

Despre limbajul C

Programarea calculatoarelor

Introducere

Marius Minea

26 februarie 2008

Programarea calculatoarelor. Curs 1

Marius Minea

- dezvoltat și implementat în 1972 la AT&T Bell Laboratories de Dennis Ritchie <http://cm.bell-labs.com/cm/cs/who/dmr/chist.html>
- nevoia unui limbaj pentru scrierea de sisteme de operare și utilitare (strâns legat de *sistemul de operare UNIX* dezvoltat la Bell Labs)
- C dezvoltat inițial sub UNIX; în 1973, UNIX rescris în totalitate în C
- cartea de referință: Brian Kernighan, Dennis Ritchie: [The C Programming Language](#) (1978)
- 1988 (K&R ediția II) limbajul a fost standardizat de ANSI (American National Standards Institute) – versiunea numită ANSI C
- versiunea curent: C99 (standard ISO 9899)

De ce folosim C?

- foarte versatil: acces la reprezentarea binară a datelor, mare libertate în lucrul cu memoria, bună interfață cu hardware
- limbaj matur, bază mare de cod (biblioteci pt. multe scopuri)
- eficient: compilatoare bune, generează cod compact, rapid

Programarea calculatoarelor. Curs 1

Marius Minea

Programarea calculatoarelor. Introducere

3

Calcul, funcții și programe

Primul rol al programelor: de a efectua *calcul* (matematice)

În matematică, efectuăm calcule cu ajutorul *funcțiilor*:

- *cunoaștem* diverse funcții (sin, cos, etc.)
- *definim* funcții noi (depinzând de problemă)
- *combinăm* funcțiile existente și definite de noi
- și le *folosim* într-o anumită ordine

Toate aceste aspecte le întâlnim și în programare

Programarea calculatoarelor. Curs 1

Marius Minea

Programarea calculatoarelor. Introducere

4

Funcții în matematică și în C

Exemplu: funcția de ridicare la pătrat pentru întregi

```

sqr : ℤ → ℤ
sqr(x) = x · x

int sqr(int x)
{
    return x * x;
}

```

Definiția unei funcții conține:

- *antetul* funcției: specifică un domeniu de valori (întregi), numele funcției și parametrii acesteia (un singur parametru, întreg)
- *corpul* funcției: aici, o singură *instrucțiune* (`return`) cu o *expresie* care dă valoarea funcției (pornind de la parametri)

Limbajul are *reguli* precise de scriere (*sintaxa*):

- diversele elemente scrise într-o anumită *ordine*;
- se folosesc *separatori* pentru a le delimita precis: () ; { }

Programarea calculatoarelor. Curs 1

Marius Minea

Programarea calculatoarelor. Introducere

5

O a doua funcție

Ridicarea la pătrat pentru numere *reale*

```

sqr_f : ℝ → ℝ
sqr_f(x) = x · x

float sqr_f(float x)
{
    return x * x;
}

```

- o altă funcție decât cea dinainte: alt domeniu de definiție și de valori
- trebuie să-i dăm alt nume dacă o folosim în același program
- strict vorbind și operația * e alta, fiind definită pe altă mulțime

Cuvintele `int`, `float` denotă *tipuri*.

Un *tip* e o *mulțime de valori* împreună cu un *set de operații* permise pentru aceste valori.

Programarea calculatoarelor. Curs 1

Marius Minea

Programarea calculatoarelor. Introducere

6

Întregi, reali și operații matematice

Există diferențe importante între tipuri numerice în C și matematică.

- în matematică, $\mathbb{Z} \subset \mathbb{R}$, ambele sunt infinite, \mathbb{R} e densă
- în C, `int` și `float` sunt tipuri finite; realii au precizie finită
- *Constantele* numerice au tip determinat de modul de scriere:
 - 2 e un întreg, 2.0 e un real
 - putem scrie un real în notație științifică: 1.0e-3 în loc de 0.001
 - sunt echivalente scrierile 1.0 și 1. respectiv 0.1 și .1
- unele operații sunt diferite pentru întregi și reali:

Împărțirea întreagă e *împărțire cu rest* !!!

- 7 / 2 dă valoarea 3, pe când 7.0 / 2.0 dă valoarea 3.5
- 7 / 2 dă valoarea -3, deci la fel cu - (7 / 2)

Operatorul *modulo* (scris `%`) e definit doar pentru întregi.

- 9 / 5 este 1 9 % 5 este 4 9 / -5 este -1 9 % -5 este 4
 - 9 / 5 este -1 -9 % 5 este -4 -9 / -5 este 1 -9 % -5 este -4
- semnul restului e același cu semnul de împărțitului

e valabilă egalitatea $a == a / b * b + a \% b$ (ecuația împărțirii cu rest)

Programarea calculatoarelor. Curs 1

Marius Minea

Puțină terminologie

– *cuvinte cheie*: au un înțeles predefinit (nu poate fi schimbat) exemple: instrucțiuni (`return`), tipuri (`int`, `float`), etc.

– *identificatori* (de ex. `sqr`, `x`) aleși de programator pentru a denumi funcții, parametri, variabile, etc.

Un identificator e o secvență de caractere formată din litere (mari și mici), liniuța de subliniere `_` și cifre, care nu începe cu o cifră și nu este un cuvânt cheie

Exemple: `x3`, `a12_34`, `_exit`, `main`, `printf`, `int16_t`

– *constante* (numerice: `-2`, `3.14`; mai târziu: caractere, șiruri)

– *semne de punctuație*, cu diverse semnificații:

* e un operator

; delimitează sfârșitul unei instrucțiuni

parantezele () grupează parametrii unei funcții sau o subexpresie

acoladele { } grupează instrucțiuni sau declarații etc.

Funcții cu mai mulți parametri

Exemplu: discriminantul ecuației de gradul II: $a \cdot x^2 + b \cdot x + c = 0$

```
float discrim(float a, float b, float c)
{
    return b * b - 4 * a * c;
}
```

Între parantezele rotunde () din antetul funcției putem specifica oricâți parametri, fiecare cu tipul propriu, separați prin virgulă.

Apelul de funcție

Până acum, am *definit* funcții, fără să le folosim.

Valoarea unei funcții poate fi folosită într-o expresie cu aceeași sintaxă ca și în matematică: *funcție(parametru, parametru, ..., parametru)*

Exemplu: în discriminantul dinainte, puteam scrie:

```
return sqrtf(b) - 4 * a * c;
```

Sau putem defini:

```
int cube(int x)
{
    return x * sqrt(x);
}
```

IMPORTANT: înainte de a folosi orice identificator (nume) în C, el trebuie să fie *declarat* (trebuie să știm ce reprezintă)

⇒ Exemplele sunt corecte dacă `sqrtf` respectiv `sqr` sunt definite *înainte* de `discrim`, respectiv `cube` în program.

Un prim program C

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

– cel mai mic program: nu face nimic !

– orice program conține funcția *main* și e executat prin apelarea ei (programul poate conține și alte funcții)

– în acest caz: funcția nu are parametri (*void*)

void e un cuvânt cheie pentru tipul vid (fără nici un element)

– cf. standard: `main` returnează un cod întreg către sistemul de operare (convenție: 0 pt. terminare cu succes, \neq 0 pt. cod de eroare)

Un program comentat

```
/* Acesta este un comentariu */
int main(void) // comentariu pana la capat de linie
{
    /* Acesta e un comentariu pe mai multe linii
       obisnuit, aici vine codul programului */
    return 0;
}
```

– programele pot conține comentarii, înscrise între `/*` și `*/`

sau începând cu `//` și terminându-se la capătul liniei

– orice conținut între aceste caractere nu are nici un efect asupra generării codului și execuției programului

– programele *trebuie* comentate

– pentru ca un cititor să le înțeleagă (alții, sau noi, mai târziu)

– ca documentație și specificație: funcționalitate, restricții, etc.

– ce reprezintă parametrii funcțiilor, rezultatul, variabilele,

ce condiții trebuie îndeplinite, cum se comportă la eroare

Tipărirea (scrierea)

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

```
int main(void)
{
    printf("hello, world!\n"); // tipareste un text
    return 0;
}
```

– `printf` (de la "print formatted"): o funcție standard

(N.B.: `printf` nu este *instrucțiune* sau *cuvânt cheie*)

– e apelată aici cu un parametru șir de caractere

– constantele șir de caractere: incluse între ghilimele " "

– `\n` este notația pentru caracterul de linie nouă

– prima linie e o *directivă de preprocesare*, include fișierul `stdio.h`

cu *declarațiile* funcțiilor standard de intrare / ieșire

– *declarația* = informațiile (nume, parametri) necesare pentru folosire

– *implementarea* (codul obiect, compilat): într-o bibliotecă din care

compilatorul ia cele necesare pentru generarea programului executabil

Tipărirea unei valori numerice

```
#include <math.h>          #include <stdio.h>
#include <stdio.h>        int sqr (int x) { return x * x; }
int main(void)           int main(void)
{
    printf("cos(0) = ");   printf("2 ori -3 la patrat e ");
    printf("%f", cos(0));  printf("%d", 2 * sqr(-3));
    return 0;             return 0;
}
```

Pentru a tipări valoarea unei expresii, `printf` ia două argumente:

– un șir de caractere (specificator de format):

`%d` (întreg, *decimal*), `%f` (real, *floating point*)

– expresia, al cărei tip trebuie să fie compatibil cu cel indicat (verificarea cade în sarcina programatorului !!!)

Secvențierea: instrucțiunile unei funcții se execută *una după alta*

– excepții: instrucțiunea `return` încheie execuția funcției

(după ea nu se mai execută nimic)

Funcții definite pe cazuri

$$abs : \mathbb{Z} \rightarrow \mathbb{Z} \quad abs(x) = \begin{cases} x & x \geq 0 \\ -x & \text{altfel} \end{cases}$$

Cu cele discutate până acum, nu putem defini această funcție în C.

Valoarea funcției nu e dată de o singură expresie, ci de una din două expresii diferite (x sau $-x$), în funcție de o condiție ($x \geq 0$ sau nu)

⇒ e necesară o facilitate de limbaj pentru a *decide* valoarea pe care o ia o expresie în funcție de valoarea unei condiții (adevărat/fals)

Operatorul condițional ? : în C

O **expresie condițională** în C are sintaxa: *condiție ? expr1 : expr2*
– dacă condiția e adevărată, se evaluează doar *expr1*, și întreaga expresie ia valoarea acesteia

– dacă e falsă, se evaluează doar *expr2* și întreaga expresie ia valoarea acesteia

```
int abs(int x)
{
    return x >= 0 ? x : -x;    // operator minus unar
}
```

Operatori de comparație în C: == (egalitate), != (diferit), <, <=, >, >=

IMPORTANT! Testul de egalitate în C e == și nu = simplu !!!

Obs.: Funcția `abs` există ca funcție standard, declarată în `stdlib.h`

Funcții definite pe mai mult de două cazuri

$$sgn : \mathbb{Z} \rightarrow \{-1, 0, 1\} \quad sgn(x) = \begin{cases} -1 & x < 0 \\ 0 & x = 0 \\ 1 & x > 0 \end{cases}$$

Chiar cu operatorul condițional nu putem transcrie funcția direct în C (el permite doar decizia cu două ramuri (adevărat/fals), nu cu un număr mai mare de condiții / ramuri)

⇒ trebuie să descompunem calculul funcției `sgn` (de fapt decizia asupra valorii parametrului x)

– **descompunerea în subprobleme** mai mici: principiu *foarte important* în rezolvarea de probleme

Rescriem funcția cu o singură decizie în fiecare punct:

$$sgn(x) = \begin{cases} \text{dacă } x < 0 & -1 \\ \text{altfel } (x \geq 0) & \begin{cases} \text{dacă } x = 0 & 0 \\ \text{altfel } (x > 0) & 1 \end{cases} \end{cases}$$

Scrierea unei funcții pe mai multe cazuri în C

$$sgn(x) = \begin{cases} \text{dacă } x < 0 & -1 \\ \text{altfel } (x \geq 0) & \begin{cases} \text{dacă } x = 0 & 0 \\ \text{altfel } (x > 0) & 1 \end{cases} \end{cases}$$

```
int sgn (int x)
{
    return x < 0 ? -1
           : x == 0 ? 0 : 1;
}
```

– putem grupa arbitrar de mulți operatori ? :

– *expr1* și *expr2* pot fi la rândul lor expresii condiționale

– într-o expresie scrisă corect, un : corespunde univoc unui ?