

Instrucțiunile limbajului C.

Responsabili:

- Darius Neațu (CA 2019-2020) [mailto:neatudarius@gmail.com]
- Dorinel Filip (CA 2019-2020) [mailto:ion_dorinel.filip@cti.pub.ro]
- Emil Racec (2012) [mailto:emil.racec@gmail.com]
- Octavian Rînciog (2010) [mailto:octavian.rinciog@gmail.com]

Obiective

În urma parcurgerii acestui laborator studentul va fi capabil să:

- scrie programe C utilizând structuri condiționale și de repetiție

Noțiuni teoretice

Instrucțiuni condiționale

If-else

`if...else` este cea mai simplă instrucțiune condițională. Poate fi folosită în mai multe forme:

```
if (condition) {  
    // instructions  
    //...  
}
```

```
if (condition) {  
    // instructions  
    // ...  
} else {  
    // other instructions  
    // ...  
}
```

```
if (condition1) {  
    ...  
} else if (condition2) {  
    ...  
}  
...  
else if (conditionN) {  
    ...  
}
```

Instrucțiunea evaluează expresia `condition` și execută instrucțiunile dintre acolade doar dacă rezultatul este nenul. În varianta cu `else`, pentru rezultat nul este executat blocul de instrucțiuni aflat după `else`.

În al treilea caz, sunt evaluate pe rând condițiile și este executat blocul corespunzător primei condiții adevărate. Un exemplu de folosire este:

```
if (a == b) {  
    printf("Numerele sunt egale");  
} else if (a > b) {  
    printf("A este mai mare");  
}
```

Switch

`switch` este o instrucțiune menită să simplifice structurile condiționale cu mai multe condiții.

```
switch (expression) {  
    case constant1:  
        // instructions1  
    case constant2:  
        // instructions2  
    ...  
    default:  
        // instructions  
}
```

Valoarea `expression` este evaluată la un tip întreg, apoi această valoare este comparată cu fiecare constantă; este rulat blocul de instrucțiuni al valorii găsite. În caz ca numărul nu este egal cu nici una dintre constante, este executat blocul aflat după `default`.

După executarea ultimei instrucțiuni dintr-un bloc `case`, execuția nu continuă după blocul `switch`, ci la începutul următorului bloc `case` *constant* sau `default`. Pentru a ieși din blocul `switch`, se folosește instrucțiunea `break`.

```
int main() {  
    char c;  
    printf("Alegeți o opțiune:\n\t[a] afișare\n\t[s] ștergere\n\t[e] ieșire\n");  
    scanf("%c", &c);  
    printf("Ați ales: ");  
  
    switch (c) {  
        case 'a':  
            printf("afișare");  
            break;  
  
        case 's':  
            printf("ștergere");  
            break;  
  
        case 'e':  
            printf("ieșire");  
            break;  
  
        default:  
            printf("O opțiune inexistentă");  
            break;  
    }  
    return 0;  
}
```

```
int main() {  
    int n, n2;  
    printf("Introduceți o valoare între 0 și 5:");  
    scanf("%d", &n);  
    n2 = 1;  
    switch (n) {  
        case 5:  
            n2 *= 2;  
            /* fără break, continuă la următoarea instrucțiune */  
        case 4:  
            n2 *= 2;  
        case 3:  
            n2 *= 2;  
        case 2:  
            n2 *= 2;  
        case 1:  
            n2 *= 2;  
        case 0:  
            printf("2 la puterea %d este %d\n", n, n2);  
            break;  
    }
```

```
default:
    printf( "Valoare invalidă\n" );
}
return 0;
}
```

Instrucțiuni de repetiție

while

while execută un bloc de instrucțiuni atâta timp cât o anumită condiție este adevărată. Forma generală a unui ciclu **while** este:

```
while (expression) {
    // instructions
}
```

Câtă vreme **expression** are o valoare nenulă, instrucțiunile din blocul de după **while** sunt executate. Expresia este reevaluată după fiecare ciclu. Un astfel de ciclu poate să se execute o dată, de mai multe ori sau niciodată, în funcție de valoarea la care se evaluează expresia.

do ... while

do ... while este o instrucțiune repetitivă similară cu cea precedentă, singura diferență fiind că expresia este evaluată după executarea instrucțiunilor, nu înainte. Astfel, blocul va fi executat cel puțin o dată.

```
do {
    // instructions
} while (expression);
```

for

for reprezintă o formă mai simplă de a scrie un **while** însoțit de o expresie inițială și de o expresie de incrementare. Forma sa este:

```
for (expression1; expression2; expression3) {
    // instructions
}
```

Secvența de cod de mai sus este echivalentă cu:

```
expression1
while (expression2) {
    // instructions
    expression3
}
```

În cazul instrucțiunii **for**, oricare dintre cele 3 expresii poate lipsi. Lipsa expresiei condiționale este echivalentă cu o buclă infinită, cum ar fi:

```
for ( ; ; ) {
    /* instrucțiunile de aici sunt într-o buclă infinită */
}
```

În acest caz, ieșirea din buclă trebuie făcută explicit, cu ajutorul instrucțiunii **break**.

Exemplul următor prezintă un ciclu cu funcționalitate identică (tipărirea primelor 10 numere naturale), folosind cele 3 instrucțiuni repetitive:

```
int main() {
    short i;
```

```

printf("Ciclu for\n");

for (i = 1; i <= 10; i++) {
    printf("i=%d\n", i);
}

printf("Ciclu while\n");
i = 1;
while (i <= 10) {
    printf("i=%d\n", i);
    i++;
}

printf("Ciclu do while\n");
i = 0;
do {
    i++;
    printf("i=%d\n", i);
} while(i < 10);
}

```

Pentru blocuri de o singură instrucțiune (cum este și cazul instrucțiunii executate de for în exemplul de mai sus) nu este nevoie să folosim acolade. Totuși, folosirea acoladelor, chiar și în această situație, este recomandată pentru o depanare mai ușoară și o lizibilitate mai bună a programelor.

Instrucțiuni speciale

break

break, pe lângă utilizarea descrisă la instrucțiunea switch, poate fi folosită pentru a ieși forțat dintr-o instrucțiune de repetiție. Secvența următoare este echivalentă cu cele de mai sus:

```

i = 0;
for( ; ; ) {
    i++;
    if (i > 10) {
        break; /* ieșire forțată din buclă */
    }
    printf( "i=%d\n", i );
}

```

continue

continue forțează terminarea iterației curente a buclei și trecerea la iterația următoare. În cazul instrucțiunii for, acest lucru presupune executarea instrucțiunii de incrementare; apoi se evaluează condiția de continuare a buclei. Exemplul următor demonstrează implementarea unei bucle infinite cu ajutorul instrucțiunii continue:

```

for (i = 0; i < 10; ) {
    if (i == 0) {
        continue;
    }
    i++;
}

```

return

return este instrucțiunea de terminare a funcției curente. Aceasta poate fi apelată în forma **return;** în cazul funcțiilor care returnează **void** și în forma **return result;** pentru funcțiile care întorc o valoare.

goto

goto este o instrucțiune de salt a execuției. Instrucțiunea primește ca parametru o etichetă; următoarea instrucțiune executată după **goto** este cea de la eticheta dată.

```
int main() {
    goto et;
    printf("Asta nu apare la executie\n");

et:
    printf("Asta apare la rulare\n");
    return 0;
}
```

În majoritatea cazurilor, utilizarea instrucțiunii **goto** nu este recomandată și poate fi evitată folosind alte instrucțiuni de control și funcții. Programele care folosesc această instrucțiune pentru a sări între secvențe îndepărtate de cod sunt dificil de depanat și analizat.

Exerciții Laborator CB/CD

1. Primul exercitiu presupune modificarea/adaugarea de instructiuni unui cod pentru a realiza anumite lucruri. In momentul actual programul afiseaza daca un numar este prim, iar in caz contrar afiseaza divizorii nebanali.
 - Nu uitati ca trebuie sa utilizam un coding style [<http://ocw.cs.pub.ro/courses/programare/coding-style>] adecvat atunci cand scriem sursele.

ex1.c

```
#include <stdio.h>

int main(void)
{
    int nr, i;
    int prim = 1;

    scanf("%d", &nr);

    for (i = 2; i < nr; i++) {
        if (nr % i == 0) {
            printf("%d ", i);
            prim = 0;
        }
    }

    if (nr > 1 && prim) {
        printf("Numarul %d este numar prim", nr);
    }

    printf("\n");

    return 0;
}
```

Cerinte:

- Modificati codul astfel incat sa se afiseze doar daca un numar este prim sau nu.
- Trebuie sa adaugati o instructiune repetitiva astfel incat programul sa afiseze daca numerele din intervalul [1; n] sunt prime sau nu (pentru fiecare numar din interval se va verifica daca este prim sau nu).
- Sariti peste numar daca acesta are restul impartirii la 5 egal cu 4.
- Realizati urmatoarele actiuni in functie de restul impartirii numarului la 10:
 - Restul 1: Afisati numarul;
 - Restul 2: Afisati numarul / 10;

- Restul 3: Afisati daca numarul este prim sau nu;
- Restul 5: Cresteti limita for-ului cu 1.
- Daca restul impartirii numarului la 10 nu se afla in cazurile specificate atunci nu faceti nimic (prelucrati urmatorul numar)

Următoarele două probleme vă vor fi date de asistent în cadrul laboratorului.

Tasks laborator 3 [https://drive.google.com/drive/folders/1qB6EZLGVubKbuTXMtMue06egH_8fo25M]

Pentru utilizarea checkerului:

- Se va scrie cate un fisier sursa pentru fiecare problema;
- La finalul fiecarui printf utilizat pentru afisarea rezultatului trebuie sa existe un newline;
- Sursa nu trebuie sa contina alte printf-uri in afara de cele care scriu rezultatul asteptat la stdout.
- Se va dezarhiva arhiva specifica exercitiului;
- In directorul curent se afla checkerul, executabilul generat, folderele de input si output specifice problemei;
- Se va rula "bash checker.sh <executabil>" unde <executabil> este numele executabilului generat;

Probleme

1. [1.5p] Se citește de la tastatură un număr întreg si pozitiv N. Să se scrie un program care determină care număr cuprins între 2 si N are suma divizorilor nebanali maximă (adică printre divizori nu sunt considerate numerele 1 si N). Dacă există mai multe asemenea numere se va afișa numai primul dintre ele.

Exemplu

```
N=100
OUT: 96 //are suma divizorilor 155
```

2. [1.5p] De la tastatură se introduc mai multe numere întregi si pozitive, terminate printr-un număr negativ. După fiecare număr introdus, se va afișa lista divizorilor lui nebanali sau textul PRIM. La sfârșit se va afișa numărul de numere prime găsite.

Exemplu

```
35
OUT: 5 7
36
OUT: 2 3 4 6 9 12 18
17
OUT: PRIM
2
OUT: PRIM
12
OUT: 2 3 4 6
25
OUT: 5
53
OUT: PRIM
```

-4

OUT: S-au găsit 3 numere prime.

3. [2p] Fie funcția $f: [u, v] \rightarrow [\min, \max], f(x) = a * x^2 + b * x + c$, în care a, b, c, u și v sunt date. Determinați \min și \max , apoi rezolvați ecuația $f(x)=0$ în mulțimea numerelor reale. Afișați doar rădăcinile din intervalul $[u, v]$.
4. [2p] Scrieți un program care să convertească numere din baza 2 în baza 10 și invers. Numerele se introduc de la tastatură până la întâlnirea unui număr negativ. Pentru fiecare număr introdus, se va afișa pe o linie rezultatul.
5. [3p] Scrieți un program care verifică dacă un număr citit de la tastatură este palindrom. Un număr se consideră palindrom dacă citit invers este identic cu numărul inițial.

Bonus

1. [2p] De la tastatură se introduc N (fiind dat și el de la tastatură) numere citite pe rând. Găsiți cel mai mare divizor comun al acestor N numere, fără a folosi vectori.

Extra

- Cheatsheet [<https://github.com/cs-pub-ro/ComputerProgramming/blob/master/Laboratories/Lab3/cheatsheet.pdf>]

Referințe

- C - Control Statements [http://www.tutorialspoint.com/ansi_c/c_control_statements.htm]

programare/laboratoare/lab03.txt · Last modified: 2020/10/12 11:26 by george.muraru