LUCRAREA 5

Scopul lucrării este de a observa efectele echivalente ale unui `if-else` prin varianta `switch-case`, înţelegerea întreruperilor şi salturilor într-un limbaj de nivel-mediu şi studiul introductiv asupra instrucţiunilor de ciclare, prin prezentarea ciclului `for( )`.

I. OBSERVAŢII TEORETICE

1.1. Instrucţiunea de decizie `switch-case`

Dacă este să traducem în română pe `switch`, atunci termenul cel mai potrivit ar fi `întrerupător` (pentru două valori), sau `nod de cale ferată` (pentru mai multe situaţii).

Tocmai acesta este şi sensul din `C`, anume de test a unei variante, de alegere între două alternative (variante), sau de testare între mai multe cazuri posibile.

Instrucţiunea `switch-case` este legată (similară) lui `if-else`, deoarece funcţionarea sa se apropie, în mod echivalent, de cea a lui `if-else`. Există variantă `switch` şi cu un singur `case`, ca şi variante cu număr mai mare de `case`, după cum există şi `if` fără `else`, ca şi `if` cu mai multe ramuri de tip `else-if`.

Sintaxa este relativ simplă. Cuvântul-cheie `switch` este urmat de paranteze rotunde, ca şi cum ar reprezenta un nume de funcţie. Între parantezele rotunde se scrie numele variabilei de test. Urmează corpul instrucţiunii (între acolade). În interiorul acestui corp de instrucţiuni apar testele `case`.

```
switch(valIntreaga)
{
   case val_1:   {
                     instrucțiuni_1;
                     [break;]
                 }
   case val_2:   {
                     instrucțiuni_2;
                     [break;]
                 }
   case val_3:   {
                     instrucțiuni_3;
                     [break;]
                 }
   …
   default:      {
                    instrucțiuni_n;
                 }
}
```

Valorile `val_i` sunt proprii tipului de dată întreg, al variabilei care se testează. După cum ne amintim din laboratorul 2, din familia întregilor fac parte şi caracterele, astfel că se poate ca un `switch-case` să testeze valorile unei variabile caracter.

Se observă notaţia cu paranteze drepte a instrucţiunilor `break;`. Acesta semnifică faptul că sunt opţionale, deci pot lipsi. Funcţionarea lui `switch` în această situaţie este diferită de cea în care ele apar.

Modul de interpretare este următorul: se compară valoarea variabilei de test de la intrarea în `switch` cu `val_1`. Dacă cumva chiar aceasta este valoarea pe care variabila o are, se execută corpul de instrucţiuni asociat acelei `etichete case`, adică `instrucţiune_1`, după care, există două variante:

- dacă `break;` este prezentă atunci se sare pe linia următoare întregului corp al lui `switch-case`, instrucţiunea `switch` încheindu-se;
- dacă `break;` nu apare, atunci execuţia se continuă până se întâlneşte un `break;`, sau până se încheie de drept corpul asociat lui `switch` (prin acolada închisă de final). Deci se continuă cu corpul `instructiune_2`, apoi cu `instrucţiune_3`, până la întâlnirea primei instrucţiuni `break;`.

Dacă variabila de test nu are, la un moment dat, nici una dintre valorile `val_i`, atunci se ajunge pe eticheta `default:` şi se execută corpul asociat acestei etichete.

Observaţii:

1. Poate lipsi şi `default:`, astfel că există situaţia în care trecerea printr-un `switch` nu are nici un efect. **De obicei** este recomandat ca `default:` să apară, iar în cadrul său să se dea o explicaţie asupra valorilor presupuse corecte ale variabilei de test.
2. Un `switch-case` poate conţine alte instrucţiuni de acelaşi tip (tot `switch-case`). Aceasta se numeşte imbricare (cuprindere) şi am întâlnit-o deja la comentarii şi la corpuri de instrucţiuni.
3. Acoladele care compun corpul unui `case` pot lipsi, pentru că execuţia instrucţiunii `switch` se opreşte automat la întâlnirea unui `break;` Deci, apariţia instrucţiunilor `break;` face inutilă prezenţa acoladelor unui `case`.

1.2. Echivalenţa dintre `switch-case` şi `if-else` multiplu

După cum am văzut în încheierea laboratorului 4, într-un `if-else` multiplu la fiecare ramură `else` se merge pe condiţia negată a ramurii `if` asociate. Iar ultimul `else` singular, are semnificaţia de "toate valorile rămase, în afară de cele luate deja în consideraţie".

Pentru un `switch-case` primul `case` are rolul primului `if` al unui `if-else` multiplu. `case`-ul următor are semnificaţia primului `else-if`. Al treilea case

înlocuieşte al doilea `else-if`. În mod analog se continuă până la `default:`, care are semnificaţia ultimului `else` singular dintr-un `if-else` multiplu.

Astfel vă puteţi construi deciziile pe baza uneia sau alteia dintre aceste două instrucţiuni echivalente (`if` sau `switch`).

1.3. Instrucţiunea de întrerupere `break;`. Noţiunea de etichetă (`label`)

1.3.1. Asociată unui `switch-case`

Aţi observat efectul prezenţei/absenţei unui `break`; în cazul lui `switch`.

Instrucţiunea `break;` are rolul de **întrerupere** a execuţiei, în momentul în care este întâlnită. Acesta este rezultatul în momentul unei apariţii a sa în cadrul unui `case`: încheierea corpului lui `switch`.

1.4. Instrucțiunea `goto`

Toate apariţiile `case val_i:` dintr-un `switch` poartă numele de `etichete`.

Prin definiţie, o etichetă reprezintă combinaţia dintre o denumire corectă (care începe cu literă sau `underscore`) urmată de două puncte.

Efectul instrucţiunii `goto` este acela de întrerupere forţată a execuţiei secvenţiale a programului, şi trecerea la o instrucţiune ulterioară sau anterioară celei curente, prin ***generarea unui salt*** către instrucţiunea respectivă (`salt` = `jump`).

Dezavantajul major este în procesul de depanare. Să presupunem că într-un program există cel puţin două cerinţe de funcţionare în urma cărora să se facă un salt către o anumită instrucţiune. În cadrul procesului de depanare, odată situaţi pe acea instrucţiune, nu vom şti de unde s-a făcut saltul, deci se generează o ***ambiguitate de nepermis*** în anumite situaţii.

Sintaxa este următoarea:

```
goto label;
```

unde denumirea `label:` apare ulterior sau anterior acestui moment.

1.5. Instrucţiuni de ciclare (buclare) - instrucţiunea `for()`

În cadrul programelor *C* este nevoie şi de instrucţiuni care se repetă de un număr de ori, sau de instrucţiuni proprii unor tipuri de dată (cum sunt `vectorii` sau `matricele`) care apar strâns legate de ciclări.

Sunt două mari clase de instrucţiuni de ciclare:

- cu test iniţial - `for()` şi `while()`;
- cu test final - `do-while`.

Ne reamintim: condiţiile care apar în testul ciclurilor pot lua valorile de adevăr adevărat/fals, care sunt valori logice. Deci expresiile care apar în cadrul condiţiilor de cicluri sunt evaluate d.p.d.v. `boolean`.

Observaţie:

În *C*, valoarea `Fals` este asociată numai lui `0`, şi orice valoare diferită de `0` reprezintă `Adevărat` (inclusiv valori întregi negative, sau numere reale).

1.5.1. Ce semnifică testul iniţial? De ce este ciclul `for()` ciclu *cu test iniţial*

Prin **test iniţial** se înţelege acea condiţie care se evaluează **`înainte de intrarea în corpul ciclului`**.

Prin opoziţie, **testul final** reprezintă condiţia care se evaluează mereu **`la finalul unei instrucţiuni de ciclare`** (înaintea următorului pas din ciclu).

`for()` este o instrucţiune de ciclare cu test iniţial, deoarece compilatorul permite intrarea sa în execuţie doar dacă valoarea de adevăr a `condiţiei de test` este `Adevărat`.

Sintaxa generală a instrucţiunii `for()` are forma:

```
for(start; condiţie; pas) {corpInstrucţiuni;}
```

Modul de funcţionare este următorul:

- se evaluează instrucţiunea expresie pentru `start`;
- se evaluează apoi `condiţia de test`. Dacă aceasta este adevărată, atunci se intră în corpul ciclului. Dacă nu, ciclul este ignorat, iar execuţia continuă cu linia de program imediat următoare lui `for()`;
- dacă am avut condiţie adevărată, atunci se execută, secvenţial, fiecare instrucţiune din corpul ciclului. Iar ultima dată se evaluează expresia pentru `pas`.

Acoladele se impun doar dacă în corpul instrucţiunii va apărea cel puţin o instrucţiune (re-vedeţi laboratorul 2). Corpul de instrucţiuni se bucură de aceleaşi proprietăţi pe care le-am amintit în laboratorul 2.

Cazuri de sintaxă particulară pot fi:

a)

```
   start;
   for(; condiţie; pas) {corpInstrucţiuni;}
```

b)

```
start;
for(; condiţie; ) {
   corpInstrucţiuni;
   pas;
   }
```

În situaţia **a)**, `expresia din start` lipseşte dintre parantezele rotunde, dar nu lipseşte definitiv. Apariţia sa precedă (cum este normal) intrarea în ciclu, şi evaluarea sa nu lipseşte. Această evaluare poate influenţa valoarea de adevăr a

`condiției de test`. Ea poate conţine şi condiţiile de intrare în ciclu, prin influenţa variabilelor de control al instrucţiunii.

Fiecare dintre porţiunile unui ciclu `for()` poate conţine expresii, sau instrucţiuni expresii ce conţin operatorul virgulă, având o comportare interesantă.

Exemplu:

```
for(i=1, i++; (i+=2) <= 4; i++, i -= 2)
   printf("\n Valoarea lui i: %i", i);
```

Pentru situaţia **b)**, singura apariţie în locul indicat conform sintaxei generale este a `condiției de test`. `Expresia de start` precede, corect, intrarea în ciclu, iar instrucţiunea pentru `pas` apare prinsă în interiorul corpului ciclului. Se respectă modul general de funcţionare al lui `for()`.

Exemplu:

```
int a = 2 , i;
i = 6;                  // expresia pentru start
for(; a % i <= 5; ){   // restul împărțirii lui 'a' la 'i'
   printf("\n restul împărțirii a doi întregi: %i", a % i);
   ++i;  // Mă asigur că la un moment dat condiția devine falsă.
         // Expresia pentru pas.
}
```

1.6. Cicluri infinite

Sintaxa generală este:

```
for( ;; ) {corpInstrucțiuni;}      // ciclu infinit
```

Compilatorul are o variantă implicită de funcţionare pentru acest caz: va recunoaşte în această sintaxă un **`ciclu infinit`**.

După cum se vede, nu apare nici o condiţie deci nu avem cum controla acest ciclu.

Chiar dacă prevedem expresiile pentru `start` şi `pas`, nu reuşim să oprim funcţionarea altfel, decât prin folosirea unei instrucţiuni amintite deja, anume `break;`. Pe o condiţie impusă (un `if()`) vom folosi instrucţiunea `break;`, moment în care, ca la `switch-case`, ciclul `for()` se încheie forţat, iar controlul este dat liniei următoare ciclului.

În cele două exemple următoare, avem două situaţii: o variantă fără instrucţiune `break;`, şi două cu.

Exemple:

1. Ciclu `for()` infinit, fără instrucţiune `break` (ieşire forţată)

```
int i;    // o variabila neinițializată
for(;; ){
   printf("\n ! Ciclu infinit !");
```

```
    puts("\n Apăsați Ctrl+break sau Ctrl+C pentru a-l opri");
    printf("\n %i", i);
    }
```

2. Ciclu `for()` infinit, cu ieşire forţată

```
unsigned long int i = 0; // pentru a vedea funcționarea am ales
   // un tip de dată cuprinzător (valoarea maximă este 2^32-1).
for(;; ){
   if(i == (pow(2, 8) - 1)) break;  // oprire forțată pe o condiție:
         // daca 'i' are valoarea 255 (FF|16) atunci ciclul se opreste.
   ++i;
   printf("\n %i", i);
   }
```

3. Ciclu `for()` infinit cu ieşire forţată

unsigned char ch;

```
unsigned long int i; // pentru a vedea funcționarea am ales un tip
         // de dată cuprinzător (valoarea maximă este 2^32-1).
for(;; ){
   if(! (ch = kbhit()) ) printf("\n %c", ch);
   else break;        // oprire forțată pe o condiție.
   ++i;
   printf("\n %i", i);
   }
```

Atenţie!

Funcţia `kbhit()` este o funcţie proprie compilatoarelor `BorlandC/C++`.

Observaţii generale:

- Pentru orice instrucţiune de buclare, corpul său trebuie să prevadă modificarea condiţiei, astfel încât la un moment dat aceasta să devină falsă, şi astfel, ciclul să se încheie. Dacă aşa ceva nu se întâmplă, programatorul va genera fie un ciclu infinit (condiţia este mereu adevărată), fie - în funcţie de context -, un ciclu care nu rulează `niciodată`.
- Există cicluri `for()` (şi nu numai) imbricate.

1.7. Ciclu `for()` cu instrucţiune vidă

În *C* există instrucţiunea vidă, adică instrucţiunea care nu are nici un efect. Aceasta este instrucţiunea care nu conţine altceva decât **`punct şi virgulă`**.

Rolul ei este acela de `a aştepta` într-o situaţie impusă.

De exemplu, dacă am întâlnit o valoare, sau avem de ignorat o gamă de valori, putem opta pentru varianta în care programul să nu facă nimic.

În exemplul următor se numără literele mari, ignorându-se cele mici:

```
…
for(; (ch = getch()) != ESC; )
{
   if(ch>'a' && ch<= 'z');
   else {
   printf("\n Literă mare");
   nrLitMare ++;
   }
}
…
```

Aici am presupus că, dacă o variabilă caracter `ch` este în gama literelor mici (sau `minuscule`, în contrast cu `majuscule`) atunci nu se execută nimic. Iar dacă se întâlneşte vreo literă mare, apariţia acesteia se numără (`nrLitMare++`).

1.8. Întârzieri - cicluri `for()` de aşteptare

O altă situaţie utilă în care ne ajută instrucţiunea vidă este aceea a `aşteptărilor` (`delay`).

Un exemplu este acela în care aplicaţia ar putea fi scrisă pentru `rutine de nivel jos`, caz în care este posibil să aşteptăm un răspuns din partea unui periferic, sau a unui port de comunicaţie (serial sau paralel).

Un exemplu de buclă de aşteptare ar putea fi:

```
typedef unsigned long int ULI; // tipul întreg lung fără semn
ULI i = 0;      // o variabilă de acest tip
for(; i<= pow(2, 31); i++);  // aşteptare.
```

Puterea `31` a lui `2` semnifică `2 Giga`. Tipul de dată corect pentru a putea stoca această valoare este `unsigned long`. Astfel că întregul `i` este de acest tip. În acest ciclu, la fiecare pas se incrementează `i` şi se verifică dacă condiţia este încă adevărată. Prezenţa instrucţiunii vide impune acest mod de funcţionare.

Timpul de aşteptare aici este dat de:

$2^{31} * (t_{incrementare} + t_{execuție\ Funcție})$

Există tabele ale operaţiilor de bază, în care se specifică şi numărul de `cicli de tact` pe care îi cere acea operaţie din timpul procesor.

Deoarece ciclurile se pot imbrica (se pot cuprinde unul în altul), dacă este cazul putem genera şi aşteptări mai îndelungate.

1.9. Detalii de programare

Pentru înţelegerea programelor utilizate în lucrare studiaţi *ANEXA* lucrării 5.

1.10. Comenzile compilatorului

Înainte de compilare codul sursă trebuie salvat (cu combinaţia de taste *CTRL+S*), sau, echivalent, din meniul File, cu comanda *Save as…*.

Compilarea se va face cu comanda *CTRL+F9*, iar rularea cu comanda *CTRL+F10*. Compilarea şi rularea se pot executa, succesiv, printr-o singură tastă, anume *F9*. Acţiunile echivalente din meniu: meniul *Execute* comanda *Compile*, respectiv *Compile + Run*.

Observaţii:

1. Orice nouă modificare a codului-sursă trebuie urmată obligatoriu de salvarea acestuia (`CTRL+S`). Abia după aceea pot urma celelalte acţiuni dorite.
2. Dacă nu se realizează aducerea la zi a programului, compilarea şi rularea ce urmează se fac tot pe varianta veche, adică ce de dinaintea ultimelor modificări.

II. DESFĂŞURAREA LUCRĂRII

2.1. Sintaxa instrucţiunii `switch-case` se poate vedea în programul `lab5_1.cpp`. Se prezintă şi situaţiile speciale, în care:

- instrucţiunea `break` poate lipsi de pe fiecare ramură de decizie.
- nu apar acoladele corpurilor de instrucţiuni asociate fiecărei decizii `case:`
- eticheta `default` lipseşte.

Rulaţi programul şi observaţi efectele de la ieşire în fiecare din aceste situaţii. Notaţi-vă şi reţineţi comportamentul tipic fiecărei situaţii. Reveniţi dacă este necesar la explicaţiile din paragraful 1.1.

2.2. Echivalenţa la execuţie între un `if-else` şi `switch-case` este tratată în programul *lab5_2.cpp*. Prin rularea programului şi observarea afişărilor se justifică această echivalenţă. Apar aici situaţii din programul anterior (lipsa corpurilor de instrucţiune în situaţi anumitor decizii `case:`). Reveniţi la paragraful 1.1. dacă este necesar.

2.3. Efectul instrucţiunii `break` este studiat în programul `lab5_3.cpp` pentru un alt exemplu de `switch-case` decât cel de la punctul 2.1.

Modificaţi codul-sursă pornind de la forma clasică de `switch-case`, pentru a înţelege comportamentul compilatorului.

2.4. Instrucţiunea `goto` şi noţiunea de `etichetă (label)` sunt tratate în programul `lab5_4.cpp`. Eticheta are o utilizare mai largă decât cea

pentru `default:` în situaţia `switch-case`. Puteţi observa o asemănare între modul de scriere a programelor în limbaj de asamblare şi cel din acest program. De obicei nu este recomandată conceperea uni program plecând de la ideea lui `goto` (a salturilor forţate, în general).

Modificaţi `'linia algoritmică'` de execuţie a programului păstrând însă acelaşi efect la ieşire.

2.5. Instrucţiunea `for()` în varianta generală este prezentată în programul `lab5_5.cpp`. Citiţi comentariile din textul programului. Sunt importante şi ajută la înţelegerea comportamentului compilatorului.

Aşa cum se observă, condiţia de rămânere în ciclu conţine o modificare a valorii contorului `i`, pe lângă modificarea din zona de `pas` a ciclului. Dacă se renunţă la paranteza rotundă, deci se modifică ordinea operaţiilor, efectul este altul la ieşire (comportamentul ciclului `for()` se schimbă).

Renunţaţi la comentariile de pe liniile `53` şi `62` şi reluaţi programul. Notaţi-vă concluziile pe care le credeţi utile în înţelegerea funcţionării programului.

2.6. Ciclurile infinite sau cele speciale sunt studiate în programul `lab5_6.cpp`. Apare ca necesară varianta de ieşire dintr-un asemenea ciclu. Aceasta de face forţat, pe anumite condiţii, sau folosind instrucţiunea `break`.

Prin înlăturarea comentariilor de pe liniile `45` şi `53` se obţine la ieşire un ciclu infinit prin definiţie. Ieşirea din acesta se face cu ajutorul tastelor `CTRL+C` sau `CTRL+Break`.

Apare aici scrierea funcţională a unui program, pentru că s-a creat o funcţie proprie, de calcul a puterii întregi a unui număr. Reveniţi la **`Lucrarea 3`** dacă este necesar.

Rulaţi programul de mai multe ori, cu modificări astfel făcute încât să înţelegeţi - prin efectele de la ieşire - modul de execuţie propriu compilatorului. Notaţi-vă concluziile şi reţineţi efectele fiecărei modificări pe care o faceţi.

2.7. Un ciclu *for()* urmat de o instrucţiune vidă (`;`) este o situaţie posibilă. Aceasta este tratată în programul `lab5_7.cpp`. Observaţi cum instrucţiunea vidă generează de fapt o `aşteptare` (nu se face nimic) pe timpul cât condiţia este adevărată.

2.8. O `întârziere` prin definiţie este studiată în programul `lab5_8.cpp`. La punctul anterior s-a pregătit modul de lucru propriu acestui punct din desfăşurarea lucrării. Apare operatorul `typedef` şi construcţia unei funcţii proprii. Reveniţi la ***Lucrarea 4*** (pentru `typedef`) respectiv la ***Lucrarea 3*** (pentru lucrul în variantă funcţională).

Funcţia `putere(baza, exponent)` din acest program poate genera depăşiri dacă valoarea exponentului nu rămâne în gama `[0, 31]`.

Încercaţi şi alte valori ale exponentului decât cea existentă în codul sursă.

Observaţi relativa rapiditate de apariţie la ieşire a mesajului de final. Aceasta ne conduce la ideea unor cicluri `for()` cuprinse unul în altul (`imbricate`) pentru a genera aşteptări mai îndelungate.

III. ÎNTREBĂRI

3.1. Daţi un exemplu de `switch-case` care testează valoarea unui caracter şi afişează mesajul `"Foarte Bine"` dacă acel caracter este 'A', `"Bine"` dacă el este 'B', `"Trecere"` dacă este 'C' sau 'D', şi `"Greşeală"` dacă este 'F'. Pentru orice altă valoare trebuie afişat mesajul `"Caracter Invalid"`.

3.2. Să presupunem că avem o funcţie denumită `sqrt()`, care returnează (întoarce) valoarea rădăcinii pătrate a unui număr. Scrieţi un ciclu `for()` ce afişează valoarea rădăcinii pătrate a tuturor numerelor între `100` şi `2`, în ordine descrescătoare.

3.3. Gândiţi-vă la un exemplu de `switch-case` în care, depinzând de valoarea întreagă a variabilei testate să se calculeze puterea întreagă (dată chiar de valoarea testată) a unei baze de numeraţie, folosind doar ciclu `for()`.

3.4. Scrieţi un ciclu for() infinit care se opreşte forţat în situaţia în care un caracter întâlnit este una dintre valorile: `'a'`, `'A'`, `'z'`, `'Z'`. Pentru testul acestor valori folosiţi un `switch-case`. Restul valorilor sunt ignorate.

3.5. Cu ajutorul unei instrucţiuni *switch-case* afişaţi într-un mesaj tipul rădăcinilor (*reale, egale, complexe*) funcţie de semnul discriminantului unei ecuaţii de grad 2.

IV. ANEXA - sursele complete ale programelor

```
// lab5_1.cpp
// switch-case clasic (variante).
#include<stdio.h>

int main(void){
 char ch = '1';     // codul hexa al lui 1 este 0x30h = 48 in baza 10
 int i3, i4;        // pentru ultimele variante de switch-case, de test pentru valori
             // intregi.
//varianta clasica
 switch(ch){// switch_1
  case '0': {
```

```
                    printf("\n switch_1: Valoarea lui ch: %c", ch);
                    break;
                    }
  case '1': {
                    printf("\n switch_1: Valoarea lui ch: %c", ch);
                    break;
            }
  default: printf("\n switch_1: Valoare ne-interesanta: %c, %i", ch, ch);
 }// end_switch_1.

// varianta clasica in care nu apar acoladele corpurilor 'case'
 printf("\n\n Dati o valoare tip caracter (sau intreg scurt, fara semn): ");
 scanf("%c", &ch);

 switch(ch){// switch_2
  case 'a': printf("\n switch_2: Valoarea lui ch: %c", ch);
                    break;
  case 'B': printf("\n switch_2: Valoarea lui ch: %c", ch);
                    break;
  default: printf("\n switch_2: Valoare ne-interesanta: %c, %i", ch, ch);
 }// end_switch_2.

// variantă in care 'break' apare sau nu. Se testeaza valorile unui intreg, deci
// nu vom mai folosi apostrof pentru incadrarea valorii. Aceasta valoare este
// acum chiar un intreg.
 i3^=i3;    // reset
 printf("\n\n Dati o valoare tip intreg (sau intreg scurt, fara semn): ");
 scanf("%i", &i3);
 printf("\n valoarea lui 'i3' inante de intrarea in switch_3: %i", i3);

 switch(i3){// switch_3
  case 0:   printf("\n switch_3: Valoarea lui i3: %c", i3);
  case 1:   printf("\n switch_3: Valoarea lui i3: %i", i3);
  case 2:   printf("\n switch_3: Valoarea lui i3: %i", i3);
                    break;
  default: printf("\n switch_3: Valoare ne-interesanta: %c, %i", i3, i3);
 }// end_switch_3.

// varianta fara 'break'. Executia se opreste in acolada de final
 printf("\n\n Dati o valoare tip intreg (sau intreg scurt, fara semn): ");
 scanf("%i", &i4);

 switch(i4){// switch_4
  case 0:   printf("\n switch_4: Valoarea lui i4: %c", i4);
  case 1:   printf("\n switch_4: Valoarea lui i4: %i", i4);
  case 20:  printf("\n switch_4: Valoarea lui i4: %i", i4);
  default: printf("\n switch_4: Valoare ne-interesanta: %c, %i", i4, i4);
 }// end_switch_4.

// final
 scanf("%i", &ch);
}
```

// lab5_2.cpp
// Echivalența *if-else*, *switch-case*.

```
#include<stdio.h>

int main(void)
{
 int Int = 30;
```

```
 if(Int >=1 && Int < 10)
        printf("\n if: \t Mai mic decat 10 (cel putin 1 si cel mult 9).");
 else       if (Int < 20) printf("\n if: \t Intre 10 si 19.");
            else    if (Int < 30) printf("\n if: \t Intre 21 si 29.");
                    else    printf("\n if: \t\t Minim 30 si peste.");
 switch(Int) {
  case 1:
  case 2:
  case 9:   printf("\n switch: \t Mai mic decat 10.");
                break;
  case 10:
  case 11:
  case 19:  printf("\n switch: \t Intre 10 si 19.");
                break;
  case 20:
  case 21:
  case 24:
  case 29:  printf("\n switch: \t Intre 21 si 29.");
                break;
  default:  printf("\n switch: \t Minim 30 si peste.");
 }// end_switch

// final
 scanf("%i", &Int);
}
```

// lab5_3.cpp
// Efectul lui *break* în *switch-case*.

```
#include<stdio.h>

int main(void){
 int const varInt = 4;

 switch(varInt) {
  case 4:   printf("\n Valoarea de intrare este: %i", varInt);
  case 5:   printf("\n Dati o valoare pentru variabila intreaga: ");
            scanf("%i", &varInt);
  case 10:  printf("\n Noua valoare este: %i", varInt);
            break;
  default:  printf("\n Ciclul switch se incheie.");
 }

// executia programului ajunge pe linia urmatoare, in urma apritiei lui break
// in cadrul uneia dintre etichetele 'case'
  printf("\n Valoarea de iesire: %i", varInt);

// final
 scanf("%i", &varInt);
}
```

// lab5_4.cpp
// goto

```
    #include<stdio.h>

    int main(void){
     char ch = 'a';
     int varInt = 20, nrTreceri=0;

     intrare:
     printf("\n Din nou: Valorile: ch= %c, varInt: %d", ch, varInt);
```

```
     if((varInt & ch) != 96) {
            printf("\n Rezultatul din if: %x, %u", varInt & ch, varInt & ch);
            nrTreceri++;
            goto final;    // pentru ca rezultatul sa fie 96, daca ch='a' atunci
                           // varInt= -2.
           }
     else goto final;
     iesire: goto intrare;

     final:
           printf("\n Sfarsit: Ultima valoare din if: %x, %i", varInt, varInt);
           printf("\n Numarul de treceri: %i ", nrTreceri);
           scanf("%i", &varInt);
    }
```

// lab5_5.cpp
// Instrucțiunea de ciclare *for()*. Studiu.

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

int main(void){
 int i;                    // pentru controlul ciclului 'for()'
 int pas = 0;              // pentru numaratoarea trecerilor prin ciclu
 int valoare  = !0; // pentru conditia din al doilea ciclu for().

 for(i=1, i++; (i+=2) <= 4; i++, i -= 2){
    pas++;  // la fiecare intrare in ciclu marchez pasul.
    printf("\n trecerea %i", pas);
    printf("\n Valoarea lui i: %i", i);
    }

//  La intrarea in ciclu, i = 2. Se evalueaza conditia iar valoarea ei este
// adevarata (verificati). La fiecare pas se scade -1 din valoarea lui i.
// (pentru ca i++ urmat de i-=2 are acest efect: i=i-1). In testarea conditiei
// insa, valoarea lui i creste cu 2 dupa care se face comparatia cu 4. Deci
// valoarea lui i, per ansamblu, creste cu 1 fata de anterioara, dupa care se
// evalueaza comparatia din conditie. La prima intrare in bucla, i= 4.
// La a doua, i = 5.

// in acest moment, 'valoare' este 1.
 printf("\n\n valoare: %i", valoare);
 do {
    printf("\n Ce valoare doriti?");  scanf("%i", &valoare);
    } while (valoare < 0);  // ignor toate valorile negative.

 pas = 0;
 for(i=1, i++; i+=2 <= valoare; i++, i -= 2){
    pas++;  // la fiecare intrare in ciclu marchez pasul.
    printf("\n trecerea %i", pas);
    printf("\n Valoarea lui i: %i", i);
    }

//  Un efect interesant se obtine daca se ignora parantezele rotunde din
// conditia ciclului for(). Adica se scrie conditia in varianta:
//            i+=2 <= 4.
//  Din cauza prioritatii operatorilor, <= este evaluat inaintea lui +=.
// Dar 2<=4  este adevarat, iar i creste cu 1. Apoi valoarea sa mai creste
// cu 1 (i++) si scade cu -2, adica ajunge la 2. Dupa care se evalueaza
// din nou conditia de test; el va creste cu 1 (valoarea conditiei de
// adevar), ajungand la 3.
```

```
//  Se afiseaza i: 3. Dupa care noua sa valoare inainte de conditia de test este
// tot 2, s.a.m.d.
//  Cum valoarea sa este diferita de 0 de fiecare data, rezulta o conditie mereu
// adevarata, deci un ciklu infinit.

// Renuntati la comentariile urmatoare si rulati din nou programul.
/*
 scanf("%i", &pas);
        // astept un numar intreg. Separ prima executie de ciclu
        // de a doua.
 pas = 0;   // reinitializarea lui 'pas', pentru un nou ciclu.
 for(i=1, i++; i+=2 <= 4; i++, i -= 2){
    printf("\n trecerea %i", pas);
    printf("\n Valoarea lui i: %i", i);
    pas++;
    }
*/

// final
 printf("\n Suntem la final. Dati o valoare intreaga...");
 scanf("%i", &i);
}
```

// lab5_6.cpp
// Cicluri infinite.

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

typedef unsigned long int ULI;

ULI putere(int, int);       // rezultatul maxim: (2 la puterea 32)-1

int main(void){
 int i;     // o variabila neinitializata

// 1. Ciclu for() infinit, cu iesire fortata (varianta)
 {
    unsigned long int i = 0;
                // pentru a vedea functionarea am ales un tip de
                // data cuprinzator (valoarea maxima este (2_la_32)-1 ).
    printf("\n Ciclul infinit 2.");
    for(;; ){
            if(i == (putere(2, 4) - 1)) break;
                                        // oprire fortata pe o conditie:
                                        // 'i' sa aiba valoarea 15 (F|16).
            ++i;
            printf("\n %i", i);
            }
 }
 printf("\n Pauza intre cicluri. Dati un caracter...");
 scanf("%i", &i);


// 2. Ciclu for() infinit, cu iesire fortata (varianta)
 {
    unsigned char ch;
    unsigned long int i=0;
                        // pentru a vedea functionarea am ales un tip de
                        // data cuprinzator (valoarea maxima este (2_la_32)-1).
    printf("\n Ciclul infinit 3.");
    for(;; ){
```

```
            ch = getche();
            if( ch != 'z' && ch != 'Z') {
            printf("\n caracterul: %c, codul ASCII: %i", ch, ch);
            ++i;
            }
            else break;             // oprire fortata pe o conditie.
            }
 }

 // ciclu infinit prin definitie. NU se opreste decat cu Ctrl+C sau Ctrl+Break.
 /*
 {
    for(;; ){       // prin definitie, conditia este aici Mereu adevarata.
            printf("\n ! Ciclu infinit !");
                    puts("\n Apasati Ctrl+break sau Ctrl+C pentru a-l opri");
                    printf("\n %i", i);
                    }
 }
 */

// final
 i^=i;              // reset.
 printf("\n Finalul. Dati un caracter de incheiere...");
 scanf("%d", &i);
}

ULI putere( int b,
            int e) {
 int i;
 ULI r=1;   // pentru rezultat.

 switch(e){         // testul argumentelor
    case 0 :        printf("\n Exponent nul. Stop!");
                    return 1;       // parasesc functia
    default:{       // orice valoare, cu exceptia lui 0.
                    for(i=e; i>=1; i--) {
                    r = r*b;
                    printf("\n Rezultat partial: %u", r);
                    }
    } // end_default.
 } // end_switch.
 printf("\n Rezultat final: %u", r);

 return r;
}
```

// lab5_7.cpp
// Ciclu for() cu instrucțiune vidă.

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

#define ESC 0x1b    // codul ASCII pentru Escape este 27 in zecimal, adica 1B|16

int main(void){
 char ch;
 int nrLitMare = 0; // initial nu am nici o litera (n\mica sau mare).
 int nrCiclari = 0; // numarul de treceri prin for().

 for(; (ch = getch()) != ESC; ){
    if(ch>='a' && ch<= 'z');// instructiunea vida (nu se face nimic).
    else {
```

```
            printf("\n Litera mare");
            nrLitMare ++;
            }
    nrCiclari++;    // de cate ori am trecut prin corpul buclei.
    }

 printf("\n Am intalnit %i litere mari.", nrLitMare);
 printf("\n Numarul celorlalte intrari: %i", nrCiclari - nrLitMare);

 // final
 scanf("%i", &nrLitMare);
}
```

// lab5_8.cpp
// Ciclu for() pentru așteptări, sau întârzieri (*delay*).

```
#include<stdio.h>

typedef unsigned long int ULI;      // tipul întreg lung fara semn
ULI putere(int, int);       // declaratia functiei putere (prototip).

int main(void){
 ULI i = 0; // o variabila de acest tip

 for(; i<= putere(2, 24); i++);     // asteptare.
 // Puteti, pentru experimentare, sa schimbati valoarea exponentului,
// dar numai in limitele [0, 31]. Altfel se genereaza depasiri, programul
// putand avea efecte imprevizibile la rulare )poate rula incorect (!)).
 //  Incercati exponenti de valoare: 10, 20, 25, 30. Observati rapiditatea
 // afisarii (aparitiei) mesajului de final.

 // final
 printf("\n Gata! Dati un caracter...");
 scanf("%i", &i);
}

ULI putere( int b,
            int e) {
 int i;
 ULI r=1;   // pentru rezultat.

 switch(e){         // testul argumentelor
    case 0 :        printf("\n Exponent nul. Stop!");
                    return 1;       // parasesc functia
    default:{       // orice valoare, cu exceptia lui 0.
                    for(i=e; i>=1; i--) r = r*b;
    } // end_default.
 } // end_switch.

 return r;
}
```