Facultatea de Matematica si Informatica
Universitatea Bucuresti

Programare procedurala

- suport de curs -

Dobrovat Anca - Madalina

An universitar 2016 – 2017
Semestrul I

Curs 2

Agenda cursului

1. **Algoritmi. Structuri de control si structuri elementare de date (aprofundare de la cursul 1).**

2. **Limbaje de programare.**

3. **Introducere in Limbajul C. Structura unui program in C.**

Algoritmi

Rezolvarea oricărei probleme implică mai multe etape:

1. Analiza problemei
2. Găsirea soluției [optime]
3. Elaborarea algoritmului
4. Implementarea algoritmului într-un limbaj de programare
5. Verificarea corectitudinii algoritmului propus
6. Analiza complexității

Algoritmi

Definiție:

Algoritm = o secvenţă finită de comenzi explicite şi neambigue care executate pentru o mulţime de date (ce satisfac anumite condiţii iniţiale), conduce în timp finit la rezultatul corespunzător.

Caracteristici:

- generalitate, claritate, finititudine, corectitudine, performanță, robustețe

Descriere:

- limbaj natural / pseudocod, diagramă (schemă logică), program etc.

Algoritmi

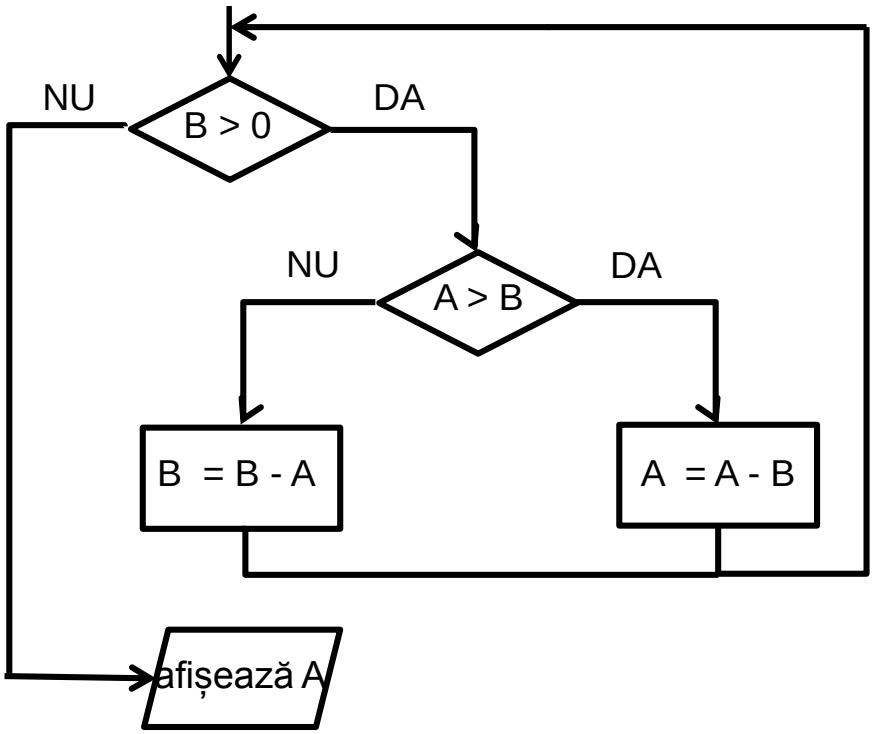
Exemplu:

Algoritmul lui Euclid pentru aflarea celui mai mare divizor comun a două numere A și B.

Pseudocod:

```
cât timp B > 0
dacă A > B
    A = A – B;
altfel
    B = B – A;
afișează A
```

Schemă logică:



Algoritmi

Instructiuni

- implementarea algoritmilor într-un limbaj de programare
- prelucarea datelor (folosim constante sau variabile) într-un limbaj de programare se realizează cu ajutorul instrucțiunilor
- **instrucțiune = proces de prelucrare al datelor pe care un calculator îl poate realiza**
  - ; //instrucțiunea vidă
  - printf(“A \n”);//afișează A și treci pe linia următoare
  - B = B – A;
  - B = -A;
  - B = 2 * A;
  - B =* 2;

Algoritmi

Structuri de control

- entitățile de bază ale unui limbaj de programare structurat
- controlează modul de executare al unui program
- trei structuri de control fundamentale:

1. structura secvențială;
2. structura condițională (de decizie, de selecție);
3. structura repetitivă (ciclică).

Algoritmi

Structura secventiala

- execută secvențial instrucțiuni

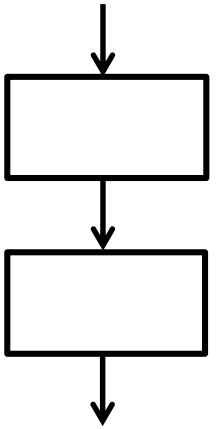
Pseudocod:

Instrucțiune1;

Instrucțiune2;

…

Schemă logică:



Exemplu:

```
aux = B;
A = B;
B = aux;
```

Algoritmi

Structura decizionala

- ramifică execuția programului în funcție de o condiție;
- instrucțiuni: if, if else, switch.

Instrucțiunea IF

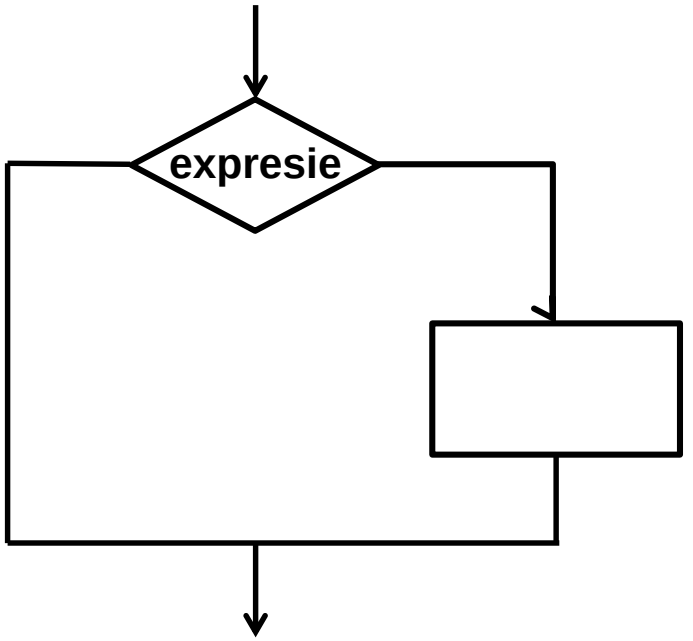
Sintaxa:

```
if (expresie)
  instructiune1;
```

Pseudocod:

```
dacă (expresie) e adevărată
    execută instructiune1;
```

Schemă logică



Algoritmi

Structura decizionala

Instrucțiunea IF ELSE

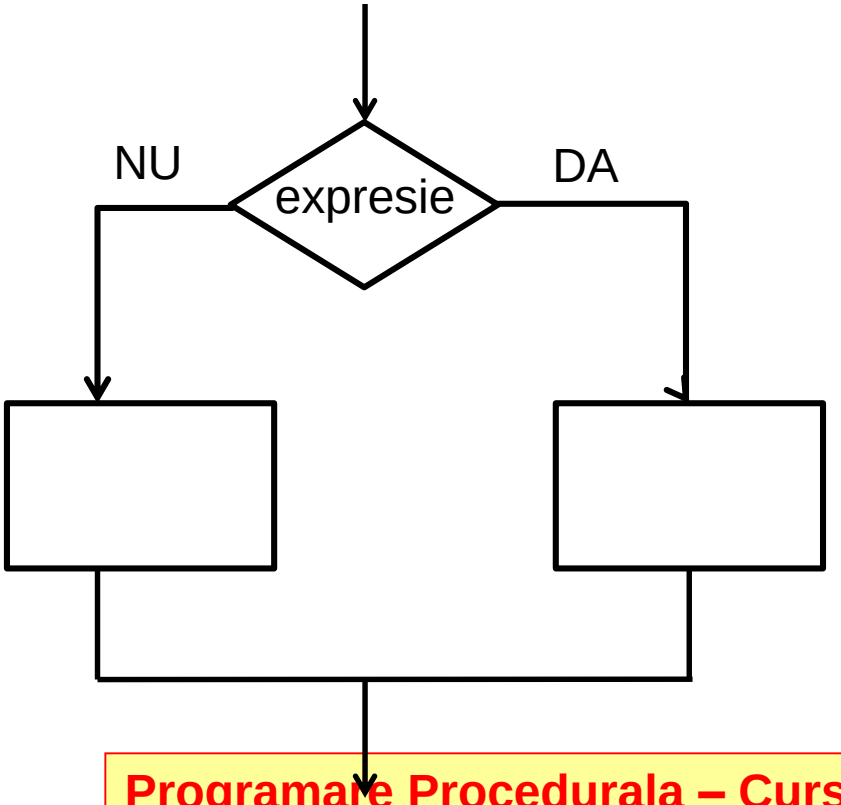
Sintaxa:

```
if (expresie)
  instructiune1;
else
     instructiune2;
```

Pseudocod:

```
dacă (expresie) e adevărată
    execută instructiune1;
altfel
     execută instructiune2;
```

Schemă logică



Algoritmi

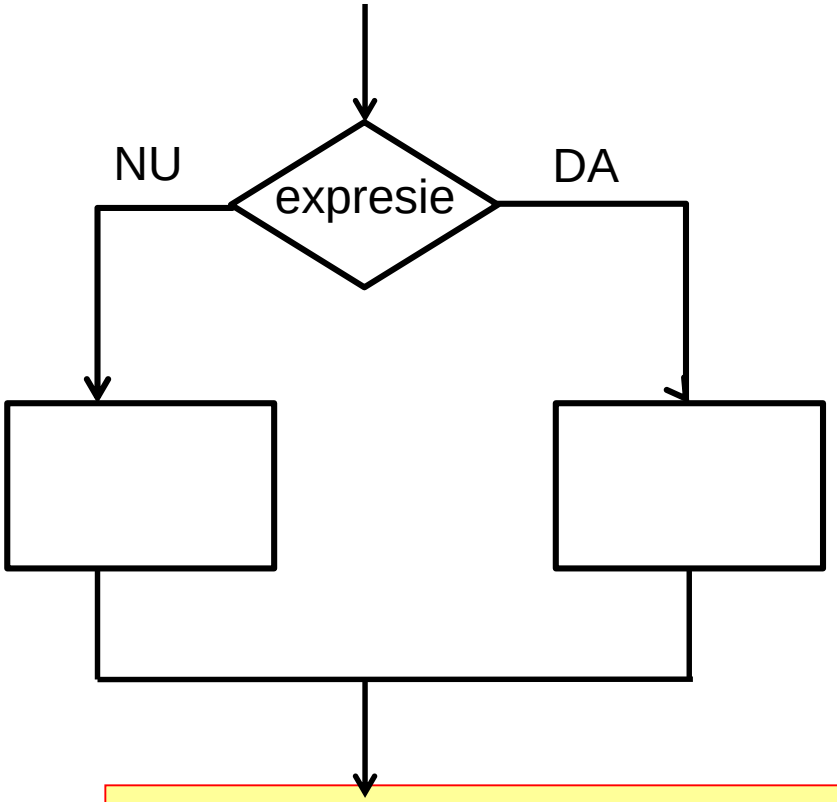
Structura decizionala

Alternativă la IF ELSE: operatorul ternar ?

Sintaxa:

(expresie) ? (instructiune1) : instructiune2

Schemă logică



Pseudocod:

```
dacă (expresie) e adevărată
    execută instructiune1;
altfel
    execută instructiune2;
```

Algoritmi

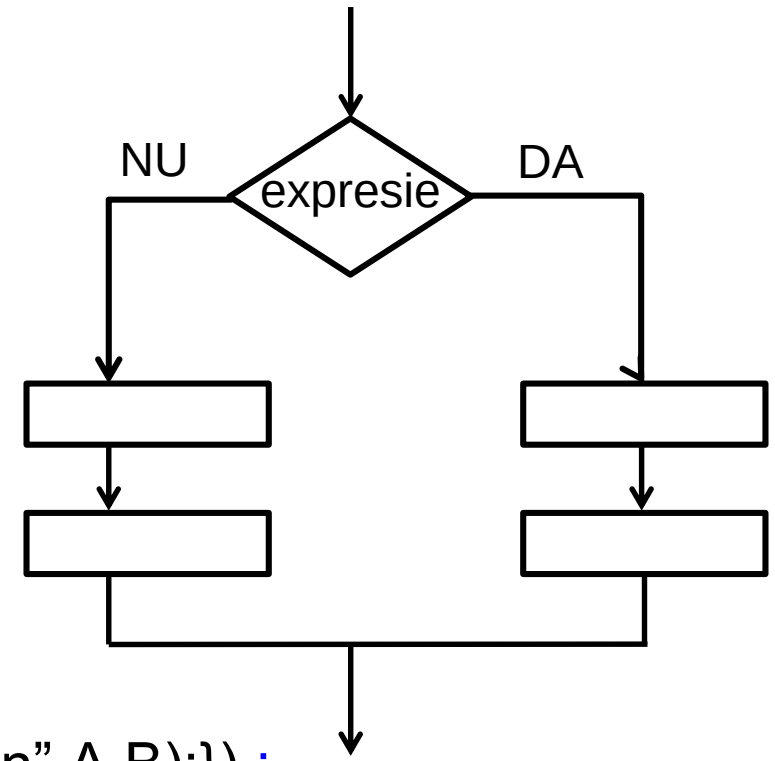
Structura decizionala

Alternativă la IF ELSE: operatorul ternar ?

Sintaxa:

(expresie) ? ({set_de_instructiuni1}) :
({set_de_instructiuni2});

Schemă logică



Pseudocod:

dacă (expresie) e adevărată
    execută set_de_instructiuni1;
altfel
    execută set_de_instructiuni2;

Exemplu:

(A > B) ? ({A = A – B;printf("A=%d \n B = %d \n",A,B);}) :
({B = B – A; printf("A=%d \n B = %d \n",A,B);});

Algoritmi

Structura decizionala

Alternativă la IF ELSE: operatorul ternar ?

Sintaxa IF ELSE:

```
if (expresie)
  instructiune1;
else
  instructiune2;
```

→

Operator ternar:

```
(expresie) ? (instructiune1) : (instructiune2);
```

dacă

altfel

Exemplul 1: (A > B) ? (A = A-B) : (B = B – A);

Exemplul 2: (A > B) ? ({A = A – B;printf(“A > B”);}) :
({B = B – A; printf(“B > A \n”);});

Algoritmi

Structura decizionala

Instrucțiunea SWITCH

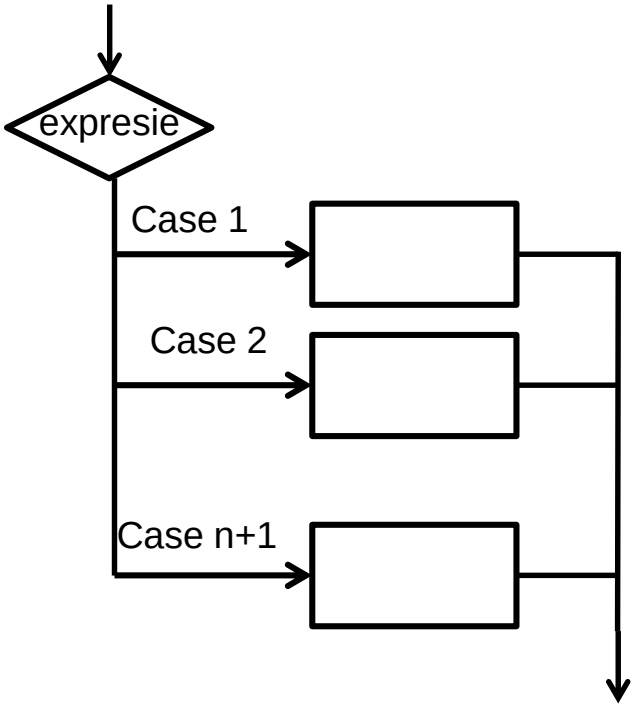
Sintaxa:

```
switch (expresie){
   case (expresie_1):
instructiune_1;
   case (expresie_2):
      instructiune_2;
.....
   case (expresie_n):
instrucțiune_n;
default:
instrucțiune_(n+1);
}
```

Pseudocod:

```
dacă expresie=expresie_1
      execută instructiune_1;
altfel dacă expresie=expresie_2
      execută instructiune_2;
.....
altfel dacă expresie=expresie_n
      execută instructiune_n;
altfel
instructiune_(n+1);
```

Schemă logică



Algoritmi

Structura decizionala

Instrucțiunea SWITCH

Exemplu:

```
scanf("%c", &oper);
switch (oper)
        {
            case ('+'): printf("Operatorul de adunare!\n");
                        break;
            case ('-'): printf("Operatorul de scadere!\n");
                        break;
        case ('*'): printf(" Operatorul de inmultire!\n");
                    break;
            default: printf("Operator ilegal!\n");
        }
    }
```

Algoritmi

Structura repetitiva

- repetă execuția unei [secvențe de] instrucțiuni în funcție de o condiție;

Clasificare:
- cu numar cunoscut de pasi (instructiunea FOR)

- cu numar necunoscut de pasi:

- cu test initial (instructiunea WHILE)

- cu test final (instructiunea DO WHILE)

Algoritmi

Structura repetitiva

Instrucțiunea FOR

Sintaxa:

```
for (expresie1; expresie2; expresie3)
    instructiune;
```

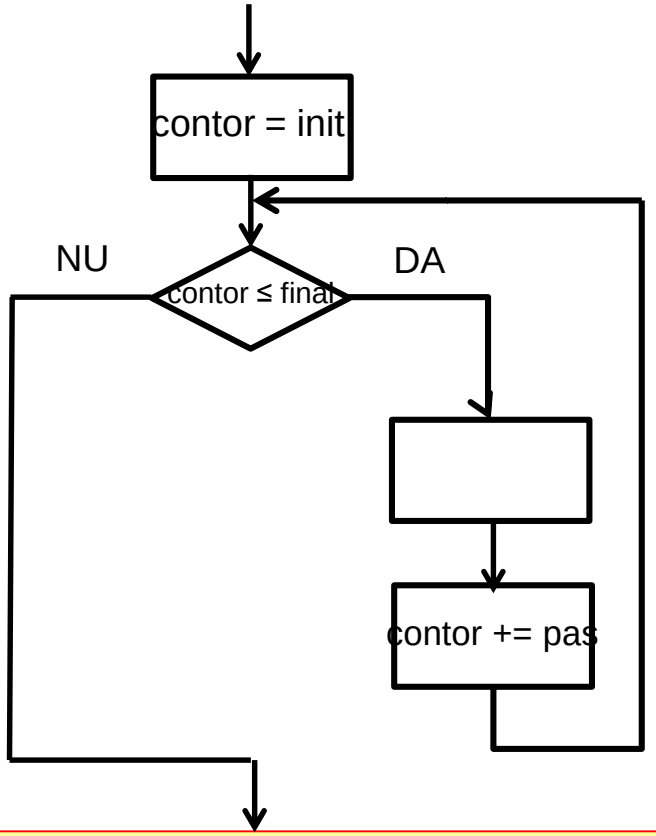
Pseudocod:

pentru contor ← init, final, pas
executa [set de instructiuni]

Cazuri:

1. init ≤ final => pas / cadenta pozitiv(a)
2. Init ≥ final => pas / cadenta negativ(a)

Schemă logică



Algoritmi

Structura repetitiva

Instrucțiunea FOR

Obs: Nu este obligatorie prezenta expresiilor, ci doar a instructiunilor vide.

```
for ( ; expresie2; )
    instructiune;
```

sau:

```
for ( ; ; )
    instructiune;
```

Exemple:

```
int  S=0, P=1, k;
for (k=1; k<=n; k++){
 S+=k; P*=k;}
```

Suma si produsul primelor n nr naturale

```
for( ; c!='@'; ){
        instructiuni
 }
```

Citirea unui caracter pana la intalnirea @

Algoritmi

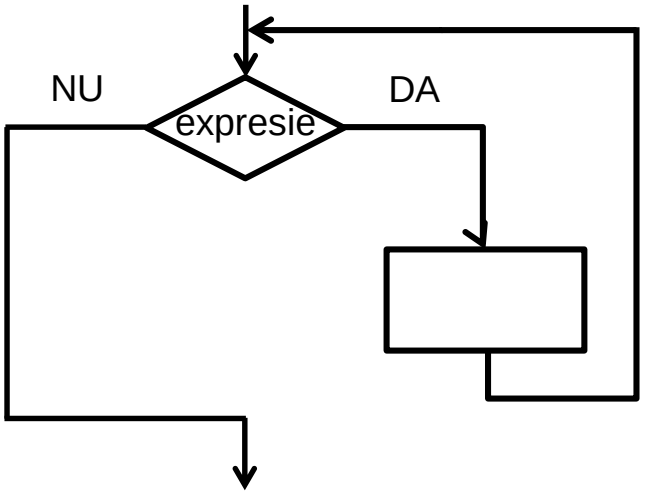
Structura repetitiva

Instrucțiunea WHILE

Sintaxa:

```
while (expresie)
      instructiune;
```

Schemă logică



Pseudocod:

cât timp (expresie) este adevărată
execută [set de instructiuni]

Exemplu:

```
int  S=0, P=1, k=1;
while (k<=n){
      S+=k; P*=k;
k++;
}
```

Algoritmi

Structura repetitiva

Sintaxa:

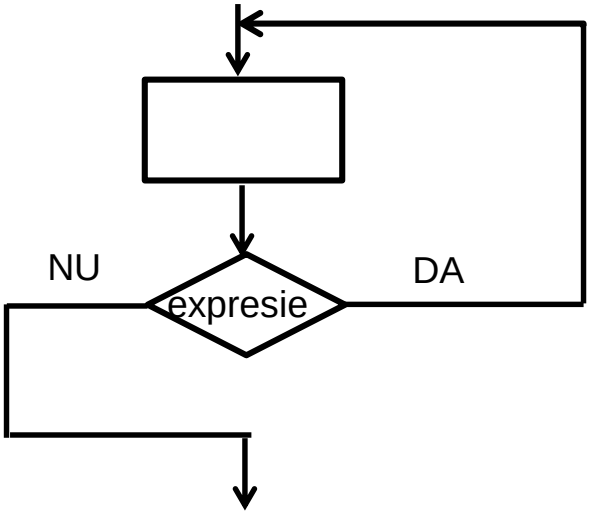
Instrucțiunea DO WHILE

```
do
instrucțiune;
while (expresie) ;
```

Pseudocod:

```
execută [set de instructiuni]
cât timp (expresie) este adevărată
```

Schemă logică



Exemplu:

```
int  S=0, P=1, k=1;
do {
      S+=k; P*=k;
k++;
} while (k<=n);
```

Algoritmi

Structuri repetitive

EXEMPLE

Suma si produsul primelor n nr naturale

```
int  S=0, P=1, k;
```

```
for (k=1; k<=n; k++)
{
 S+=k; P*=k;
}
```

```
k=1;
while (k<=n)
{
      S+=k; P*=k;
k++;
}
```

```
k=1;
do
{
      S+=k; P*=k;
k++;
} while (k<=n);
```

Ce valori vor avea variabilele S ai P pentru n=0 ?

Algoritmi

Structuri repetitive

Facilitati de intrerupere a unei secvente

Instructiunea Break

- asigura iesirea dintr-o bucla la nivelul imediat superior

- in cadrul instructiunii switch – pentru directionarea fluxului in afara instructiunii

Instructiunea Continue

- se utilizeaza pentru intreruperea executiei iteratiei curente

Se vor da mai multe detalii si exemple in cadrul cursului si al laboratorului.

Algoritmi

Structuri repetitive

Instructiunea CONTINUE

Sintaxa:

continue;

Pseudocod:

continuă execuția programului cu iterația următoare din bucla curentă;

citește 10 numere și află câte din ele sunt pare

Exemplu:

```
int  nrPare=0, N=10, k, nr;
for(k = 0; k < N; k++){
      scanf("%d",&nr);
if ((nr % 2) != 0)
      continue;
nrPare +=1;
}
printf("nrPare = %d\n",nrPare);
```

Algoritmi

Structuri repetitive

Sintaxa:

break;

Instructiunea BREAK

Pseudocod:

ieși din bucla curentă și continuă execuția programului cu instrucțiunea următoare;

Exemplu:

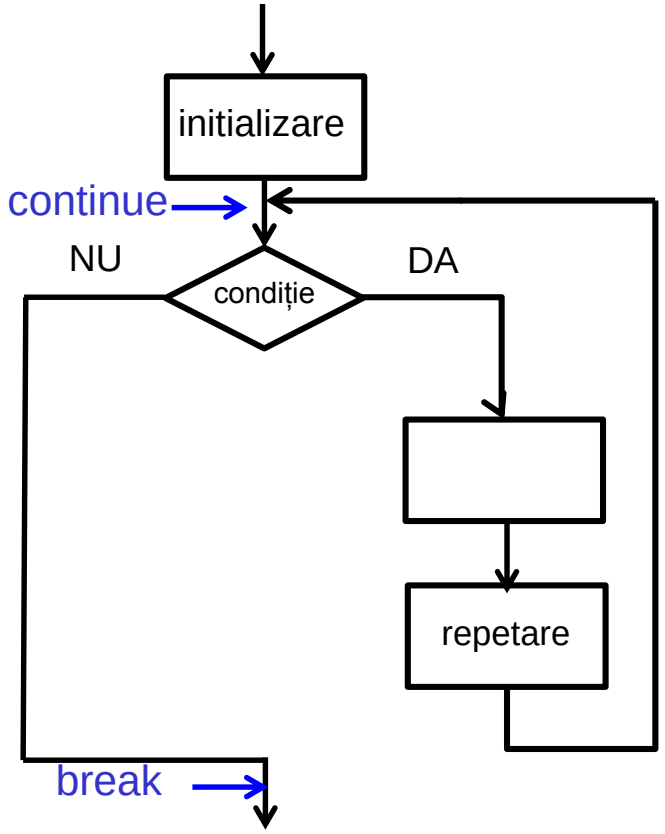
```
int  nrPare=0, N=10, k, nr;
for(k = 0; k < N; k++){
      scanf("%d",&nr);
if ((nr % 2) != 0)
      break;
nrPare +=1;
}
printf("nrPare = %d\n",nrPare);
```

la primul număr impar citit ieși din buclă (se citesc maxim 10 numere pare)

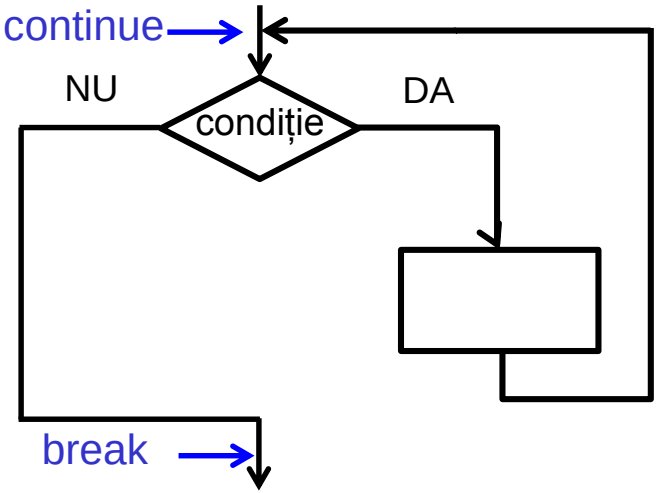
Algoritmi

Structuri repetitive

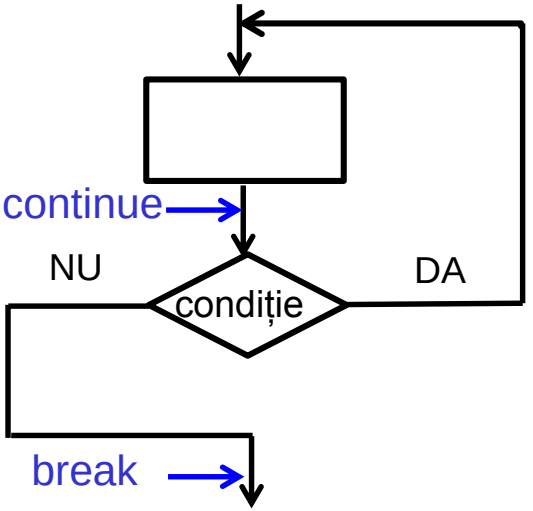
Locurile în schema logică unde apar CONTINUE și BREAK



for



while



do … while

Limbaje de programare

Rezolvarea oricărei probleme implică mai multe etape:

1. Analiza problemei
2. Găsirea soluției [optime]
3. Elaborarea algoritmului
4. Implementarea algoritmului într-un limbaj de programare
5. Verificarea corectitudinii algoritmului propus
6. Analiza complexității

Limbaje de programare

Limbaj de programare - notație sistematică prin care este descris un proces de calcul.

Proces de calcul - succesiunea de operații elementare (pe care un calculator le poate executa) asociate algoritmului de rezolvare a unei probleme.

Limbajele de programare sunt limbaje artificiale.

Limbaje de programare

Evolutie si clasificare

Limbaje native / limbaje masina

Limbaje de asamblare

Limbaje de nivel inalt

```
00000000101000010000000000011000
00000000000011000000110000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Instrucțiuni în limbaj mașină

```
swap:
        muli  $2, $5,4
        add   $2, $4,$2
        lw    $15, 0($2)
        lw    $16, 4($2)
        sw    $16, 0($2)
        sw    $15, 4($2)
        jr    $31
```

Instrucțiuni în limbaj de asamblare

```
swap(int v[], int k)
{int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

Instrucțiuni în limbajul C

Limbaje de programare

Limbaje low-level

Limbaj mașină

- limbajul nativ al unui calculator (mașină);
- șabloane de numere binare (reprezintă modul binar de codificare a instrucțiunilor și datelor în memorie)
- depinde de arhitectura sistemului

```
00000000101000010000000000011000
00000000000011000000110000010000 1
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Instrucțiuni în limbaj mașină

Limbaj de asamblare

- în loc de cod mașină folosește o desemnare simbolică a elementelor programului (instrucțiuni, date)
- 01011011 = ADD, 01011100 = SUB

```
swap:
        muli  $2, $5,4
        add   $2, $4,$2
        lw    $15, 0($2)
        lw    $16, 4($2)
        sw    $16, 0($2)
        sw    $15, 4($2)
        jr    $31
```

Instrucțiuni în limbaj de asamblare

Limbaje de programare

Limbaje de nivel inalt

- cuprind mecanisme de exprimare apropiate de limbajul natural.

- folosesc verbe pentru a desemna acțiuni (**do, repeat, read, write, continue, switch, call, goto**, etc.), conjunctii (**if, while**), adverbe (**then, else**), mecanisme de declare si definire.

- oferă suport pentru importul/exportul de module (pachete, sub-proiecte).

Limbaje de programare

Limbaje de nivel inalt

- au o descriere sintactică si semantică bine definită
- descurajează greșelile de programare
- independente de procesor (pentru asigurarea portabilității codului)
- independente de sistemul de operare (pentru a permite realizarea de software multi-platforma)

Limbaje de programare

Paradigme de programare

Sursa: Albeanu G – Programare procedurala (note de curs 2013)

A: PARADIGMA PROGRAMARII PROCEDURALE SI STRUCTURATE :
Un program este privit ca o multime ierarhica de blocuri si proceduri;
B: PARADIGMA PROGRAMARII ORIENTATE SPRE OBIECT: Un program este constituit dintr-o colectie de obiecte care interactioneaza;
C: PARADIGMA PROGRAMARII CONCURENTE SI DISTRIBUITE:
Executia unui program este constituita din actiuni multiple posibil a fi executate in paralel pe una sau mai multe masini;
D: PARADIGMA PROGRAMARII FUNCTIONALE: Un program este descris pe baza unor functii de tip matematic (fara efecte secundare), utilizate de obicei recursiv;
E: PARADIGMA PROGRAMARII LOGICE: Un program este descris printr-un set de relatii intre obiecte precum si de restrictii ce definesc cadrul in care functioneaza acele obiecte. Executia inseamna activarea unui proces deductiv.
F: PARADIGMA PROGRAMARII LA NIVELUL BAZELOR DE DATE:
Actiunile programului sunt dictate de cerintele unei gestiuni corecte si consistente a bazelor de date asupra carora actioneaza programul.

Limbaje de programare

Paradigma programarii procedurale

execuție secvențială

variabile reprezentate ca poziții în memorie și modificate prin atribuiri

unitatea de program de bază: procedura = funcția = rutină = subrutină = subprogram

C, Pascal, Fortran, Basic, Algol

Limbaje de programare

Programarea Procedurala – Compilarea unitatilor de program

Analiza textului sursa

- lexicala – produce sir de atomi lexicali
- sintactica – produce arbori sintactici
- semantica – produce codul intermediar

Sinteza codului obiect

- optimizarea codului – produce cod intermediar optimizat
- generarea de cod – produce codul obiect final

Limbaje de programare

Programarea Procedurala – Editarea de legaturi

Mai multe module obiect (bucati de cod generate separat) se asambleaza impreuna cu module din bibliotecile standard pentru crearea aplicatiei finale.

Aplificatia finala este obtinuta prin segmentare (overlay) sau cu ajutorul bibliotecilor dinamice (dll)

Build = Compilare + Editare de legaturi

Limbaje de programare

Programarea Procedurala – Executarea programelor

Se realizeaza in urma pregatirii pentru executare de catre sistemul de operare.

Poate fi intrerupta (cazul sistemelor multitasking primitiv) de catre utilizator sau de catre sistemul de operare (sisteme multitasking pentru mai multi utilizatori) si poate fi reluata pe baza unei strategii de planificarea lucrarilor (prioritati, round robin, etc.)

Deoarece functia main (in C/C++) returneaza un rezultat aceasta face ca mai multe programe sa poata fi apelate dintr-un program “panou de comanda” pentru operare.

Introducere in Limbajul C

popular, rapid și independent de platformă

este un limbaj utilizat cel mai adesea pentru scrierea programelor eficiente și portabile: sisteme de operare, aplicații embedded, compilatoare, interpretoare, etc.

limbajul C a fost dezvoltat la începutul anilor 1970 în cadrul BellLaboratories de către Dennis Ritchie

- strâns legat de sistemele de operare UNIX

stă la baza pentru majoritatea limbajelor ”moderne”: C++, Java, C#, Javascript, Objective-C, etc.

Introducere in Limbajul C

trei standarde oficiale active ale limbajului

- **C89** (C90) – aprobat în 1989 de ANSI (American National Standards Institute) și în 1990 de către ISO (International Organization for Standardization)
  - C89 a eliminat multe din incertitudinile legate de sintaxa și gramatica limbajului.
  - cele mai multe compilatoare de C sunt compatibile cu acest standard (ANSI C)
- **C99** – standard aprobat în 1999, care include corecturile aduse C89 dar și o serie de caracteristici proprii care în unele compilatoare apăreau ca extensii ale C89 până atunci
  - compilatoarele oferă suport limitat și în multe cazuri incomplet pentru acest standard
- **C11** – standard aprobat în 2011 și care rezolvă erorile apărute în standardul C99 și introduce noi elemente, însă suportul pentru C11 este și mai limitat decât suportul pentru C99, majoritatea compilatoarelor nu s-au adaptat încă la acest standard

Introducere in Limbajul C

Programele C sunt propozitii formate cu simboluri ale alfabetului C: atomi lexicali (tokens) si separatori.

Atomii lexicali - identificatori, constante, operatori, semne de punctuatie.

Cuvinte cheie (32)

C89 = ANSI C : 32 de cuvinte cheie

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

C99: ANSI C + alte 5 cuvinte cheie

_Bool _Complex _Imaginary inline restrict

Limbaje de programare

Structura generala a unui program in C

- modul principal (functia main, clasa aplicatie cu metoda main)

- zero, unul sau mai multe module (functii/proceduri, metode ale clasei aplicatie) care comunica intre ele si/sau cu modulul principal prin intermediul parametrilor si/sau a unor variabile globale

Unitatea de program cea mai mica si care contine cod este **functia / procedura** si contine:

- partea de declaratii/definitii
- partea imperativa (comenzile care se vor executa)

Avantaje: compilare separata, reutilizarea codului, lucrul in echipa, lucrul la distanta, testarea/verificarea codului – Unit testing

Introducere in Limbajul C

Structura unui program in C simplu

directive de preprocesare

int main(void)
{
declarații de variabile locale
instrucțiuni
}

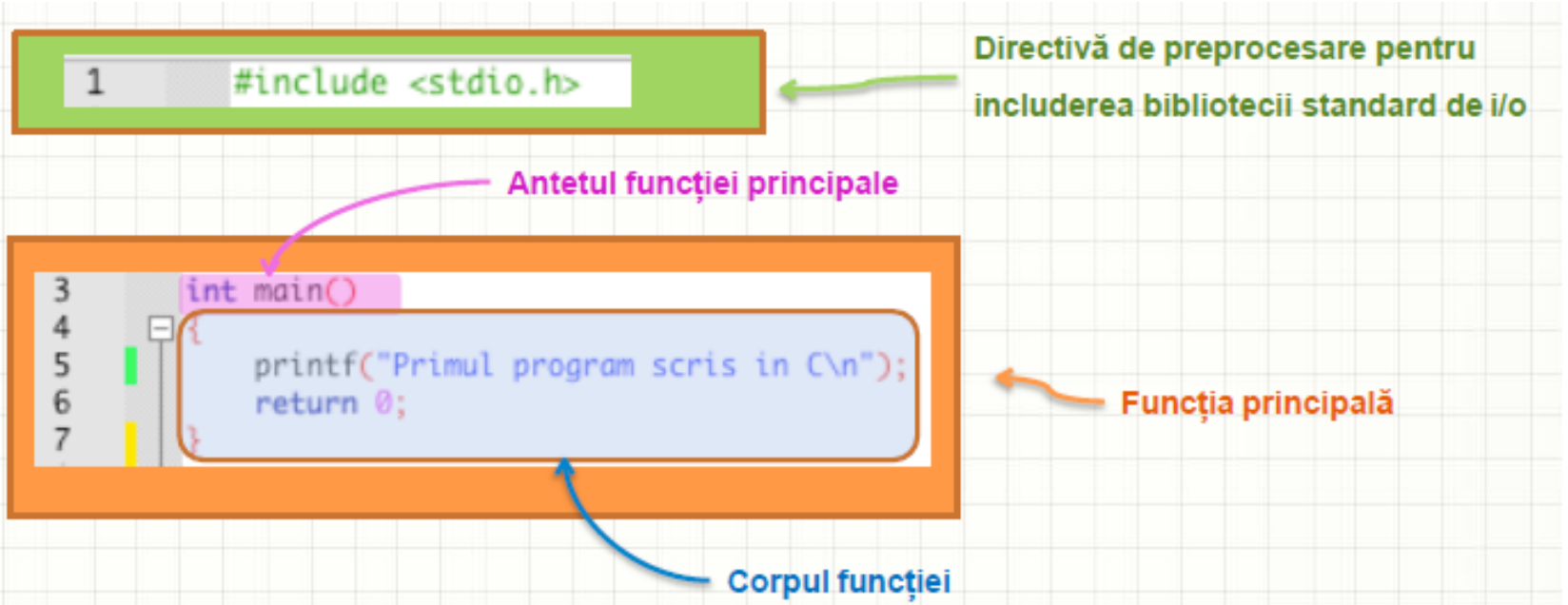
main.c

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

int main()
{
    printf("Primul program scris in C\n");
    return 0;
}
```

Introducere in Limbajul C

Structura unui program in C simplu



Observații:

- main nu este cuvânt cheie în limbajul C, îl utilizăm pentru numirea funcției principale;
- printf nu este cuvânt cheie, este funcție de bibliotecă (print (afișare) +f (format));
- C este case sensitive, se face diferență între litere mici și mari;
- toate cuvintele cheie se scriu cu litere mici;
- instrucțiunile se termină cu caracterul ; (punct și virgulă);
- mai multe instrucțiuni pot fi scrise pe aceeași linie;
- spațiile ajută la organizarea codului.

Sursa: Alexe B –Programare procedurala (note de curs 2016)

Introducere in Limbajul C

Structura unui program in C simplu

Directive de preprocesare

- directive de definiție: #define N 10
- directive de includere a bibliotecilor: #include <stdio.h>
- directive de compilare condiționată: #if, #ifdef, ...

Funcții

- grupări de instrucțiuni sub un nume;
- returnează o valoare sau se rezumă la efectul produs;
- funcții scrise de programator vs. funcții furnizate de biblioteci;
- programul poate conține mai multe funcții;
  - main este obligatoriu;
- antetul și corpul funcției.

Sursa: Alexe B –Programare procedurala (note de curs 2016)

Introducere in Limbajul C

Structura unui program in C simplu

Instrucțiuni

- formează corpul funcțiilor
  - exprimate sub formă de comenzi
- 5 tipuri de instructiuni:
  - instrucțiunea declarație;
  - instrucțiunea atribuire;
  - instrucțiunea apel de funcție;
  - instrucțiuni de control;
  - instrucțiunea vidă;
- toate instrucțiunile (cu excepția celor compuse) se termină cu caracterul ";"
  - caracterul ; nu are doar rol de separator de instrucțiuni ci instrucțiunile incorporează caracterul ; ca ultim caracter
  - omiterea caracterului ; reprezintă eroare de sintaxă

Sursa: Alexe B –Programare procedurala (note de curs 2016)

Introducere in Limbajul C

Structura unui program generalizat

Comentarii

directive de preprocesare

declarații de variabile globale

subprograme

```
int main(void)
{
declarații de variabile locale  instrucțiuni
}
```

Introducere in Limbajul C

Structura unui program generalizat

Citirea unui numar natural si afisarea inversului

main.c

```
1  //Citirea unui numar natural si afisarea inversului
2
3  #include <stdio.h>
4
5  void citire_natural(int *a)
6  {
12
13 int invers(int n)
14 {
23
24 int inv;
25
26 int main()
27 {
28     int n;
29     printf("Citirea unui numar natural\n");
30     citire_natural(&n);
31     printf("Numarul citit este %d\n", n);
32     inv = invers(n);
33     printf("Inversul numarului citit este %d\n", inv);
34     return 0;
35 }
```

```
C:\Users\Ank\Desktop\teste\bin\Debug\teste.exe
Citirea unui numar natural
1234
Numarul citit este 1234
Inversul numarului citit este 4321
```

Introducere in Limbajul C

Structura unui program generalizat

Citirea unui numar natural si afisarea inversului

```
5   void citire_natural(int *a)
6   {
7       do
8       {
9           scanf("%d",&*a);
10      }while(*a<0);
11  }
12
13  int invers(int n)
14  {
15      int og = 0;
16      while(n!=0)
17      {
18          og = og * 10 + n%10;
19          n = n / 10;
20      }
21      return og;
22  }
```

```
C:\Users\Ank\Desktop\teste\bin\Debug\teste.exe
Citirea unui numar natural
1234
Numarul citit este 1234
Inversul numarului citit este 4321
```

Introducere in Limbajul C

Comentarii

- Formă de documentare a acțiunilor programatorului
- Complet ignorate de compilator
- Foarte utile pentru organizarea și înțelegerea cu ușurință a programului
- Tipuri de comentariu
  - C89 oferă un singur fel de comentariu
    - Începe cu /* și se termină cu */
      - se pot extinde pe mai multe linii
        - nu imbricate
        - utile în inserarea unor explicații mai lungi
  - C99 oferă și comentariul pe o singură linie
    - Începe cu // și se termină la finalul liniei
      - utile ca și comentarii inserate pe marginea codului

Introducere in Limbajul C

Expresii

Variabile si Constante

- Stochează datele necesare programului
  - Valorile stocate în memoria sistemului în mod transparent programatorului
    - referirea la aceste date se face prin numele lor simbolice, adică prin identificatori

- **Variabilele** stochează date care pot fi modificate în timpul execuției

- **Constantele** păstrează aceeași valoare (cea cu care au fost inițializate) până la terminarea programului

Introducere in Limbajul C

Expresii

Variabile si Constante

tip nume_variabila = constanta;

Expl.

```
char  ch = 'A';
int x = 10;
float media = 8.57;
```

Introducere in Limbajul C

Expresii

Nume de identificatori (variabile, constante, functii, etichete etc.)

-unul sau mai multe caractere

- primul este "_" sau o litera

- urmatoarele: cifre, litere sau "_"

Corect	Incorect
numarator	1numarator
test23	salut!
bilant_mare	bilant...mare

Case sensitive – nume, NUME, Nume – identificatori diferiti

Un identificator nu poate fi un cuvant cheie sau numele unei functii din biblioteca C

Introducere in Limbajul C

Expresii

Variabile

- numele variabilei nu are legatura cu tipul ei

Variabile

- **locale** (definite in interiorul unei functii)
- **parametri formali** (declarare dupa numele functiei)
- **globale** (cunoscute de intreg programul)

Introducere in Limbajul C

Expresii - Variabile locale

Există o diferență importantă între modul de declarare a variabilelor locale în C față de C++. În C, trebuie să declarați toate variabilele locale la începutul blocului în care le definiți, înainte de orice instrucțiuni ale programului. De exemplu, următoarea funcție este greșită dacă este compilată cu un compilator de C.

```
/* Aceasta functie este gresita daca este compilata cu un
   compilator de C, dar perfect acceptabila pentru un
   compilator de C++.
*/
void f(void)
{
    int i;
    i = 10;
    int j;    /*aceasta linie va determina o eroare*/
    j = 20;
}
```

Introducere in Limbajul C

Expresii - Variabile locale

Puteți inițializa o variabilă locală cu o valoare cunoscută. Această valoare va fi atribuită variabilei de fiecare dată când se va intra în blocul de cod în care este ea declarată. De exemplu, următorul program afișează numărul **10** de zece ori.

```
#include <stdio.h>
void f(void);
void main(void)
{
    int i;
    for(i=0; i<10; i++)   f();
}
void f(void)
{
    int j = 10;
    printf("%d ", j);
    j++;  /*aceasta linie nu are nici un efect */
}
```

Introducere in Limbajul C

Expresii - Parametri formali

Daca o functie urmeaza sa foloseasca argumente, ea trebuie sa declare Variabilele pe care le accepta ca valori ale argumentelor (i.e. parametri formali).

Se comporta ca o variabila locala a functiei.

Exemplu:

```
Este_in (char *s, char c)
{
        while(*s)
                if (*s==c) return 1;
                else s++;
        return 0;
}
```

Obs: Aritmetica pointerilor (se va discuta ulterior)!

Programul returneaza 1 daca c face parte din sirul s si 0 in caz contrar.

Introducere in Limbajul C

Expresii - Variabile locale si parametri

main.c

```c
#include <stdio.h>

void f1()
{
    int a;
    a = 10;
    printf("Valoare variabilei locale din f1= %d \n",a);
}

void f2()
{
    int a;
    a = 33;
    printf("Valoare variabilei locale din f2= %d \n",a);
}

void f3(int x)
{
    x = 56;
    printf("Valoare parametrului din f3= %d \n",x);
}

int main()
{
    int b;
    f1();
    f2();
    f3(b);
    return 0;
```

```
C:\Users\Ank\Desktop\teste\bin\Debug\teste.exe
Valoare variabilei locale din f1= 10
Valoare variabilei locale din f2= 33
Valoare parametrului din f3= 56

Process returned 0 (0x0)   execution time : 0.011 s
Press any key to continue.
```

Introducere in Limbajul C

Expresii - Variabile globale

Se declara in afara oricarei functii si sunt cunoscute in intreg programul

Pot fi utilizate de catre orice zona a codului

Isi pastreaza valoarea pe parcursul intregii executii a programului.

Orice expresie are acces la ele, indiferent de tipul blocului de cod in care se afla expresia.

Introducere in Limbajul C

Expresii - Variabile globale

```
#include <stdio.h>

int a = 10;

void f1()
{
    a = 30;
    printf("Valoare variabilei globale in f1: %d \n",a);
}

int main()
{
    printf("Valoarea initiala a variabilei globale: %d\n", a);
    a = 20;
    printf("Valoarea variabilei globale in main: %d\n", a);
    f1();
    return 0;
}
```

```
C:\Users\Ank\Desktop\teste\bin\Debug\teste.exe
Valoarea initiala a variabilei globale: 10
Valoarea variabilei globale in main: 20
Valoare variabilei globale in f1: 30
```

Introducere in Limbajul C

Tipuri de date

- Tipul de dată specifică
  - natura datelor care pot fi stocate în variabilele de acel tip
  - necesarul de memorie și
  - operațiile permise asupra acestor variabile

5 tipuri de date de baza:
-char, int, float, double, void

- C99 a introdus tipul _Bool (true, false)
  - De fapt valori întregi (0 fals, orice altceva adevărat)

Introducere in Limbajul C

Tipuri de date

- tipul **întreg** – **int**: poate reține valori întregi, ex. 1, 0, -532, etc.;

- tipul **caracter** – **char**: poate reține un singur caracter sub forma codului elementelor din setul de caractere specific
  - codul **ASCII** (American Standard Code for Information Interchange) reprezentat pe 7 biți (poate reprezenta 128 de caractere) – set care este frecvent extins la codul **Latin- 1**, pe 8 biți care poate reprezenta 256 de caractere

- tipul **real** (numere în virgulă mobilă) – simplă precizie – **float**: pot reține valori mai mari decât int și care conțin parte fracționară
  - de ex. 4971.185, -0.72561, etc.

- tipul **real** (numere în virgulă mobilă) **în dublă precizie** – **double**: pot reține valori reale în virgulă mobilă cu o precizie mai mare decât tipul float

- tipul **void**: indică lipsa unui tip anume

Introducere in Limbajul C

Tipuri de date

- Programtorul poate crea noi tipuri prin combinarea tipurilor de bază
- Reprezentarea și spațiul ocupat de diferitele tipuri de date depind de
  - Platformă, sistem de operare și compilator
- Limitele specifice unui sistem de calcul pot fi aflate din fisierele header limits.h și float.h
  - exemplu: CHAR_MAX, INT_MAX, INT_MIN, UINT_MAX

Introducere in Limbajul C

Tipuri de date

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

int main (void) {
   printf ("Un int se aloca pe: %d octeti\n", sizeof(int));
   printf ("Cel mai mic int este: %d\n", INT_MIN);
   printf ("Cel mai mare int este: %d\n\n", INT_MAX);

   printf ("Un double se aloca pe: %d octeti", sizeof(double));
   printf ("\nCel mai mic double este: %e\n", DBL_MIN);
   printf ("Cel mai mare double este: %e\n", DBL_MAX);

   return 0;
}
```

Introducere in Limbajul C

Tipuri de date

5 tipuri de date de baza:
-char, int, float, double, void

Int pe 32 biti (Codeblocks)

Cum verificam?

```
sizeof(int) = 4
 sizeof(short int) =  2
  sizeof(long int) = 4
sizeof(float) = 4
 sizeof(double) =  8
  sizeof(long double) = 8
sizeof(char) = 1
 sizeof(signed char) =  1
  sizeof(unsigned char) = 1
```

Tip	Dimensiune aproximativă în biţi	Domeniu minimal de valori
char	8	de la -127 la 127
unsigned char	8	de la 0 la 255
signed char	8	de la -127 la 127
int	16	de la -32767 la 32767
unsigned int	16	de la 0 la 65535
signed int	16	Similar cu int
short int	16	Similar cu int
unsigned short int	16	de la 0 la 65535
signed short int	16	Similar cu short int
long int	32	de la -2.147.483.647 la 2.147.483.647
signed long int	32	Similar cu long int
unsigned long int	32	de la 0 la 4.294.967.295
float	32	Şase zecimale exacte
double	64	Zece zecimale exacte
long double	80	Zece zecimale exacte

Tabelul 2-1 *Toate tipurile de date definite prin standardul ANSI C*

Introducere in Limbajul C

Tipuri de date

- Numărul de octeți ocupați și domeniile de valori uzuale ale tipurilor de bază

char	1 octet (8 biți) în domeniul -128 până la 127
int	16-biți OS : 2 octeți în domeniul -32768 până la 32767 32-biți OS : 4 octeți în domeniul -2,147,483,648 până la 2,147,483,647
float	4 octeți în domeniul 10^{-38} până la 10^{38} cu precizie de 7 zecimale
double	8 octeți în domeniul 10^{-308} până la 10^{308} cu precizie de 15 zecimale

Introducere in Limbajul C

Tipuri de date – Modificatori de tip

- **signed**
  - modificatorul implicit pentru toate tipurile de date
    - bitul cel mai semnificativ din reprezentarea valorii este semnul

- **unsigned**
  - restricționează valorile numerice memorate la valori pozitive
    - domeniul de valori este mai mare deoarece bitul de semn este liber și participă în reprezentarea valorilor

- **short**
  - reduce dimensiunea tipului de date întreg la jumătate
    - se aplică doar pe întregi

- **long**
  - permite memorarea valorilor care depășesc limita de stocare specifică tipului de date
    - se aplică doar pe int sau double
      - la int dimensiunea tipului de bază se dublează
      - La double se mărește dimensiunea de regulă cu doi octeți (de la 8 la 10 octeți)

- **long long**
  - Introdus in C99 pentru a facilita stocarea unor valori întregi de dimensiuni foarte mari
    - cel puțin 8 octeți.

Introducere in Limbajul C

Expresii - Modelatori de acces

Const

Variabilele de tip const nu pot fi modificate de program (dar pot primi valori initiale).

```
const int a = 10;
```

Modelatorul const poate fi folosit pentru a proteja obiectele indicate de argumentele unei functii pentru a nu fi modificate de acea functie.

Volatile

Valoarea unei variabile poate sa fie modificata pe cai nedeclarate explicit de program (Expl. Adresa unei variabile globale poate fi transmisa rutinei ceasului sistemul de operare si utilizata pentru a pastra timpul real al sistemului → continutul variabilei se modifica fara o instructiune de atribuire explicita.

(Exemple in laborator).

Introducere in Limbajul C

Expresii - Constante ([2])

- Sunt formate din unul mai multe caractere incluse intre apostrofu
  Pentru un caracter, tip constantei este char, i valoarea este reprezen de codul ASCII.
- Daca sint mai multe caractere, intre apostrofuri, tipul constantei este int, iar valoarea depinde de implementare.
- Constantele predefinit sint reprezentate de secventele speciale (escape).

Caractere - ASCII

(c)G.Albeanu

- un singur caracter, intre apostrofuri, 'a'
- secvente speciale (escape – de evitare a situatiilor care ar parea ambigue)

\a	BELL	generator de sunet
\b	BS	backspace
\f	FF	form feed
\n	LF	line feed
\r	CR	carriage return
\t	HT	Horizontal TAB
\v	VT	Vertical TAB
\\	\	backslash
\'	'	apostrof
\"	"	ghilimele
\?	?	semnul ?
'\0'..'\0377'		orice caracter ASCII specificat OCTAL
'\0x0'..'\0xFF'		orice caracter ASCII specificat HEXAZECIMAL

[2] Albeanu G –Programare pr

Introducere in Limbajul C

Expresii - Constante ([2])

Constantele sir de caractere

- String.h
- Au tipul **char**[].
- Au clasa de memorare **static**
- "null-terminating string"
- Un sir de n caractere va avea n+1 octeti
- Unicode (Java – 2 octeti)

Siruri de caractere

(c)G.Albeanu

- Secventa de caractere, inclusiv secvente escape, intre ghilimele.
- Exemplu: "Acesta este un sir\n"
- Au o reprezentare interna speciala. Sunt, de fapt, tablouri de caractere (array of char) care se incheie cu caracterul nul ('\0').

(c)G.Albeanu

Tipuri de date fundamentale

- patru tipuri aritmetice de baza: char, int, float, double
- modificatori de tip – afecteaza domeniul de valori: signed, unsigned, short, long
- Tipul void – indica absenta oricarei valori. Este utilizat pentru: functii fara parametrii, functii fara rezultat (proceduri!), tip pointer generic – conversie de tip cu operatorul cast pentru pointeri.
- sizeof(tip) sau sizeof <expresie> - pentru aflarea dimensiunii zonei de memorie a unui tip sau ocupata de o variabila.

[2] Albeanu G –Programare pr

Introducere in Limbajul C

Operatori

pot fi unari, binari sau ternari, fiecare având o precedenţă şi o asociativitate bine definite

Precedenţă	Operator	Descriere	Asociativitate
1	[]	Indexare	stanga-dreapta
	. şi ->	Selecţie membru (prin structură, respectiv pointer)	stânga-dreapta
	++ şi --	Postincrementare/postdecrementare	stânga-dreapta

Sursa: http://ocw.cs.pub.ro/courses/programare/laboratoare/lab02

Introducere in Limbajul C

Operatori

Precedenţă	Operator	Descriere	Asociativitate
2	!	Negare logică	dreapta-stânga
	~	Complement faţă de 1 pe biţi	dreapta-stânga
	++ şi --	Preincrementare/predecrementare	dreapta-stânga
	+ şi -	+ şi - unari	dreapta-stânga
	*	Dereferenţiere	dreapta-stânga
	&	Operator *adresă*	dreapta-stânga
	(tip)	Conversie de **tip**	dreapta-stânga
	sizeof()	Mărimea în octeţi	dreapta-stânga

Sursa: http://ocw.cs.pub.ro/courses/programare/laboratoare/lab02

Introducere in Limbajul C

Operatori

Precedenţă	Operator	Descriere	Asociativitate
3	*	Înmulţire	stânga-dreapta
	\/	Împărţire	stânga-dreapta
	%	Restul împărţirii	stânga-dreapta
4	+ şi -	Adunare/scădere	stânga-dreapta
5	<< si >>	Deplasare stânga/dreapta a biţilor	stânga-dreapta
6	<	Mai mic	stânga-dreapta
	<=	Mai mic sau egal	stânga-dreapta
	>	Mai mare	stânga-dreapta
	>=	Mai mare sau egal	stânga-dreapta

Sursa: http://ocw.cs.pub.ro/courses/programare/laboratoare/lab02

Introducere in Limbajul C

Operatori

Precedenţă	Operator	Descriere	Asociativitate
7	==	Egal	dreapta
	!=	Diferit	stânga-dreapta
8	&	ŞI pe biţi	stânga-dreapta
9	^	SAU-EXCLUSIV pe biţi	stânga-dreapta
10	|	SAU pe biţi	stânga-dreapta
11	&&	ŞI logic	stânga-dreapta
12	||	SAU logic	stânga-dreapta
13	:?	Operator condiţional	dreapta-stânga

Sursa: http://ocw.cs.pub.ro/courses/programare/laboratoare/lab02

Introducere in Limbajul C

Operatori

Precedenţă	Operator	Descriere	Asociativitate
14	=	Atribuire	dreapta-stânga
	+= şi -=	Atribuire cu adunare/scădere	dreapta-stânga
	*= şi /=	Atribuire cu multiplicare/împărţire	dreapta-stânga
	%=	Atribuire cu modulo	dreapta-stânga
	&= si |=	Atribuire cu ŞI/SAU	dreapta-stânga
	^=	Atribuire cu SAU-EXCLUSIV	dreapta-stânga
	<<= şi >>=	Atribuire cu deplasare de biţi	dreapta-stânga
15	,	Operator secvenţa	stânga-dreapta

Sursa: http://ocw.cs.pub.ro/courses/programare/laboratoare/lab02

Introducere in Limbajul C

Functii de citire / scriere cu format

- Funcțiile **printf** și **scanf** permit controlul formatului în care se scriu respectiv se citesc datele

- **printf**
  - afișează un șir de caractere la ieșirea standard – implicit pe ecranul monitorului
  - dacă șirul conține specificatori de format, atunci argumentele adiționale (care urmează după șir) sunt formatate în concordanță cu specificatorii de format (subșir care începe cu caracterul %) și inserate în locul și pe pozițiile acestora din cadrul șirului

Sursa: http://ocw.cs.pub.ro/courses/programare/laboratoare/lab02

Introducere in Limbajul C

Functii de citire / scriere cu format

- **scanf**
  - citește date de intrare (implicit de la tastatura) în formatul indicat de șirul de formatare și le salvează la adresele indicate de argumentele adiționale (variabilele de intrare)

- Șirul de formatare poate include următoarele elemente:
  - Spațiu alb funcția citește și ignoră spațiile albe (spațiu, tab, linie nouă) înaintea următorului caracter diferit de spațiu
    - un singur spațiu în șirul de formatare se suprapune asupra oricâtor spații din șirul introdus, inclusiv asupra nici unui spațiu
  - Caracter diferit de spațiu cu excepția caracterului %: funcția citește următorul caracter de la intrare și îl compară cu caracterul specificat în șirul de formatare
    - Dacă se potrivește, funcția are succes și trece mai departe la citirea următorului caracter din intrare
    - Dacă nu se potrivește, funcția eșuează și lasă următoarele caractere din intrare nepreluate

Introducere in Limbajul C

Functii de citire / scriere cu format

– specificatorii de format încep cu % urmat de

Specificator	Format
d	Întreg cu semn în baza 10
i	Întreg cu semn în baza 8, 10 sau 16
U	Întreg fără semn în baza 10
O	Întreg fără semn în baza 8
x	Întreg fără semn în baza 16
X	Întreg fără semn în baza 16 (cu majuscule)
f	Real în simplă precizie în baza 10
e	Real cu notație științifică (mantisă și exponent)
E	Real cu notație științifică (mantisă și exponent) cu majuscule
g	Real în reprezentarea mai scurtă (%e sau %f)
G	Real în reprezentarea mai scurtă (%e sau %f) cu majuscule
c	Caracter
s	Șir de caractere
p	Adresa (pointer)

Concluzii

1. S-au trecut in revista urmatoarele aspecte:

- Algoritmi. Structuri de control si structuri elementare de date

- Limbaje de programare

- Introducerea in Limbajul C

Perspective

Cursul 3:

- Complexitatea si corectitudinea algoritmilor