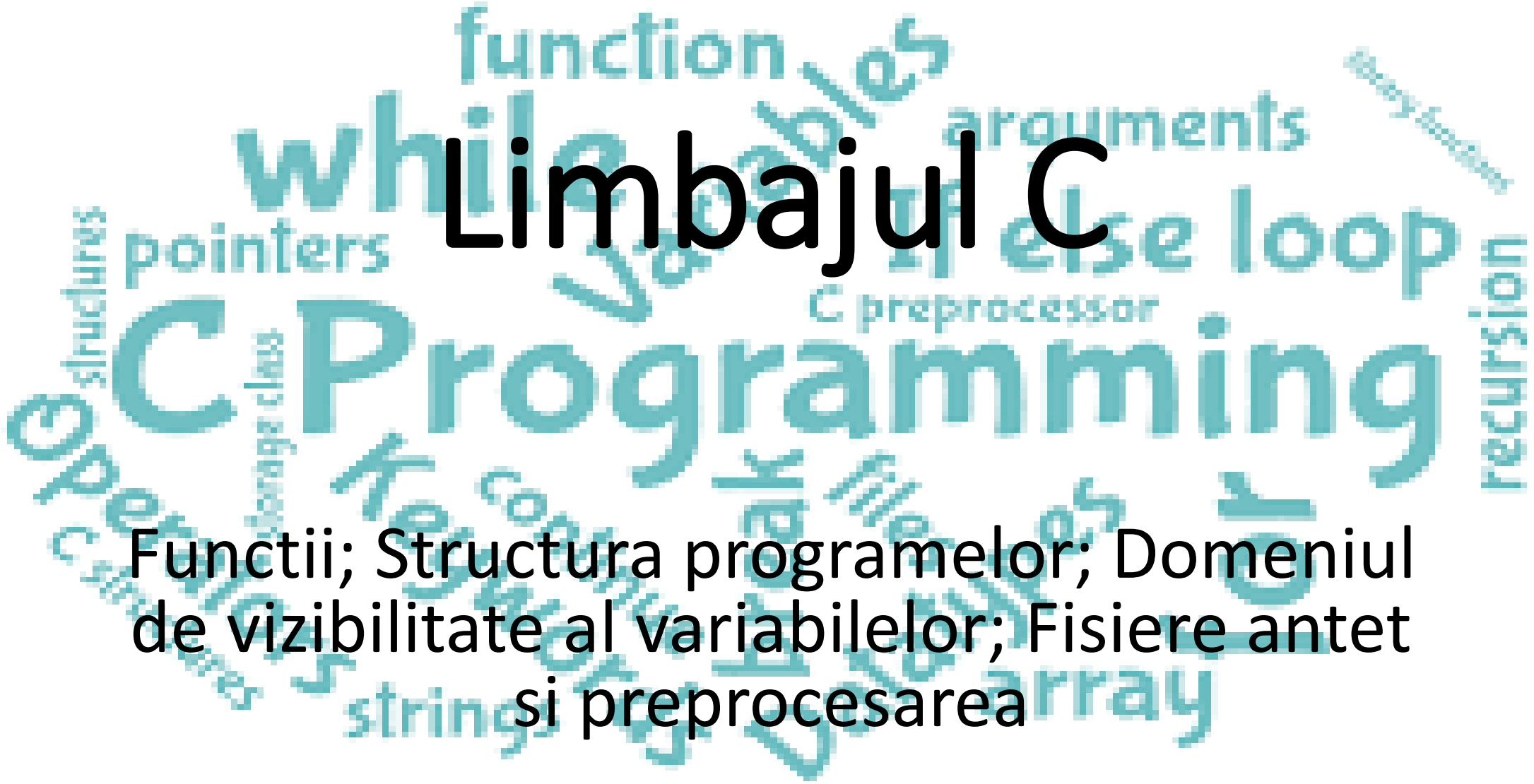


Limbajul C

Functii; Structura programelor; Domeniul de vizibilitate al variabilelor; Fisiere antet si preprocesarea

Cuprins

- Functii C
- Categorii de variabile:
  - Variabile externe
  - Variabile statice
  - Variabile registru
  - Variabile declarate intr-un bloc
- Domenii de vizibilitate ale variabilelor
- Fisiere antet
- Preprocesorul C

Functii C

- Functiile au rolul de a diviza programele mari in mai multe componente mai mici;
- Functiile pot ascunde adesea detalii ale operatiilor din parti ale programului pe care nu e nevoie sa le cunoastem;
- Limbajul C a fost proiectat pentru a face functiile eficiente si usor de folosit;
- Programele C constau, in general, din numeroase functii mici (mai degraba decat din cateva functii mari);
- Avantaje: programul este usor de citit, erorile pot fi corectate mai usor, programul este usor de intretinut;

Functii C

- Fiecare functie are forma:

```
tip_data_ intors nume_functie (lista de argumente, daca exista)  {
        declaratii si instructiuni, daca exista
}
```

- Exemplu:

```
double sum(double x, double y)  {
    double z;
    z = x + y;
    return z;
}
```

- Cea mai simpla functie (nu face nimic):

```
void dummy()  {}
```

Functii C

- Comunicarea intre functii este facuta prin argumente si valori returnate de functii;
- Ea poate fi facuta deasemenea prin variabile externe (vom vedea in acest curs ce sunt variabilele externe);
- Instructiunea ***return*** este mecanismul de returnare a unei valori din functia apelata in apelant:

```
return (expresie);
```

- Implicit (daca nu se specifica altfel) se presupune ca o functie returneaza o valoare de tip *int*;

Functii C

- Valoarea expresiei din:

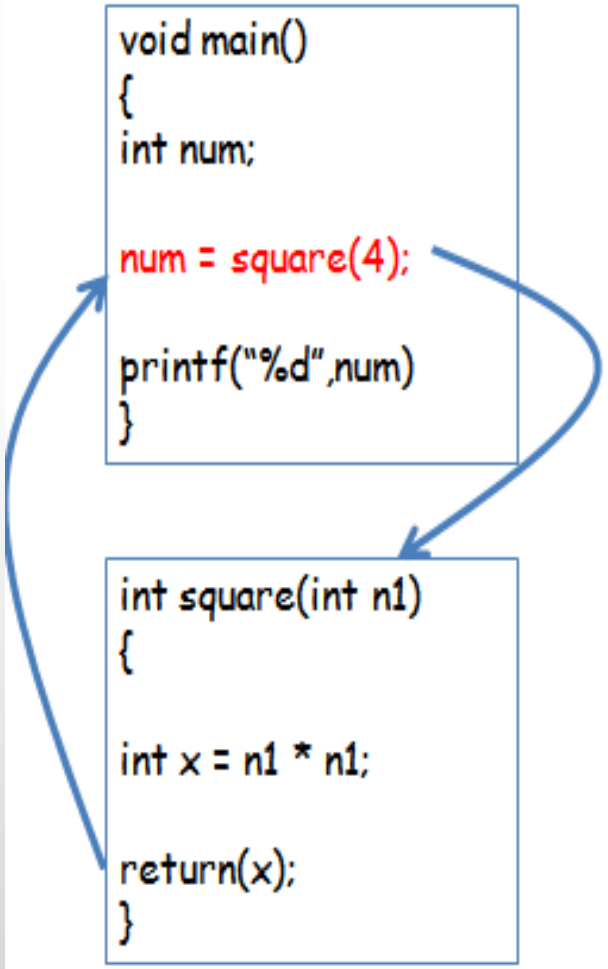
    return (expresie);

este intodeauna convertita la tipul functiei (tipul de data declarant al functiei) inainte ca rezultatul sa fie returnat in apelant;

Argumentele functiilor. Transmitere prin valoare si referinta

- Argumentele functiilor sunt trimise prin **valoare**, adica functia apelata primeste o copie temporara si privata a fiecarui argument si nu adresele lor;
- Consecinta: functia nu poate afecta argumentul original din functia apelanta;
- ATENTIE!!! Exista o exceptie: cand un nume de tablou apare ca argument al unei functii este transmisa de fapt locatia de inceput a tabloului (elementele nu sunt copiate) !!!
- Consecinta: functia poate altera elementele tabloului. Efectul este ca tablourile sunt trimise prin **referinta**!

Mecanismul de transmitere al argumentelor



Definitia functiei

Apelul functiei

```
#include <stdio.h>

/*
 * Print a simple greeting.
 */

void sayHello ( void )
{
  printf("Hello World!\n");
}

/*
 * Call a function which prints a
 * simple greeting.
 */

int main()
{
  sayHello();

  return 0;
}
```

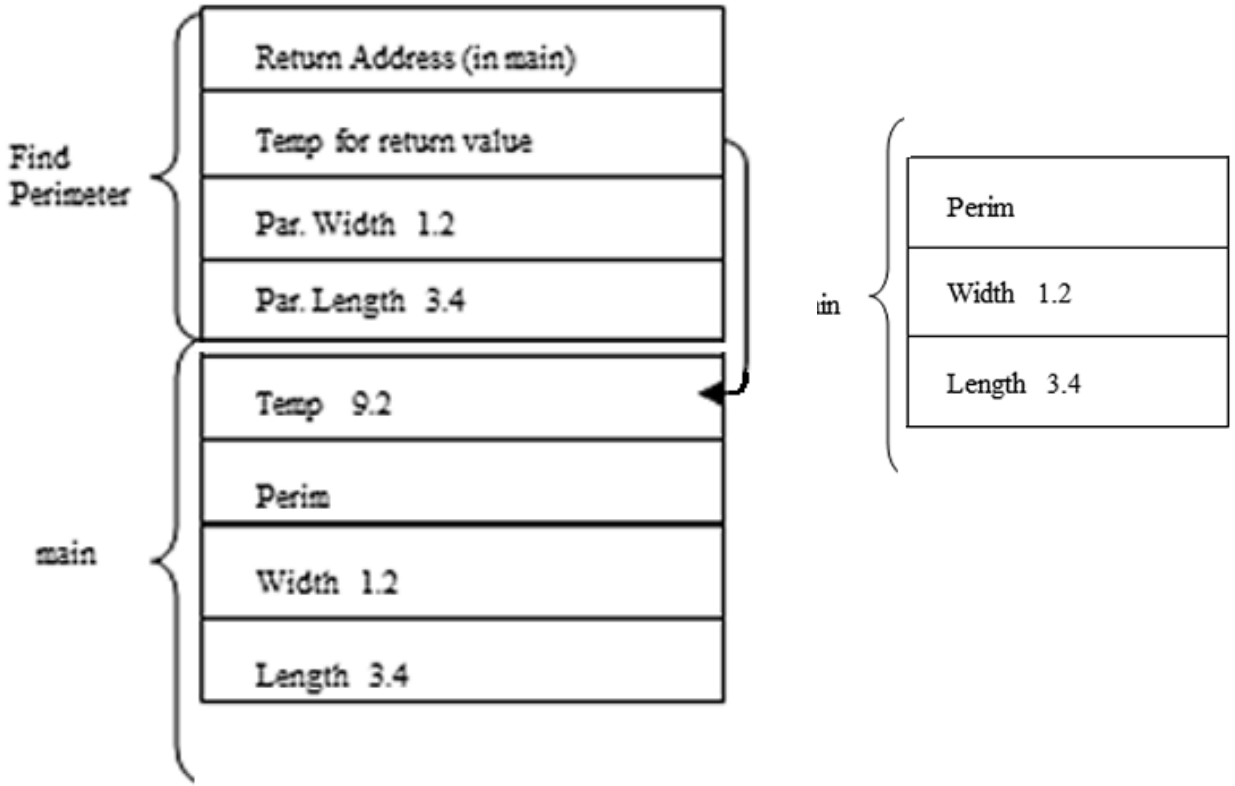
Mecanismul de transmitere al argumentelor

```
float FindPerimeter(float Length, float Width);

int main(void)   {
   float Length, Width, Perim;
   Length = 3.4;
   Width = 1.2;
   Perim = FindPerimeter(Length, Width);
}

float FindPerimeter(float Length, float Width) {
   return 2 * (Length + Width);
}
```

Mecanismul de transmitere al argumentelor



Categorii de variabile. Variabile externe

- Variabilele externe sunt definite in afara oricarei functii si sunt astfel disponibile potential pentru mai multe functii;
- Functiile insesi sunt intodeauna externe, deoarece limbajul C nu permite definitii de functii in interiorul altor functii;
- Deoarece variabilele externe sunt global accesibile, ele ofera o alternativa la argumente de functii si valori returnate pentru comunicari de date intre functii;
- Aceasta modalitate trebuie totusi utilizata cu grija, deoarece ea poate avea efecte negative asupra structurii programului si poate conduce la programe cu multe conexiuni de date intre functii;

Categorii de variabile. Variabile externe

- Alt motiv pentru folosirea variabilelor externe este domeniul si timpul lor de viata;
- Variabilele automate sunt interne unei functii: ele sunt create atunci cand functia este executata si dispar atunci cand functia se termina;
- Variabilele externe, pe de alta parte, sunt permanente;
- Ele nu sunt create si dispar odata cu functia care le contine, ele retin valorile de la un apel de functie la altul;
- Daca doua functii trebuie sa-si partajeze niste date (chiar nefolosite de alte functii niciodata) este adesea convenabil daca datele partajabile sunt pastrate in variabile externe decat trimise via argumente;

Categorii de variabile. Variabile externe

- Daca o variabila externa trebuie sa fie referita inainte de a fi definita sau este definita intr-un alt fisier sursa decat cel in care este folosita, atunci este necesara o declaratie "***extern***“;
- Este important sa distingem intre declaratia unei variabile externe si definitia sa;
- O declaratie anunta proprietatile unei variabile (tipul, marimea, etc);
- O definitie provoaca in plus o alocare de memorie;

Categorii de variabile. Variabile externe

- Exemplu:

```
int x;
int main(void)  {
   x = 10;
   return 0;
}
```

x este definita si declarata implicit global (in afara oricarei functii).

Categorii de variabile. Variabile externe

- Exemplu:

```
extern int x;
int main(void)  {
  x = 10;
  return 0;
}
```

- La compilare se va semnala o eroare: x este declarata dar nu este definita nicaieri!

Categorii de variabile. Variabile externe

```
#include "file.h"
extern int x;  //declaratia variabilei x
int main(void)  {
 x = 10;
 return 0;
}
```

Fisierul file.h contine:

```
int x;  //definitia variabilei x
……
```

Categorii de variabile. Variabile statice

- Variabilele statice sunt a treia clasa de variabile, pe langa cele externe si cele automate;
- Variabilele de tip "static" pot fi atat interne cat si externe;
- Variabilele statice sunt locale unei functii particulare la fel ca cele automate dar, spre deosebire de acestea, ele exista tot timpul si nu apar si dispar de fiecare data cand functia este activa sau se termina;
- O variabila statica externa este recunoscuta in restul fisierului sursa in care este declarata, dar nu intr-un alt fisier;

Categorii de variabile. Variabile statice

- Exemplu de variabila statica

```
static int x = 15;
```

- Exemplul 2:

```
#include<stdio.h>
int fun()  {
  static int count = 0;
  count++;
  return count;
}
int main()  {
  printf("%d ", fun());
  printf("%d ", fun());
  return 0;
}
```

Programul va printa 1 2

Categorii de variabile. Variabile registru

- Declaratia unei variabile registru este de forma:

```
register int x;
register char c;
```

- Declaratia unei variabile registru poate fi aplicata numai variabilelor automate si parametrilor formali ai unei functii;
- Se foloseste indeosebi pentru optimizarea performantelor programelor;

Categorii de variabile. Variabile declarate in blocuri

- O variabila poate fi declarata in interiorul unui bloc ca in exemplul de mai jos:

```
if (n > 0) {
        int i;   /* declara i */
        for (i = 0; i < n; i++)
            ...

    }
```

- La iesirea din bloc, variabila i nu va mai exista;

Domeniul de vizibilitate al unei variabile

- Domeniul de vizibilitate al unui nume (de variabila) este acea parte de program in care numele este definit;
- Pentru o variabila automata declarata la inceputul unei functii, domeniul este functia in care numele este declarant; variabilele cu acelasi nume dar in functii diferite sunt fara legatura unele cu altele;
- Domeniul unei variabile externe dureaza din punctul in care ea este declarata intr-un fisier sursa pana la sfarsitul acelui fisier:

```
int sp = 0;
double val[MAXVAL];
double push(f) {...}
double pop() {...}
clear() {...}
```

Initializarea variabilelor

- In absenta initializarii explicite, variabilele externe si statice se initializeaza pe zero;
- Variabilele automate si registru sunt nedefinite (i.e. garbage);
- Pentru variabilele externe si statice, initializarea se face o singura data, la compilare;
- Tablourile automate *nu pot fi initializate*;
- Tablourile externe si statice pot fi initializate punand dupa declaratie o lista de valori de initializare inclusa intre paranteze si separate prin virgule;

Initializarea variabilelor

```
int nwhite = 0;
int nother = 0;
int ndigit[10] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } ;
char pattern[] = "the“;
char pattern[] = { 't', 'h', 'e', '\0' };

main() {
    int c, i;
    ...
}
```

Preprocesorul C

- C admite unele extensii de limbaj cu ajutorul unui simplu macropreprocesor;
- Posibilitatile lui #define sunt cele mai obisnuite exemple despre aceste extensii;
- Alta posibilitate este de a include continutul altor fisiere in timpul compilarii ca in exemplul de mai jos:

```
#include "filename"
```

Preprocesorul C

- Exemplu:

```
#define then
#define begin {
#define end   ;}
```

- Cum se va expanda urmatoarea secventa?

```
if (i > 0) then
    begin
        a = 1;
        b = 2
    end
```

Preprocesorul C

- Este de asemenea posibil de definit macro instructiuni cu argumente (“macro-uri”, astfel ca textul de inlocuire depinde de felul in care macro-ul este apelat;
- De exemplu sa definim un macro numit *max* astfel:

```
#define max(A, B)   ((A) > (B) ? (A) : (B))
```

- Linia

```
x = max(p+q, r+s);
```

- va fi inlocuita de linia:

```
x = ((p+q) > (r+s) ? (p+q) : (r+s));
```