

Utilizarea și programarea calculatoarelor

Introducere

Marius Minea

5 octombrie 2004

Ce este un program

= o secvență de instrucțiuni care comandă execuția calculatorului

- program executabil: cod mașină interpretabil direct de calculator
- program sursă: în limbaj inteligibil de programatorul uman

Traducerea din format sursă în format executabil:

- *compilare*: anterior rulării programului (pt. C, C++, PASCAL)
- *interpretare*: direct la rulare: (pt. variante de BASIC, LISP)

În mod tipic, un program generic:

- citește *datele de intrare*
- efectuează *prelucrări* (calculare) asupra lor
- produce niște *rezultate* la ieșire

Funcționarea arhitecturii von Neumann

John von Neumann (1945)

- propune arhitectura menționată (control, UAL, memorie, I/O)
 - și conceptul de *program memorat* (deosebirea esențială, de ex. față de calculatorul de buzunar, acționat direct de utilizator)
- Acestea stau la baza tuturor calculatoarelor convenționale.

Funcționarea:

1. citește instrucțiunea de la adresa din numărătorul de program
2. înaintează numărătorul de program la următoarea instrucțiune
3. unitatea de control decodifică instrucțiunea și comandă operația (care poate modifica regiștri, memoria, numărătorul de program)
4. se reia ciclul de la punctul 1.

Scopul cursului

Utilizarea calculatoarelor

- familiaritate cu sisteme PC, lucru cu fișiere, medii de programare
- în principal la laborator

Programarea calculatoarelor

- de la exemple la programe pentru situații reale
- programarea = dezvoltarea unui produs software, de la A la Z
- buna cunoaștere a unui limbaj de programare și un punct de plecare pentru altele
- principii și stil de programare

Arhitectura unui calculator

- *unitate centrală* de prelucrare (CPU)
 - unitate de control
 - unități aritmetice și logice pentru calcul
- *memorie*
 - primară: circuite integrate
 - secundară: medii magnetice (disc fix, floppy), optice (CD)
- *echipamente periferice* (dispozitive de intrare/ieșire)
 - tastatură, mouse, ecran, imprimantă, joystick

Sisteme de operare

- *gestionează resursele* unui sistem de calcul (timpul de procesare al unității centrale, memoria, perifericele, sistemul de fișiere)
- creează o interfață *independentă de hardware*
- oferă *apeluri sistem* utilizate din limbaje de programare (alocare de memorie, citire, tipărire)

Vom programa: sub Windows în semestrul I, sub Linux în semestrul II
Urmărim: scrierea de programe *portabile*, independent de sistemul de operare și mediul de programare folosit.

Ciclul de dezvoltare al unui program

- Definirea și analiza specificațiilor
- Proiectare
- Implementare (Codare) ← **doar o parte !!!**
- Testare
- Mentenanță

Dezvoltarea programului

Cerințe și specificații

La curs: reprezentate de enunțul problemei. În realitate însă:

- adesea cea mai dificilă parte
- trebuie eliminate ambiguitățile
- neînțelegerile au efecte pentru tot restul proiectului
- important: nu "ce știu eu să fac" ci "ceea ce se cere" (de client)

Proiectarea soluției

- arhitectura programului
- împărțirea în componente și interfața între ele
- proiectarea structurilor de date
- proiectarea algoritmilor
- interfața cu utilizatorul

Corectitudinea programului

Raționament logic

- ce face programul ? pot defini în mod precis ? (absolut necesar!)
- pot găsi o relație matematică ?
- pot urmări pas cu pas transformarea pe care o efectuează programul și să demonstrez (să mă conving) astfel că rezultatul final e corect ?
- ce se schimbă pe parcursul programului ? ce rămâne neschimbat ? (exemplu: folosirea invarianților în raționamentul despre cicluri)
- urmărirea raționamentului în faza de implementare reduce erorile

Testare

- ce presupuneri/garanții există despre intrare ?
- ce presupuneri/garanții există despre alte module de program ?
- cum se comportă programul: pentru date normale, limită, eronate care e performanța pt. date de dimensiuni mari ?

Dincolo de program

Documentarea

- complexitatea sistemului crește pe măsura realizării
- documentație necesară pentru:
 - descriere exactă a funcționalității sistemului, împreună cu toate cerințele, restricțiile, presupunerile
 - comunicarea dintre programatori (chiar pt. programatorul inițial!)
 - proiectarea de teste, evoluția și mentenanța ulterioară

Reutilizare. Portabilitate

- cât de generală este soluția ? poate fi reutilizată ?
- se pot folosi elemente existente ? (funcții de bibliotecă, module de program, obiecte, etc.)

Modificare și mentenanță

- programul e proiectat pentru a fi întreținut ușor ?

Securitate

- robustețe, rezistență la date de intrare invalide

Organizarea cursului

- 2 ore de curs
- 1 ora de seminar (2 la 2 săptămâni): prep. ing. Gabriela Bobu
- 2 ore de laborator: prep. ing. Elena Doandș

Evaluare

- 60% examen
 - 1/2 parțial (30%), 1/2 final (30%)
- 40% activitate pe parcurs (30% laborator, 10% seminar)

Consultații: la birou (B 531)

- o oră fixă pe săptămână (liberă în orar): joi 8-10 ?
- sau stabiliți o altă ora prin e-mail (marius@cs.utt.ro)

Pagina de curs: la <http://www.cs.utt.ro/~marius/curs/upc>

Important: Onestitate

Scopul cursului: *fiecare* din voi să programați bine în C
⇒ laboratorul și examenul evaluează rezultatele *fiecăruia dintre voi* (nu colectiv!)

DA:

- consultați cadrele didactice în caz de nelămuriri
- învățați împreună

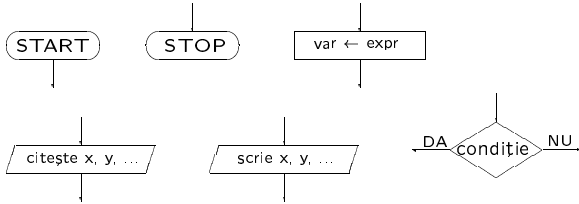
NU:

- prezentați soluțiile altora (modificate sau nu) ca ale voastre

Principiu (nu numai la acest curs): *orice sursă folosită trebuie citată* (cărți, articole, pagini de web, idei ale altora)

Algoritmi și scheme logice

algoritm = secvență *finită* de pași pentru rezolvarea unei probleme
schemă logică: reprezentarea grafică a unui algoritm
 – fără particularitățile unui anumit limbaj de programare, dar precis
 Blocuri (instrucțiuni) componente în scheme logice:
 start, stop, atribuire, citire, scriere, decizie



schemă logică: graf format din cele 6 tipuri de instrucțiuni, cu un singur nod de START și unul singur de STOP

Istoricul limbajului C

- dezvoltat și implementat în 1972 la AT&T Bell Laboratories de Dennis Ritchie <http://cm.bell-labs.com/cm/cs/who/dmr/chist.html>
- limbaj de *programare structurată* (blocuri, cicluri, funcții) (concept apărut în ALGOL 60, apoi ALGOL 68, PASCAL, ...)
- necesitatea unui limbaj pentru *programe de sistem* (legătură strânsă cu *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)
- în 1988 (vezi K&R ediția II) limbajul a fost standardizat de ANSI (American National Standards Institute)
- dezvoltări ulterioare: C99 (standard ISO 9899)

Caracteristici ale limbajului C

- limbaj de nivel *mediu*: oferă tipuri, operații, instrucțiuni simple fără facilitățile complexe ale limbajelor de nivel (foarte) înalt (nu: tipuri mulțime, concatenare de șiruri, etc.)
- limbaj de programare *structurat* (funcții, blocuri)
- permite programarea *la nivel scăzut*, apropiat de hardware
 acces la reprezentarea binară a datelor
 mare libertate în lucrul cu memoria
 foarte folosit în programarea de sistem, interfața cu hardware
- produce un cod *eficient* (compact în dimensiune, rapid la rulare)
 apropiat de eficiența limbajului de asamblare
 datorită caracteristicilor limbajului, și maturității compilatoarelor
- *slab tipizat* → necesită mare atenție în programare
 conversii implicite și explicite între tipuri, `char` e tip întreg, etc.

Un prim program C

```
void main(void)
{
}
```

- cel mai mic program: nu face nimic !
- pornind de la el, scriem orice program, adăugând cod între { și }
- orice program conține funcția *main* și e executat prin apelarea ei (programul poate conține și alte funcții)
- în acest caz: funcția nu returnează nimic (primul `void`), și nu are parametri (al doilea `void`)
- Vom discuta: `main` poate lua și argumente, și returna un `int`

Un program comentat

```
/* Acesta este un comentariu */
void main(void) // comentariu până la capăt de linie
{
    /* Acesta e un comentariu pe mai multe linii
       obisnuit, aici vine codul programului */
}
```

- programele pot conține *comentarii*, înscrise între /* și */ sau începând cu // și terminându-se la capătul liniei (ca în C++)
- 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.

Să scriem ceva!

```
#include <stdio.h>

void main(void)
{
    printf("hello, world!\n"); /* tipăreste un text */
}
```

- prima linie: obligatorie pentru orice program care citește sau scrie
 = o *directivă de preprocesare*, include fișierul `stdio.h` care conține declarațiile funcțiilor standard de intrare/ieșire – adică informațiile (nume, parametri) necesare compilatorului pt. a le folosi corect
- `printf` ("print formatted"): o *funcție standard* implementată într-o bibliotecă care e inclusă (linkeditată) la compilare
- N.B.: `printf` *nu* este o instrucțiune sau cuvânt cheie
- e apelată aici cu un parametru șir de caractere
- șirurile de caractere: incluse între ghilimelele duble "
- `\n` este notația pentru caracterul de linie nouă.

```
void main(void)
{
    int sum; /* declarăm o variabilă întreagă */
    int a, b; /* declarăm încă două variabile întregi */

    a = 2;
    b = 3;
    sum = a + b; /* semnul de atribuire în C este = */
}
```

- pentru a memora data și calcula, avem nevoie de *variabile* o variabilă are un *nume*, un *tip* și o *valoare*
- o variabilă trebuie *declarată* (cu tipul ei) înainte de folosire
- câteva tipuri standard: caracter `char`, întreg `int`, real `float`
- corpul unei funcții formează un *bloc*, între `{` și `}`
- blocul poate conține *declarații* și o *secvență* de instrucțiuni

Utilizarea și programarea calculatoarelor. Curs 1

Marius Minea

```
#include <stdio.h>
void main(void)
{
    int x;

    x = 5;
    printf("Numarul x are valoarea: ");
    printf("%d", x);
}
```

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

- un șir de caractere (*specificator de format*):
`%c` (caracter), `%d` (întreg), `%s` (șir), etc.
- expresia, al cărei tip trebuie să fie compatibil cu cel indicat (verificarea cade în sarcina programatorului !!!)

Utilizarea și programarea calculatoarelor. Curs 1

Marius Minea

```
#include <stdio.h>
void main(void)
{
    int x;

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

- `scanf`: funcție de citire formatată, perechea lui `printf`
- primul argument (șirul de format) la fel ca la `printf`
- deosebirea: înainte de numele variabilei apare operatorul `&` (adresă) transmitând explicit *adresa* lui `x`, `scanf` știe unde să pună valoarea

Utilizarea și programarea calculatoarelor. Curs 1

Marius Minea

```
#include <stdio.h>
void main(void)
{
    int a, b, sum;

    printf("Introduceți un număr: ");
    scanf("%d", &a); /* numărul se citește în variabila a */
    printf("Introduceți alt număr: ");
    scanf("%d", &b);
    sum = a + b;
    printf("Suma este %d\n", sum);
}
```

Obs.: `\n` este caracterul de linie nouă.

Utilizarea și programarea calculatoarelor. Curs 1

Marius Minea

```
#include <stdio.h>
void main(void)
{
    int x;

    printf("Introduceți un număr: ");
    scanf("%d", &x);
    if (x < 0) {
        printf("x este negativ");
    } else {
        printf("x este nenegativ");
    }
    if (x == 0) printf("x este zero");
}
```

Utilizarea și programarea calculatoarelor. Curs 1

Marius Minea

Formatul:

```
if ( expresie logică )
    instrucțiune
else
    instrucțiune
```

- ramura `else` este opțională
- instrucțiunile din ramuri pot fi compuse (blocuri `{ }`)
- N.B.: NU CONFUNDAȚI în limbajul C
`=` este operatorul de atribuire
`==` este operatorul test de egalitate
- operatori logici: `==`, `!=`, `<`, `>`, `<=`, `>=`

Utilizarea și programarea calculatoarelor. Curs 1

Marius Minea