

# Utilizarea și programarea calculatoarelor

Introducere

Marius Minea

2 octombrie 2003

## 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

## 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
- *interpretare*: direct la rulare

În mod tipic, un program generic:

- citește *datele de intrare*
- efectuează calcule/prelucrări asupra lor
- produce niște *rezultate* de ieșire

## 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

## Funcționarea arhitecturii von Neumann

---

John von Neumann (1945)

- propune arhitectura menționată (control, UAL, memorie, I/O)
- și conceptul de *program memorat*

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.

## Sisteme de operare

---

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

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: 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 invariantilor 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): ing. Radu Fericean
- 2 ore de laborator: prep. ing. Loredana Ștefănuț

### 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 12-14 ?
- 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: ca *fiecare* din voi să ajungeți să programați bine  
⇒ cursul va evalua rezultatele *fiecăruia dintre voi*

### DA:

- consultați cadrele didactice în caz de nelămuriri
- învățați împreună

### NU:

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

Principiu de bază: orice sursă folosită trebuie citată

## Algoritmi și scheme logice

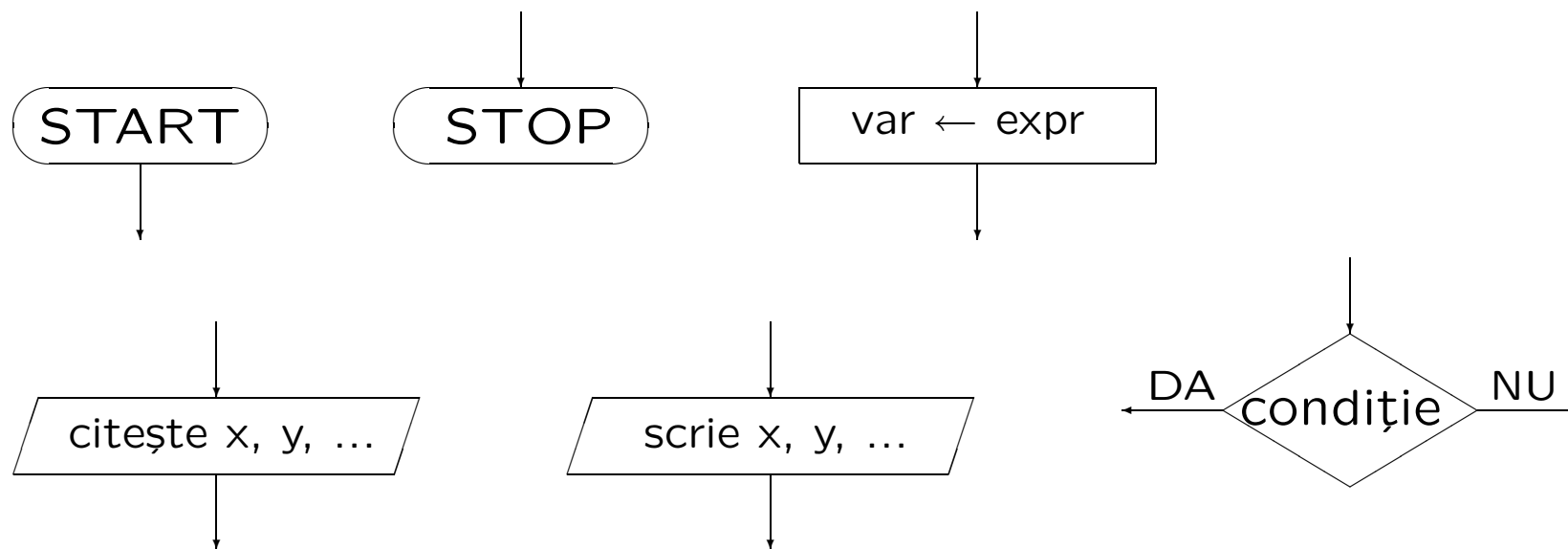
**algorithm** = secvență finită de pași pentru rezolvarea unei probleme

**schemă logică**: reprezentarea grafică a unui algorithm

– 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
- contextul: evoluția conceptului de programare structurată (atât Pascal cât și C inspirate de ALGOL 68)
- 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
- limbaj de programare *structurat*
- 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

## 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 (vom vedea mai târziu că poate conține și alte funcții)
- în acest caz: funcția nu returnează nimic (primul `void`), hsc nu are parametri (al doilea `void`)



## Un program comentat

---

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

- programele pot conține *comentarii*, înscrise între `/*` și `*/`
- orice conținut între aceste caractere nu are nici un efect
- programele trebuie comentate
  - pentru ca un cititor sa 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!"); /* 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)
- `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 ghilimele duble "

```
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

## Să tipărim un număr

---

```
#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 !!!)

## Să citim un număr

---

```
#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) (adresa variabilei, adică locul unde va fi memorată valoarea; detalii ulterior)

## ○ combinație: citire, calcul, tipărire

---

```
#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.

## Să luăm o primă decizie

---

```
#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");
}
```

## Instrucțiunea de decizie `if`

---

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: `==`, `!=`, `<`, `>`, `<=`, `>=`