Introducere în Programarea calculatoarelor
Laborator 2

Noțiuni introductive despre limbajul C

Obiectiv: fixarea noțiunilor fundamentale despre limbajul de programare C

Introducere în limbajul C

Limbajul de programare C a fost creat în anii '70 de către Dennis Ritchie în cadrul firmei Bell Telephone Laboratories (Bell Labs), având ca obiectiv principal dezvoltarea unui limbaj care sa fie utilizat pentru implementarea sistemului de operare UNIX. La vremea respectiva, existau limbaje de programare de nivel scăzut (apropiate de nivelul limbajului mașină al procesorului), precum si limbaje specializate de nivel înalt (FORTRAN, COBOL). Cele de nivel scăzut erau adecvate pentru dezvoltarea de sisteme de operare deoarece permiteau acces direct la resursele procesorului, dar aveau dezavantajul unei productivități extrem de scăzute a programatorului. Cele de nivel înalt permiteau dezvoltarea mai ușoară de aplicații specializate, dar ridicau numeroase probleme din punct de vedere al performantelor de execuție a programelor.

Acesta este motivul pentru care s-a căutat o cale de mijloc: dezvoltarea unui limbaj de programare de uz general, cu caracteristici de nivel înalt pentru îmbunătățirea productivității programatorilor, dar care, la nevoie, sa permită acces direct la resursele hardware ale sistemului de calcul, pentru a permite optimizarea foarte riguroasa din punct de vedere al vitezei de execuție a aplicațiilor unde acest criteriu este dominant. Astfel a luat naștere limbajul C, un limbaj de nivel mediu, care păstrează caracteristicile limbajelor de nivel înalt din care a fost derivat (B si BCPL), dar care reflecta foarte bine realitățile hardware: acces direct la adrese de memorie, operații la nivel de bit, accesul resurselor procesorului (regiștri), apel direct al funcțiilor puse la dispoziție de sistemul de operare.

Făcând referire la clasificările limbajelor de programare prezentate în laboratorul 1, limbajul C este un limbaj de nivel mediu, structurat, tipizat si compilat.

Fiind un limbaj structurat, limbajul C permite crearea de secvențe de instrucțiuni reunite sub un anumit nume, care efectuează o anumita operație clar definita si returnează un rezultat. Aceste secvențe de instrucțiuni se numesc funcții. Odată creata o funcție, ea poate fi referita (apelata) de oricâte ori este necesar, pe baza numelui acesteia. Spre deosebire de alte limbaje de programare însa, în C toate instrucțiunile trebuie sa apară numai în cadrul

unor funcții. Pe lângă posibilitatea de a returna un rezultat, o funcție are si posibilitatea de a primi anumiți parametri asupra cărora sa efectueze prelucrările.

Orice program scris în limbajul C trebuie sa conțină obligatoriu o funcție cu un rol special, funcția cu numele *main*. Aceasta este funcția de la care începe execuția programului în momentul lansării acestuia de către sistemul de operare.

În continuare este prezentat un exemplu de program care nu efectueaza nici o operatie, dar care respecta structura de baza pentru orice program C:

```
int main () { return 0; }
```

Prima linie din program declara o funcție cu numele *main*, care nu returnează nici un rezultat (cuvântul *void* dinainte de numele funcției) si nu primește nici un parametru (cuvântul *void* de după numele funcției). Simbolul '{' marchează începutul corpului funcției, iar simbolul '}' marchează finalul funcției.
Instrucțiunile care formează corpul funcției trebuie scrise între '{' si '}'.

Următorul exemplu de program afișează un mesaj ("Hello World") pe ecran:

```
#include <stdio.h>
int main ()
{
        /* apelam functia printf pentru afisare pe ecran */
        printf ("Hello World !");
        return 0;
}
```

Se observa ca si acest program conține o funcție *main*, dar în corpul funcției este apelata funcția *printf ()* pentru afișarea unui mesaj. In C, orice instrucțiune executabila (cum e cazul apelului la *printf()* trebuie urmata de **';'**. Funcția printf este o funcție din biblioteca standard pentru intrări și ieșiri a limbajului C, de aceea prima linie de program conține acum o directiva preprocesor, *#include* <*stdio.h*>, necesara pentru a putea apela funcții din biblioteca.

Tot în aceasta secvența de program se observa prezenta unui comentariu în interiorul funcției *main*:

/* apelam functia printf pentru afisare pe ecran */

În limbajul C, comentariile sunt texte precedate de secvența ' /* ' care se încheie cu ' */ '. Comentariile sunt ignorate în totalitate de către compilator.

Introducere în Programarea calculatoarelor
Laborator 2

Elemente de baza de programare în limbajul C

Orice program, indiferent de limbajul de programare în care este scris, preia un set de date de intrare, efectuează anumite prelucrări asupra acestora si generează niște rezultate (date de ieșire). În general, aceste prelucrări necesita însa memorarea temporara a unor rezultate intermediare. Pentru memorarea datelor de intrare, a rezultatelor intermediare si a datelor de ieșire, limbajul de programare trebuie sa permită declararea si utilizarea unor *variabile*. O variabila reprezintă o zona de memorie rezervata care se accesează printr-un nume asociat si care poate conține informații despre un anumit tip indicat în momentul declarării variabilei (operațiune care are ca efect si rezervarea memoriei). În general, conținutul unei variabile se poate citi si scrie. O variabila poate servi ca sursa de informații pentru diverse operații sau ca destinație a rezultatului altor operații efectuate de program asupra datelor. În limbajul C, o variabila se declara în felul următor:

```
tip_date nume_variabila;
```

tip_date reprezintă tipul de informații care se vor memora în acea variabila si este folosit de compilator în doua scopuri principale:

- determinarea dimensiunii în octeți a zonei de memorie care trebuie rezervata pentru acea variabila
- verificarea corectitudinii operațiilor care implica acea variabila (spre exemplu, daca declaram o variabila care sa conțină numere întregi, compilatorul ne va semnala eroare dacă încercam sa depozitam în acea variabila un șir de caractere).

În limbajul C, exista următoarele *tipuri de date predefinite*:

Tip	Dimensiune	Domeniu de valori
unsigned char	8 biți	0 ... 255
char	8 biți	-128 ... 127
enum	16 biți	-32,768 ... 32,767
unsigned int	16 biți	0 ... 65,535
short int	16 biți	-32,768 ... 32,767
int	16 biți	-32,768 ... 32,767
unsigned long	32 biți	0 ... 4,294,967,295
long	32 biți	-2,147,483,648 ... 2,147,483,647
float	32 biți	3.4 * (10^-38) ... 3.4 * (10^38)
double	64 biți	1.7 * (10^-308) ... 1.7 * (10^308)
long double	80 biți	3.4 * (10^-4932) ... 1.1 * (10^4932)

*nume_variabil*a reprezintă numele simbolic prin care va fi accesata acea zona de memorare. Numele unei variabile sau al unei funcții în C trebuie sa fie un identificator valid:

- poate sa înceapă numai cu o litera sau cu caracterul _ (underscore)
- după primul caracter, poate sa conțină numai litere, cifre sau _ (underscore)
- nu poate sa fie unul dintre cuvintele cheie rezervate ale limbajului: *auto break case char const continue default do double else enum extern float for goto if int long register return short signed sizeof static struct switch typedef typeid union unsigned using void volatile while* În C, variabilele se pot declara în doua locuri:
- în interiorul unei funcții, caz în care ele devin *variabile locale* si sunt accesibile doar din acea funcție; la încheierea execuției funcției variabilele se distrug automat;
- în afara oricărei funcții, caz în care ele devin *variabile globale* si sunt accesibile din orice funcție.

Limbajul C este *case-sensitive*, adică face distincție între literele mici si literele mari. Următorii identificatori sunt considerați diferiți:

Var1
var1
VAR1

O variabilă poate fi inițializată cu o anumita valoare în momentul declarării, cu sintaxa:

```
tip_date nume_var = valoare_initiala;
int a = 10;
char c = 'A';
```

În limbajul C, constantele numerice întregi se pot scrie în următoarele moduri:

- direct în forma zecimala (baza 10), daca nu sunt precedate de cifra 0: 1, 123
- în baza 16 (hexazecimala), precedate de 0x: 0x1A, 0xFFFF
- în baza 8 (octal), precedate de cifra 0: 017, 077

Constantele reale se introduc în forma zecimala sau notația exponențială: 123.1, 127.5E-4.

Constantele de tip caracter sunt incluse între doua simboluri apostrof: 'A', ':'. Constantele de tip sir de caractere sunt incluse între doua ghilimele: "șirul".

În cadrul constantelor de tip caracter sau sir de caractere, se pot introduce si caractere speciale de control, daca sunt precedate de caracterul *backslash*:

\n – are ca efect salt la linie noua când este afișat

\t – caracterul tab, 8 spatii

\r – revenire la începutul liniei curente

\\ - generează caracterul \ (backslash)

\xNN – generează caracterul ASCII cu codul NN (specificat în baza 16)

\" – generează caracterul ", folosit în cadrul constantelor de tip sir

După cum s-a văzut în al doilea exemplu, funcția printf se utilizează pentru afișarea unor mesaje pe ecran. Funcția permite si afișarea conținutului unor variabile, după cum rezulta din exemplul următor:

```
#include <stdio.h>
int main ()
{
        int v = 10;
        printf ("Valoarea variabilei este %d \n", v);
        return 0;
}
```

Acest program declara o variabila **v** de tip întreg, inițializata cu valoarea 10 si afișează conținutul
acestei variabile. În urma execuției, programul afișează următorul mesaj:

Valoarea variabilei este 10

Funcția *printf* se poate apela cu un număr variabil de parametri, dar primul parametru nu poate fi omis si trebuie sa fie un sir de caractere, numit s*ir de formatare*. Următorii parametri pot fi variabile de orice tip predefinit. În cadrul șirului de formatare se poate solicita afișarea conținutului unei variabile utilizând caracterul % urmat de un alt caracter, numit caracter de tip, care indica modul de afișare al conținutului variabilei. Variabilele sunt considerate în ordinea în care apar în lista de parametri, de la stânga la dreapta. Conținutul variabilei este introdus în șirul care se afișează pe poziția pe care apare caracterul % în șirul de formatare.

Valorile cele mai frecvent utilizate pentru caracterele de formatare sunt următoarele:

Secvența de formatare	Efect
%d	afişarea în baza 10 a unei variabile întregi (cu semn)
%u	afişarea în baza 10 a unei variabile întregi (fara semn)
%x	afişarea în baza 16 a unei variabile întregi (fara semn, litere mici)
%X	afişarea în baza 16 a unei variabile întregi (fara semn, litere mari)
%o	afişarea în baza 8 a unei variabile întregi (fara semn)
%f	afişarea în baza 10 a unui numar real, notatia zecimala
%e	afişarea în baza 10 a unui numar real, notatia exponentiala
%c	afişarea unui caracter ASCII
%s	afişarea unui şir de caractere ASCII
%%	afişarea caracterului '%'

Exercitiu: Rulati urmatorul program C si analizati mesajele afisate în urma executiei.

```
#include <stdio.h>
int main ()
{
   int v = 65;
  char c = 'A';
  printf ("Variabila v este %d (intreg zecimal) \n", v);
  printf ("Variabila v este %x (intreg hexa) \n", v);
  printf ("Variabila v este %o (intreg octal) \n", v);
 printf ("Variabila v este %c (caracter ASCII) \n", v);
 printf ("Variabila c este %d (intreg zecimal) \n", c);
 printf ("Variabila c este %c (caracter ASCII) \n", c);

return 0;
}
```

Se constată că aceeași variabilă, cu aceeași valoare, se poate afișa în diverse moduri, în funcție de necesități, utilizând caracterul corespunzător după %.

Următorul exemplu ilustrează posibilitatea de a afișa mai multe variabile cu funcția printf:

```
#include <stdio.h>
/* Programul schimba continutul a doua variabile întregi */
int main () {
        /* se pot declara mai multe variabile de acelasi tip pe o singura linie */
         int a = 10, b = 20, c;
        printf ("Inainte: prima este %d, a doua este %d \n", a, b); c
        = a;
        a = b;
        b = c;
       printf ("Dupa: prima este %d, a doua este %d \n", a, b);
return 0; }
```

Acest exemplu ilustrează si operatorul de atribuire '=', cu sintaxa generala :

```
dest = sursa;
```

si care are ca efect copierea valorii din sursa în destinație. Destinația trebuie sa fie o variabila. Sursa poate fi o variabila, o constanta sau o expresie compatibila ca tip cu destinația. Compatibilitatea de tipuri permite, spre exemplu, sa atribuim o valoare reala cu zecimale unei variabile întregi, caz în care compilatorul realizează automat trunchierea.

Funcții matematice

Limbajul C oferă o foarte mare libertate programatorului, libertate neîntâlnita în nici un alt limbaj de programare, aceasta fiind una dintre caracteristicile care l-au făcut atât de popular printre programatori. Aceasta libertate se reflecta, în special, în manipularea tipurilor, compilatorul limbajului C fiind mult mai puțin restrictiv în privința folosirii tipurilor de date (nu face verificări referitoare la compatibilitatea tipurilor). Pentru a putea folosi aceste funcții într-un program, este necesara includerea fișierului header MATH.H, folosind directiva **#include <math.h>**.

Câteva dintre funcțiile matematice folosite în limbajul C sunt următoarele:

- *sin* – calculează funcția sinus a unui unghi (dat în radiani);
- *cos* – calculează funcția cosinus a unui unghi (dat în radiani);
- *log* – calculează logaritmul natural al unui număr;
- *log10* – calculează logaritmul în baza 10 al unui număr;
- *sqrt* – calculează radicalul dintr-un număr;
- *pow* – calculează funcția sinus a unei valori (a unui unghi);
- *random* – returnează o valoare aleatoare între “0” si “parametrul dat – 1”; necesita

fișierul header STDLIB.H;

Nr. Crt	Funcția	Tipul returnat	Tipul parametrului	Observații
1	sin	double	double	valoarea se specifică în radiani
2	cos	double	double	valoarea se specifică în radiani
3	log	double	double	
4	log10	double	double	
5	sqrt	double	double	parametru pozitiv
6	pow	double	double, double	
7	random	int	int	

Funcții matematice

Funcțiile printf () si scanf ()

Funcția printf()

Funcția *printf* afișează pe ecran valorile din lista de argumente, conform formatului specificat. Șirul format poate conține caractere ordinare, care se vor afișa ca atare, si descriptori de format prefixați de caracterul %. Un descriptor de format poate conține în ordine următoarele:

- un semn minus care indica alinierea la stânga în cadrul formatului a valorii afișate;
- un număr care specifica lungimea minima a câmpului de afișare;
- un punct care separa lungimea câmpului de afișare de precizia de afișare (de ex. numărul de zecimale pentru valorile reale);

Exemplu:

```
int nr1=3,nr2=4;
printf("%-5d+%5d=%d",nr1,nr2,nr1+nr2);
```

În exemplul de mai sus nr1 si nr2 se vor afișa pe 5 spatii, nr1 va fi aliniat la stânga iar nr2 va fi aliniat la dreapta.

Exemplu:

```
float r=1.12345;
 printf(”%.3f”,r);
```

În exemplul de mai sus va fi afișat cu 3 zecimale.

Introducere în Programarea calculatoarelor
Laborator 2

Funcția scanf()

Funcția *scanf* citește date de la tastatura, conform formatului, si înscrie valorile citite la adresele specificate. Primul sau argument este un sir de control care conține formatele corespunzătoare interpretării șirurilor de intrare. Următoarele argumente sunt *adresele variabilelor*. Operatorul "&" returnează adresa memoriei unei variabile.

Exemplu:

```
scanf("%d", &x);
```

Formatul %d implica interpretarea caracterelor citite ca un întreg zecimal, urmata de memorarea valorii variabilei la adresa precizata prin &x.

Exemplu1: adunarea a doua numere reale, citite de la tastatura:

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

int main (void)
{
        float a, b, c;
        printf ("Introduceti primul nr: ");
        scanf ("%f",&a);
        printf ("Introduceti cel de-al doilea nr: ");
        scanf ("%f",&b);
        c = a + b;
        printf ("Suma celor doua numere este %f ", c);
        getch();
return 0;
}
```

Exemplu 2: calcularea ariei unui triunghi cu laturile citite de la tastatură

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

int main()
{
        float a, b, c, p, s;
        printf("a= ");
        scanf("%f", &a);
        printf("\nb= ");
        scanf("%f", &b);
        printf("\nc= ");
        scanf("%f", &c);

        p = (a + b + c) / 2;
```

```
        s = sqrt((p*(p - a)*p - b)*(p - c));
        printf("\nAria triunghiului este: %.2f \n", s);      // cu 2 zecimale dupa virgula
        getch();
return 0;
}
```

Probleme propuse

1. Să se scrie un program care sa citească de la tastatura numele si vârsta dvs. si sa le afișeze.
2. Sa se scrie un program C care afișează produsul a doua variabile întregi, citite de la tastatură.
3. Sa se scrie un program C care convertește un unghi din grade în radiani *(rad=grad*pi/180).*
4. Sa se scrie un program C care face conversia din grade Celsius în grade Fahrenheit, *C = (F – 32)* 5/9.*
5. Sa se scrie un program C care afișează cifra unitarilor unei variabile de tip întreg.
6. Sa se scrie un program care citește de la tastatura lungimile laturilor unui triunghi si afișează aria acestuia, calculata cu formula lui Heron.