printf("\ni=%d",i);
   }
```

Ca şi în cazul instrucțiunii break, nici instrucțiunea continue nu poate fi folosită în structuri secvențiale sau alternative.

Dacă o secvență conține mai multe instrucțiuni repetitive imbricate şi una dintre ele conține instrucțiunea continue, efectul va fi doar pentru ciclul care conține continue.

6.2.3. Instrucțiunea return

Formele admise pentru instrucțiunea return sunt:

return;

sau

return (expresie);

sau

return expresie;

Efectul instrucțiunii este de revenire dintr-o funcție, adică de a trece controlul la funcția care a apelat funcția ce conține instrucțiunea return, fără transmiterea unei valori în prima formă şi cu transmiterea valorii expresie în celelalte.

Tipul de dată al valorii returnată trebuie să fie compatibil cu cel al funcției.

```
int functie(void)        // antetul funcției precizează tipul valorii întoarse
{
   int a;
...
   return a;             // tipul expresiei returnate e identic cu cel al funcției
}
```

Dacă tipul expresiei returnate e nu identic cu cel al funcției, se face o conversie a tipului expresiei returnate către tipul funcției, similar cu conversiile din operațiile de atribuire. Aceste conversii pot produce trunchieri ale valorii, rezultatul fiind necontrolat.

```
int functie(void)        // antetul funcției precizează tipul valorii întoarse, int
{
   float a;
...
   return a;      // tipul expresiei returnate e float, iar al funcției este int; se face conversie
                  // cu trunchiere a valorii returnate ceea ce poate produce rezultate necontrolate
}
```

În funcțiile care conțin secvențe alternative se poate folosi instrucțiunea return de mai multe ori, dar la execuție se va executa doar cea selectată.

```
int functie(void)        // antetul funcției precizează tipul valorii întoarse
{
   int a, b;
...
   if (a>b)              // se selectează una dintre valorile care va fi returnată
      return a;
   else
      return b;
}
```

Dacă o funcție este void, instrucțiunea return nu e însoțită de valoare, sau chiar poate să lipsească.

```
void functie(void)       // antetul funcției lipsa valorii întoarse, este funcție void
{
...
   return;               // funcția nu returnează valoare
}
```

```
void functie(void)       // antetul funcției indică lipsa valorii întoarse, este funcție void
{
   int a, b;
   ...
   if (a>b)              // se verifică o condiție care poate întrerupe execuția funcției
      return;
   ….
}
```

Toate instrucțiunile care se află într-o funcție după instrucțiunea return vor fi ignorate.

6.2.4. Funcția exit()

Deşi nu este o instrucțiune de control al programului, ci este o funcție definită în fişierele header standard stdlib.h şi process.h.

Ea poate fi folosită pentru a încheia execuția programului din orice punct al său.

Spre deosebire de instrucțiunile break sau return care întrerup doar execuția unui modul din program, funcția exit() întrerupe execuția întregului program şi revenirea în sistemul de operare.

Prototipul funcției este:

void exit (int status);

Valoarea parametrului, status, reprezintă un cod ce poate fi interpretat de sistemul de operare. Valoarea 0 este interpretată ca terminare normală a programului, valorile diferite de zero pot indica diferite coduri de eroare în execuția programului.

La ieşirea din program cu funcția exit(), sunt închise toate fişierele care fuseseră deschise prin programul respectiv şi sunt golite buffer-ele de ieşire.

Este prezentat un exemplu de folosire a funcției exit() într-un program ce defineşte un meniu care permite selecția între patru opțiuni, cea de a patra producând încheierea execuției programului:

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>

void op1()
{
        printf("\noptiune 1");
        getch();
        return;         // instrucțiunea return încheie execuția funcția căreia aparține şi se revine
}                       // în funcția apelantă, în cazul de față se va reveni la funcția main()
void op2()
{       printf("\noptiune 2");
        getch();
        return;
}

void op3()
{       printf("\noptiune 3");
        getch();
        return;
}

void af_meniu()
{       printf("\n1. Optiunea 1");
        printf("\n2. Optiunea 2");
        printf("\n3. Optiunea 3");
        printf("\n4. Iesire");
        printf("\n\n\nIntrodu optiunea:");
        return;
}

void main()
{
```

```
int optiune;
do
{  af_meniu();
   fflush(stdin);
   optiune=getchar();
   switch(optiune)
   {    case '1':  op1(); break;          // instrucțiunea break întrerupe doar execuția
        case '2':  op2(); break;          // instrucțiunii switch
        case '3':  op3(); break;
        case '4':  exit(0);               // funcția exit() încheie execuția programului
   }
} while(1);          // condiția de execuție a instrucțiunii while este exprimată printr-o constantă,
                     // ceea ce duce la ciclare infinită; finalizarea ei se va face prin funcția exit()
}
```

6.2.5. Instrucțiunea de salt necondiționat goto

Instrucțiunea goto are ca efect saltul la eticheta specificată. Sintaxa sa este:

goto nume_et;

Eticheta nume_et este specificată folosind sintaxa:

nume_et :

Eticheta se poate afla, în succesiunea de instrucțiuni, atât înaintea instrucțiunii goto, cât şi după aceasta, dar ele trebuie să se regăsească în aceeaşi funcție. Nu este permis saltul dintr-o funcție în alta.

O secvență care calculează factorialul pentru o valoare N se poate scrie folosind instrucțiunea goto astfel:

```
   int i, N=4, fact;
   i=1;                  // inițializarea variabilelor de lucru
   fact = 1;

et:                      // definirea unei etichete; marchează locul din program în care se
                         // va face salt cu instrucțiunea goto
   fact *= i;
   i++;
   if (i<=N)
        goto et;         // se produce saltul la eticheta et; instrucțiunile dintre etichetă şi
                         // această instrucțiunea goto; efectul este similar cu cel obținut
                         // prin folosirea instrucțiunilor repetitive (for, while, do-while)
printf("%d", fact);
```

Observații

- Instrucțiunea goto încalcă principiile programării structurate şi reduce claritatea programului, astfel încât este recomandată evitarea ei. Instrucțiunile de selecție şi de ciclare pot înlocui această instrucțiune, oferind în plus față de aceasta un bun control al programului.

Exemple

```
/*************************************************************************
```

Exemplul 6.1. - Să se întocmească un program prin care se afişează codul ASCII. Se vor face afişări în format %c şi %d pentru a vedea corespondența caracter-valoare numerica.

Observații: Se vor face afişări în ordine crescătoare sau descrescătoare, folosind for, while sau do-while.

```
*************************************************************************/

#include<stdio.h>
#include<conio.h>

void main ()
{
  int i;

//se afiseaza codul ASCII folosind instructiunea for
  for( i=1 ; i<255 ; i++)
  {
    printf("%3d - %c  ", i, i);
    if( !( i%8 ))            // dupa afisarea a 8 valori se trece la rand nou
      printf("\n");
  }
  getch();

//se afiseaza codul ASCII folosind instructiunea while
  i=1;
  while( i<255 )
  {
    printf("%3d- %c  ", i, i);
    if( !( i%8 ))
       printf("\n");
       i++;
  }
  getch();

//se afiseaza codul ASCII folosind instructiunea do...while
  i=1;
  do
  {
    printf("%3d - %c  ", i, i);
    if( !( i%8 ))
      printf("\n");
      i++;
  }
  while( i<255 );
  getch();
}
```

```
/*************************************************************************
```
Exemplu1 6.2. - Să se întocmească un program prin care se citesc de la tastatura doi întregi >=2. Să se afişeze divizorii celor doi întregi. Să se calculeze şi să se afişeze c.m.m.d.c. şi c.m.m.m.c. pentru cele două valori.

```
*************************************************************************/

#include<stdio.h>
#include <conio.h>

void main(void)
{
 int nr1, nr2, i, min, cmmdc, cmmmc;

 do
 { printf("nr1=");
  scanf("%d",&nr1);
 }
 while (nr1<=1);        //se citesc valori >1

 do
 { printf("nr2=");
  scanf("%d",&nr2);
 }
 while (nr2<=1);        //se citesc valori >1

 printf("\nDivizorii numarului %d sunt:\n", nr1);
 for(i=2 ; i<=nr1 ; i++)
  if (!(nr1 % i))
    printf("%d ", i);

 printf("\nDivizorii numarului %d sunt:\n", nr2);
 for(i=2 ; i <= nr2 ; i++)
  if (!(nr2%i))
    printf("%d ", i);

 min=nr1<nr2 ? nr1 : nr2;      // se determina minimul dintre nr1 si nr2

// se determina c.m.m.d.c.
 for( i=2, cmmdc=1; i<=min ;i++)
        if(!(nr1%i)&&!(nr2%i))
                cmmdc = i;

// se determina c.m.m.m.c.
 cmmmc = nr1*nr2/cmmdc;

 printf("\ncmmdc = %d", cmmdc );
 printf("\ncmmmc=%d\n",cmmmc );

 getch();
}
```

Întrebări. Exerciții. Probleme.

1. Se consideră intervalul de valori [a , b]. Se citesc valori pentru a şi b. Să se afişeze toate valorile divizibile cu 5 din intervalul dat.

2. Sa se scrie un program in care intr-o secvența repetitiva se citesc de la tastatura valori întregi pozitive pentru o variabilă de tip int, pentru fiecare valoare afişându-se factorialul. Secvența este finalizata la introducerea valorii 0 (zero).

3. Să se scrie un program în care se declară o variabilă de tip int. Se citeşte de la tastatură. Să se determine şi să se afişeze inversul valorii acesteia. (De exemplu, inversul lui 12345 este 54321).

   Notă: Se vor determina cifrele care alcătuiesc numărul ca resturi obținute prin împărțiri succesive la 10, inversul numărului obținându-se prin adăugarea acestor cifre însoțite de înmulțiri succesive cu 10.

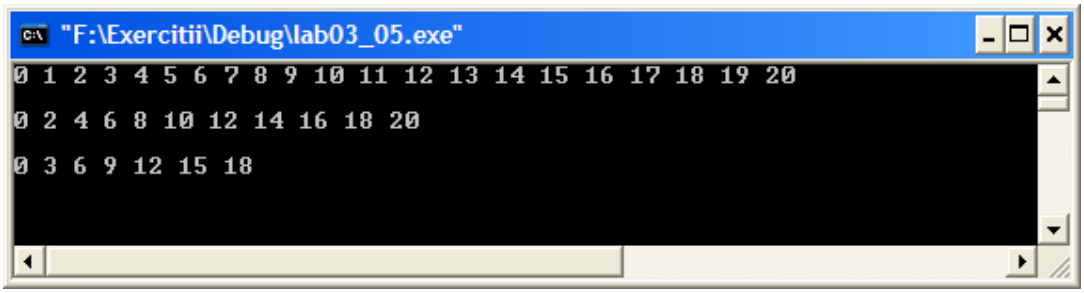
4. Să se întocmească un program în care :
   - se citeşte o valoare x cuprinsă în intervalul [1,170];
   - se citeşte o valoare y cuprinsă în intervalul [1,x];
   - se calculează şi afişează x! şi y!;
   - se calculează şi afişează aranjamente de x elemente luate câte y

     $$A(x, y)=x*(x-1)*(x-2)*\ldots* (x-y+1)$$

   - se calculează şi afişează combinări de x elemente luate câte y

     $$C(x, y) = A(x, y) / y!$$

5. Să se scrie un program care afişează toate valorile întregi din intervalul [0, 20], valorile divizibile cu 2 şi apoi cele divizibile cu 3 din acelaşi interval, ca în exemplul următor.



Nota:

Se vor realiza variante de program folosindu-se pe rând instrucțiunile while, do while, for.