Programarea Calculatoarelor

Curs 6

Iulian Năstac

Recapitulare din cursul precedent

Tipuri de date de bază

- În limbajul *C* există 5 tipuri de date de bază:
  - *char* – caracter
  - *int* – întreg
  - *float* – virgulă mobilă
  - *double* – virgulă mobilă cu dublă precizie
  - *void* – nu definește o valoare

- *specificatorii de conversie*:
  - signed,
  - unsigned
  - short
  - long

Recap.

Variabile

- **Variabilele** sunt pur și simplu denumirile unor valori ce pot suferi modificări pe parcursul unui program.

- Tipuri de variabile:
  - Variabile locale
  - Variabile globale

Recap.

O clasificare mai specializată defineşte următoarele tipuri de variabile:

- **Variabile automatice**
  - O variabilă automată este o variabilă definită în interiorul unui bloc funcțional
  - sunt alocate și dezalocate în mod automat atunci când fluxul de program intră și iese din blocul unde a fost definită variabila
- **Variabilele externe**
  - variabilă este definită în afara oricărui bloc
- **Variabilele locale statice**
  - variabile care au fost alocate static - a căror existență/valabilitate se extinde pe întreaga durata a programului
- **Variabilele registru**
  - alocare în regiștri este procesul de atribuire a unui număr de variabile de program direct în regiștrii procesorului

Recap.

Rutine standard de intrare şi ieşire

- Rutinele standard de intrare şi ieşire sunt funcţii care suplinesc lipsa instrucţiunilor de intrare şi ieşire.

- Acestea se regăsesc în diferite biblioteci de headere (în special *stdio.h*, *conio.h și stdlib.h*).

Sinteza rutinelor de bază intrare – ieşire

IEŞIRI	INTRĂRI
printf	scanf
puts	gets
putchar	getchar
putch	getch getche

Recap.

Expresii

- O expresie într-un limbaj de programare este o combinație de valori explicite, constante, variabile, operatori și funcții care sunt interpretate în conformitate cu normele specifice de prioritate și de asociere pentru un anumit limbaj de programare. Programul calculează expresia și produce o nouă valoare.
- Acest proces asupra unei expresii se numește evaluare.
- Valoarea unei expresii poate fi de diferite tipuri, cum ar fi numerică, șir de caractere, sau logică.

Operatori în C

- Limbajele de programare utilizează un set de operatori cu o sintaxă proprie.

- Limbajul de programare C conține un număr fix de operatori predefiniți.

1	**() [] -> . ::**	grupare, matrici, acces
2	**! ~ - + * & sizeof *type cast* ++x - -x**	Operatori unari, sizeof și casts
3	*** / %**	Multiplicare, împărțire, modulo
4	**+ -**	Adunare și scădere
5	**<< >>**	Deplasare stânga și dreapta
6	**< <= > >=**	Comparație inegalitate
7	**== !=**	Comparație egalitate
8	**&**	ȘI logic (pe biți)
9	**^**	SAU exclusiv (pe biți)
10	**|**	SAU (pe biți)
11	**&&**	ȘI logic
12	**||**	SAU logic
13	**? : = += -= *= /= %= &= |= ^= <<= >>=**	Expresia condițională și operatorii de atribuire
14	**,**	Operatorul virgulă

1. **Operatorii paranteză**

- Sunt întâlniţi în două situaţii distincte:

  ( ) – generali pentru operaţii şi funcţii;

  [ ] – pentru matrici.

		Regula de evaluare
( )	- Apelul parametrilor de funcție - Stabilește o prioritate într-o expresie	De la stânga la dreapta
[ ]	- Utilizată în vectori și matrici	
.	- Pentru selecția unui element prin referință (la structuri de date)	
->	- Pentru selecția unui element prin adresa sau pointer (la structuri de date)	

2. **Operatori unari**

		Regula de evaluare
++	Increment sufix	De la stânga la dreapta
- -	Decrement sufix	
++	Increment prefix	De la dreapta la stânga
--	Decrement prefix	
+	Plus unar	
-	Minus unar	
!	Negare logică	
~	Negare logică pe biți (complement pe fiecare bit)	
(type)	Conversie tip cast	
*****	Operator de indirectare	
&	Operator de obținere adresă (pointer)	
sizeof	Pentru obținerea dimensiunii în biți	

3. **Operatori de multiplicare**

		Regula de evaluare
*	Multiplicare	De la stânga la dreapta
/	Împărțire	
%	Modulo (rest)	

Exemplu:

```
int a,b;
int E1, E2 ;

….
E1 = (a/b)*b ;

E2 = (a/b)*b + a%b ;
```

4. **Operatori aditivi**

		Regula de evaluare
+	Sumă	De la stânga la dreapta
-	Scadere	

5. **Operatori de deplasare**

		Regula de evaluare
<<	Deplasarea biților către stânga	De la stânga la dreapta
>>	Deplasarea biților către dreapta	

Ex.: x = y << 2;

produce deplasarea la stânga a operandului din stânga cu un număr de 2 poziţii binare (indicat de operandul din dreapta).

Notă:

Formatul expresiei de deplasare: *expresie* >> *expresie*
expresie << *expresie*

<< - produce deplasarea la stânga a operandului din stânga cu un număr de poziţii binare dat de operandul din dreapta.
>> - produce deplasarea la dreapta a operandului din stânga cu un număr de poziţii binare dat de operandul din dreapta.

Observaţii:

- operanzii din stânga trebuie să fie de tip int (întregi).
- operandul din dreapta este convertit la tipul int.
- biţii eliberaţi devin 0.

6. **Operatori relaționali**

		Regula de evaluare
<	Mai mic decât	De la stânga la dreapta
<=	Mai mic sau egal cu	
>	Mai mare decât	
>=	Mai mare sau egal cu	

7. **Operatori de egalitate**

		Regula de evaluare
==	Egal cu	De la stânga la dreapta
!=	Diferit de	

Exemplu:

a<b == c>d ⇔ *1 dacă a<b şi c>d*

0 în rest

8. **Operatori logici pe biți**

		Regula de evaluare
&	**Operatorul ȘI pe biţi**	De la stânga la dreapta
^	**Operatorul pe biţi SAU EXCLUSIV**	
|	**Operatorul pe biţi SAU INCLUSIV**	

Notă:

Operatorii pe biţi pot fi folosiţi pentru a seta biţi.
Exemplu: *x = x & mask*
unde mask este o mască plasată pe biţii de interes.

Dacă, de exemplu ne interesează biţii 2 şi 3 dintr-un octet => mask = 00000110.
Prin aplicarea operatorului & asupra acestei măşti cu orice alt octet vor rămâne neafectaţi din acesta din urmă doar biţii 2 şi 3.

Observaţie: Uneori ne poate interesa ultimul bit pentru a vedea dacă octetul respectiv este par sau impar.

9. **Operatori logici**

		Regula de evaluare
&&	**ȘI logic**	De la stânga la dreapta
||	**SAU logic**	De la stânga la dreapta

10. **Operatorul condițional**

- În C ” **? :** ” este un operator ternar
- Se utilizează în cadrul unei expresii condiţionale care are forma:

  expresie1 ? expresie2 : expresie3

- Dacă expresie1 are valoarea

  de adevăr: 1 => rezultatul este expresie2
- 0 => rezultatul este expresie3
- Ex.:

  variabila = conditie **?** value_if_true **:** value_if_false

  Operatorul **? :** este oarecum similar cu o expresie sau instrucțiune condițională (construcția *if-then-else*)

Observaţie:

- Trebuie acordată atenţie corectitudinii scrierii.

Astfel:

E = ET1 ? (ET2 ? e1 : e2) : e3;

este corect scrisă, în timp ce

E = ET1 ? e1 : ET2 ? e2 : e3;

este incorect scrisă (trebuie puse paranteze).

11. **Operatorii de atribuire**

		Regula de evaluare
= += -= *= /= %= <<= >>= &= ^= |=	Atribuire directă Atribuire cu sumare Atribuire cu diferență Atribuire cu produs …. 	De la dreapta la stânga

Observație:

Operatorii de atribuire intră în expresii care se grupează de la dreapta la stânga, atribuirea fiind singurul caz cu tipul acesta de grupare.

Expresia de atribuire:

lvaloare operator expresie

Dacă operatorul este “=”, valoarea expresiei înlocuieşte pe cea a obiectului referit de lvaloare.

Expresiile de forma: *E1 op* = *E2* sunt echivalente cu *E1* = *E1 op E2* unde “op” este unul dintre operatorii +, –, *, /, %, >>, <<, &, ^ sau |.

Exemplu:

x *= y+1; ⇔ ***x = x*(y+1)***

şi nu ***x = x*y+1***

Observaţie: Pentru **+=** şi **–=** operatorul stâng poate fi şi un pointer (a se vedea capitolul Pointeri).

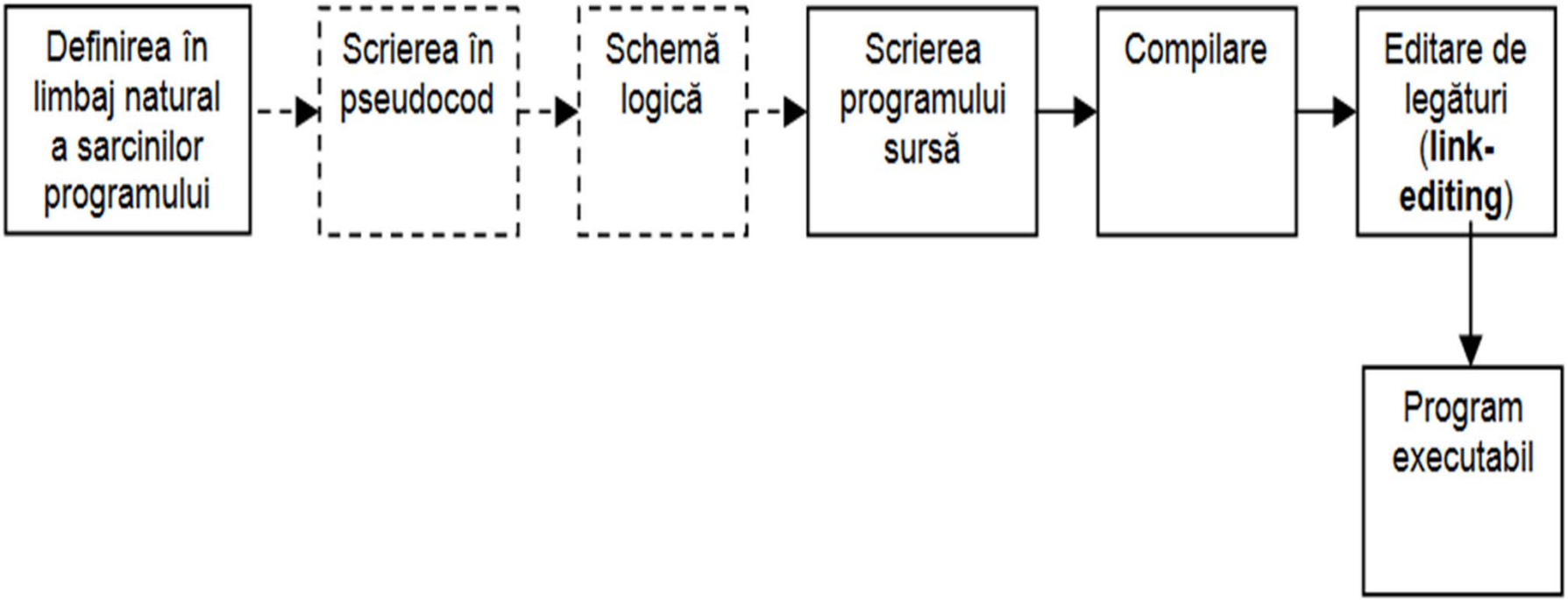
12. **Operatorul virgulă**

Expresia virgulă:

expresie1, expresie2

- Se grupează de la stânga la dreapta.

În funcţie de dimensiunile unui program acesta poate parcurge mai multe etape:

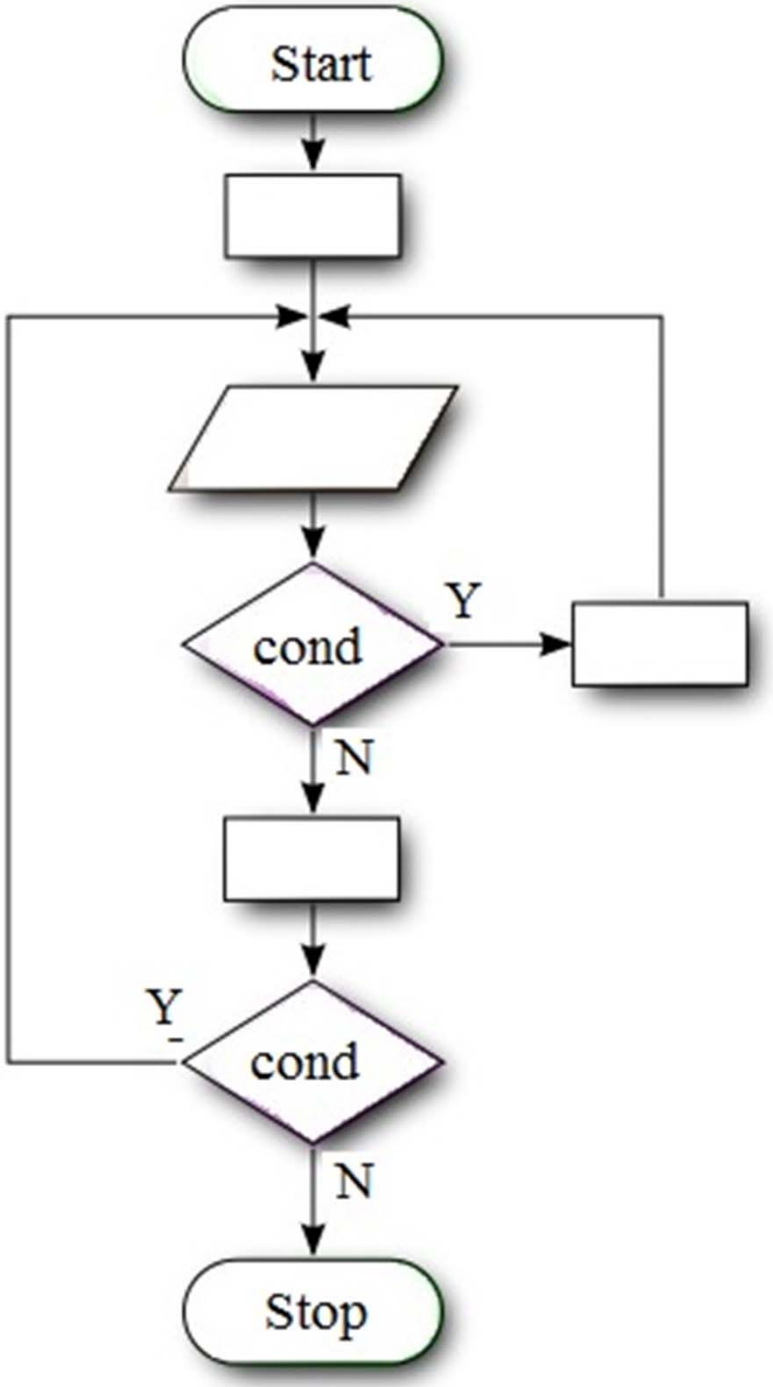


Descrierea unui program în limbaj natural

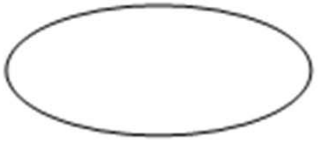
- Odată stabilite obiectivele unui program, descrierea acestuia în limbaj natural este practic independentă de calculatorul folosit.
- În principiu aceasta se referă la enunţarea problemei şi a formalismului matematic utilizat.
- Este neapărat necesar să se studieze toate cazurile problemei încă din acest stadiu pentru că o problemă bine pusă şi rezolvată algoritmic este practic ca şi implementată (restul ţine de experienţa în programare şi sistemul de calcul).
- Descrierea tuturor aspectelor problemei se face cât mai pe larg posibil în limbajul natural obişnuit, folosindu-ne de algoritmi şi de relaţiile matematice cunoscute.

Diagrame logice (Scheme logice)

- Schemele logice (sau bloc) sunt scheme organizate pe blocuri astfel încât să fie respectat principiul de secvenţialitate al unui program.
- *Principiul de secvenţialitate (Von Neumann)*: Trecerea unui program într-o altă etapă se poate face prin terminarea celei precedente sau printr-o condiţie impusă de aceasta.
- Nu se pot desfăşura două etape simultan într-un sistem de calcul secvenţial.



Formate grafice întâlnite în cadrul unei scheme logice:



← alocate proceselor de - început (START)
- sfârşit (STOP)

← leagă blocurile între ele

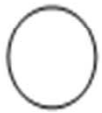
← bloc de operaţie (calcul) poate conţine o instrucţiune, un grup de instrucţiuni, un bloc de instrucţiuni, o buclă, o funcţie, fişier de funcţii etc.

← bloc de decizie

Uneori mai pot fi întâlnite şi alte componente ale unei scheme logice (mai puţin utilizate), cum ar fi:

← bloc de I/E

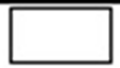
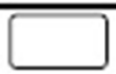
← bloc de procedură



← conector simplu (leagă puncte ale schemei logice din cadrul aceleiaşi pagini)

← conector de pagină (leagă puncte aflate pe pagini diferite)

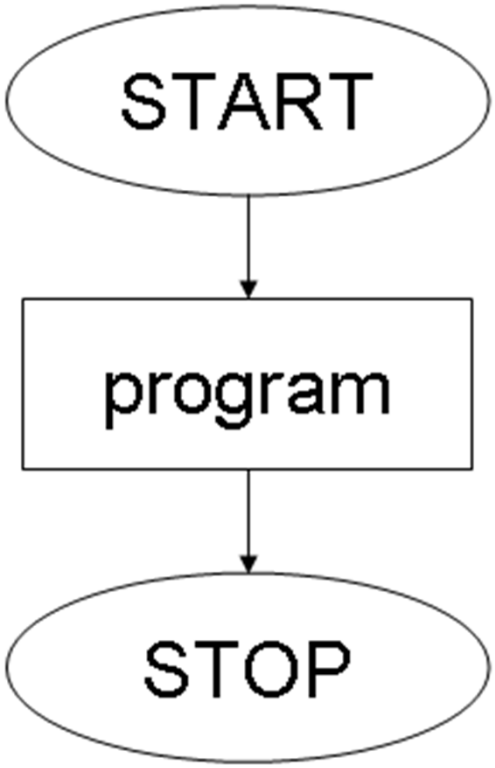
Flowchart Symbol Cheat Sheet

Flowchart Symbol	Name (Alternates)	Description
	Process	An operation or action step.
	Terminator	A start or stop point in a process.
	Decision	A question or branch in the process.
	Delay	A waiting period.
	Predefined Process	A formally defined sub-process.
	Alternate Process	An alternate to the normal process step.
	Data (I/O)	Indicates data inputs and outputs to and from a process.
	Document	A document or report.
	Multi-Document	Same as Document, except, well, multiple documents.
	Preparation	A preparation or set-up process step.
	Display	A machine display.
	Manual Input	Manually input into a system.
	Manual Operation	A process step that isn't automated.
	Card	A old computer punch card.
	Punched Tape	An old computer punched tape input.

Flowchart Symbol Cheat Sheet

Flowchart Symbol	Name (Alternates)	Description
	Connector	A jump from one point to another.
	Off-Page Connector	Continuation onto another page.
	Transfer	Transfer of materials.
	Or	Logical OR
	Summing Junction	Logical AND
	Collate	Organizing data into a standard format or arrangement.
	Sort	Sorting of data into some pre-defined order.
	Merge (Storage)	Merge multiple processes into one. Also used to show raw material storage.
	Extract (Measurement) (Finished Goods)	Extract (split processes) or more commonly - a measurement or finished goods.
	Stored Data	A general data storage flowchart symbol.
	Magnetic Disk (Database)	A database.
	Direct Access Storage	Storage on a hard drive.
	Internal Storage	Data stored in memory.
	Sequential Access Storage (Magnetic Tape)	An old reel of tape.
	Callout	One of many callout symbols used to add comments to a flowchart
	Flow Line	Indicates the direction of flow for materials and/or information

Cea mai simplă schemă bloc (dar practic inutilă) a oricărui program este:



Limbajul pseudocod (algoritmic)

- Se poate întâmpla ca descrierea algoritmilor unui program prin intermediul schemelor logice să se dovedească destul de greoaie din cauza dificultăţilor legate de reprezentarea grafică.
- *Limbajul pseudocod* este la limita sau la graniţa dintre limbajul natural şi instrucţiunile sau/şi funcţiile predefinite ale unui limbaj de programare.
- Practic nu există o standardizare a limbajului algoritmic pseudocod (ceea ce poate fi apreciat uşor şi prin caracterizarea de pseudocod).

Pseudocodul poate avea stiluri de prezentare diferite

- Există practic pentru fiecare limbaj de programare un pseudocod apropiat de natura instrucţiunilor sale.
- Totuşi similitudinea la nivel de instrucţiune pentru diferite limbaje de programare (de exemplu Pascal, C, Java, etc.) face ca pentru un program ajuns deja la stadiul de pseudocod să fie facilă implementarea sa în oricare dintre limbajele de programare înrudite.

Pentru limbajul de programare C putem genera următorul pseudocod:

- Atribuire $v \leftarrow e$

unde v este o variabilă care va avea valoarea dată de rezultatul evaluării expresiei e.

- Citire *read* sau *scan*
- Scriere *write* sau *print*
- Bucle condiţionale *do ... while* sau *while*
- Structură repetitivă *for (iniţializare; condiţii de stop; pas)*
  { instrucţiuni }
- Structură alternativă *if … else*
- Stuctură alternativă generalizată *switch … case*
- Comentarii */* şir de caractere */*
  sau *// şir de caractere* → numai pentru un singur rând.

Exemplu de pseudocod în C

```
void function fizzbuzz
for (i = 1; i<=100; i++)
        { set print_number to true;
        if i is divisible by 3
                print "Fizz“ ;
                set print_number to false;
        if i is divisible by 5
                print "Buzz“ ;
                set print_number to false;
         if print_number, print i;
        print a newline;
        }
```

Scrierea programului în cod sursă

- Scrierea programului în cod sursă se face în editoare de tip text (ASCII) într-un format inteligibil, respectând riguros sintaxa limbajului de programare utilizat.

- Este indicat să se păstreze alineate de blocuri şi instrucţiuni imbricate.

Depanarea programului şi compilarea sa

- Uzual se fac compilări repetate până când sunt eliminate toate erorile şi se obţine programul executabil.
- Uneori, se pot obţine programe fără erori din punct de vedere sintactic dar cu erori logice care nu sunt semnalate de compilator.
- În acest caz trebuie reluată proiectarea programului dintr-o fază anterioară până când se obţine rezultatul scontat

Instrucţiuni

- Ca definiţie, o instrucţiune reprezintă o porţiune a programului care poate fi executată.

- Aceasta specifică o acţiune de un anumit tip.

Standardul ANSI C împarte instrucţiunile în următoarele grupe:

- *Selecţie* (**if** şi **switch** → se mai numesc şi *instrucţiuni condiţionale*)
- *Iterare* (**while**, **for** şi **do – while** → denumite uneori *instrucţiune de buclare*)
- *Salt* (**break**, **continue**, **goto**, **return**)
- *Etichetă* (**case** şi **default** (adiţionate la *switch*) şi **etichetele** (la *goto*))
- *Expresie* (instrucţiuni compuse dintr-o expresie validă)
- *Bloc* (sunt blocuri de cod sau instrucţiuni compuse; un bloc începe cu **{** şi se termină cu **}**)