

Programarea calculatoarelor

FMI

Secția Calculatoare și tehnologia informației, anul I

Cursul 5 / 04.11.2024

Programa cursului

❑ Introducere

- Algoritmi
- Limbaje de programare.

❑ Fundamentele limbajului C

- Introducere în limbajul C. Structura unui program C.
- Tipuri de date fundamentale. Variabile. Constante. Operatori. Expresii. Conversii.
- Tipuri derivate de date: pointeri, tablouri, șiruri de caractere, structuri, uniuni, câmpuri de biți, enumerări
- Instrucțiuni de control
- Directive de preprocesare. Macrodefiniții.
- **Funcții de citire/scriere.**

❑ Fișiere text

- **Funcții specifice de manipulare.**

❑ Funcții (1)

- **Declarare și definire. Apel. Metode de transmitere a paramerilor.** Pointeri la funcții.

❑ Tablouri și pointeri

- Legătura dintre tablouri și pointeri
- Aritmetica pointerilor
- Alocarea dinamică a memoriei
- Clase de memorare

❑ Șiruri de caractere

- Funcții specifice de manipulare.

Fișiere binare

- Funcții specifice de manipulare.

❑ Structuri de date complexe și autoreferite

- Definire și utilizare

❑ Funcții (2)

- Funcții cu număr variabil de argumente.
- Preluarea argumentelor funcției main din linia de comandă.

Cuprinsul cursului de azi

- 1. Funcții de citire/ scriere**
2. Fișiere text. Funcții specifice de manipulare.
3. Funcții

Funcții de citire/ scriere

Cuprins

- Funcțiile getchar și putchar
- Funcțiile gets/ fgets și puts
- Funcția printf. Modelatori de format
- Funcția scanf

Funcții de citire și scriere

- ❑ operații de citire și scriere în C:
 - ❑ de la tastatură (stdin) și la ecran (stdout);
 - ❑ prin fișiere;
 - ❑ efectuate cu ajutorul funcțiilor de bibliotecă
- ❑ citirea de la tastatură și scrierea la ecran
 - ❑ fără formatare:
getchar, putchar, getch, getche, putch, gets, puts
 - ❑ cu formatare: scanf, printf
 - ❑ incluse în biblioteci:
 - ❑ stdio.h (getchar, putchar, gets, puts, scanf, printf)
 - ❑ conio.h (getch, getche, putch)

Obs. CODE::BLOCKS nu include biblioteca conio.h

Funcțiile getchar și putchar

- operații de citire și scriere a caracterelor:

- **int getchar(void)**

- citește un caracter de la tastatură

- așteaptă până este apasată o tastă și returnează valoarea sa → tasta apăsată are imediat ecou pe ecran.

- **int putchar(int c)** - scrie un caracter pe ecran în poziția curentă a cursorului

- fișierul antet pentru aceste funcții: **stdio.h**.

Funcțiile gets și puts

□ operații de citire și scriere a șirurilor de caractere:

□ `char *gets(char *s)`

□ citește caractere din **stdin** și le depune în zona de date de la adresa s, până la apăsarea tastei Enter. În șir, tastei Enter îi va corespunde caracterul `'\0'`.

□ **dacă operația de citire reușește, funcția întoarce adresa șirului, altfel valoarea NULL (= 0)**

□ `int puts(const char *s)`

□ scrie pe ecran șirul de la adresa s sau o constantă șir de caractere și apoi trece la linie nouă.

□ **dacă operația de scriere reușește, funcția întoarce ultimul caracter, altfel valoarea EOF (-1).**

□ fișierul antet pentru aceste funcții este **stdio.h**

Funcțiile gets și puts

□ exemplu:

```
main.c ✕
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  int main(){
6
7      char sir[10];
8
9      printf("Ce zi e astazi: ");
10     gets(sir);
11     puts(sir);
12
13     puts("Alegeti DA sau NU. Optiunea dumneavoastra este: ");
14     gets(sir);
15     puts(sir);
16
17
18     return 0;
19 }
20
```

```
Ce zi e astazi: joi
joi
Alegeti DA sau NU. Optiunea dumneavoastra este:
DA
DA
```


Funcția gets. Dezavantaje

- ❑ **char *gets(char *s)**

- ❑ primește ca input numai un buffer (s), nu știm dimensiunea lui

- ❑ problema de buffer overflow: citim în s mai mult decât dimensiunea lui, gets nu ne împiedică, scrie datele în altă parte

- ❑ Soluție: funcția fgets:

- char *fgets(char *s, int size, FILE *stream)**

- ❑ fgets(buffer, sizeof(buffer), stdin);

Obs. În standardul C11 funcția gets este eliminată.

Funcțiile printf și scanf

- ❑ funcții de citire (scanf) și scriere (printf) cu formatare;
- ❑ formatarea specifică conversia datelor de la reprezentarea externă în reprezentarea internă (scanf) și invers (printf);
- ❑ formatarea se realizează pe baza descriptorilor de format

%[flags][width][.precision][length]specifier

- ❑ detalii aici:

<http://www.cplusplus.com/reference/cstdio/printf/>

Funcțiile printf și scanf

- formatarea se realizează pe baza descriptorilor de format

`%[flags][width][.precision][length]specifier`

Specificator de format	Reprezentare
<code>%c</code>	caracter
<code>%s</code>	șir de caractere
<code>%d, %i</code>	întreg în zecimal
<code>%u</code>	întreg în zecimal fără semn
<code>%o</code>	întreg în octal
<code>%x</code>	întreg în hexazecimal fără semn (litere mici)
<code>%X</code>	întreg în hexazecimal fără semn (litere mari)
<code>%f</code>	număr real în virgulă mobilă
<code>%e, %E</code>	notație științifică - o cifră la parte întreagă
<code>%ld, %li, %lu, %lo, %lx</code>	cu semnificațiile de mai sus, pentru întregi lungi
<code>%p</code>	pointer

Funcția printf

□ prototipul funcției:

int printf(const char *format, argument1, argument2, ...);

unde:

- **format** este un șir de caractere ce definește textele și formatele datelor care se scriu pe ecran
- **argument1, argument2,...** sunt expresii. Valorile lor se scriu pe ecran conform specificatorilor de format prezenți în format
- funcția **printf** întoarce numărul de octeți transferați sau EOF (-1) în caz de eșec.

Funcția printf

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(){
5
6      char sir[10] = "Azi e joi";
7      printf("Primul caracter din sirul \"%s\" este %c \n",sir,sir[0]);
8      int x = 1234;
9      printf("Reprezentare lui x in baza 10: x=%d\n",x);
10     printf("Reprezentare lui x in baza 8: x=%o\n",x);
11     printf("Reprezentare lui x in baza 16 (litere mici): x=%x\n",x);
12     printf("Reprezentare lui x in baza 16 (litere mari): x=%X\n",x);
13
14     float y = 12.34;
15     printf("Reprezentare lui y ca numar real: y=%f\n",y);
16     printf("Reprezentare lui y in notatie stiintifica: y=%e\n",y);
17     printf("Reprezentare lui y in notatie stiintifica: y=%E\n",y);
18
19     return 0;
20 }
21
```

```
Primul caracter din sirul "Azi e joi" este A
Reprezentare lui x in baza 10: x=1234
Reprezentare lui x in baza 8: x=2322
Reprezentare lui x in baza 16 (litere mici): x=4d2
Reprezentare lui x in baza 16 (litere mari): x=4D2
Reprezentare lui y ca numar real: y=12.340000
Reprezentare lui y in notatie stiintifica: y=1.234000e+01
Reprezentare lui y in notatie stiintifica: y=1.234000E+01
```

Funcția printf

main.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 int main() {
6
7     int a = printf("Ce zi e astazi: ");
8     printf("\n a = %d \n", a);
9
10    a = printf("Alegeti DA sau NU. Optiunea dumneavoastra este: ");
11    printf("\n a = %d \n", a);
12
13    return 0;
14 }
```

Număr de octeți:

Ce zi e astazi:
a = 16
Alegeti DA sau NU. Optiunea dumneavoastra este:
a = 48

Modelatori de format

- ❑ mulți specificatori de format pot accepta modelatori care modifică ușor semnificația lor:
 - ❑ alinierea la stânga
 - ❑ minim de mărime a câmpului
 - ❑ numărul de cifre zecimale
- ❑ modelatorul de format se află între semnul procent și codul pentru format:
 - ❑ caracterul ‘—’ specifică aliniere la stânga;
 - ❑ șir de cifre zecimale specifică dimensiunea câmpului pentru afișare
 - ❑ caracterul ‘.’ urmat de cifre specifică precizia reprezentării

Modelatorul flags

□ formatarea se realizează pe baza descriptorilor de format

%[flags][width][.precision][length]specifier

flags	description
-	Left-justify within the given field width; Right justification is the default (see <i>width</i> sub-specifier).
+	Forces to precede the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign.
(space)	If no sign is going to be written, a blank space is inserted before the value.
#	Used with o, x or X specifiers the value is preceded with 0, 0x or 0X respectively for values different than zero. Used with a, A, e, E, f, F, g or G it forces the written output to contain a decimal point even if no more digits follow. By default, if no digits follow, no decimal point is written.
0	Left-pads the number with zeroes (0) instead of spaces when padding is specified (see <i>width</i> sub-specifier).

Modelatorul width

- formatarea se realizează pe baza descriptorilor de format

`%[flags][width][.precision][length]specifier`

<i>width</i>	<i>description</i>
<i>(number)</i>	Minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger.
<i>*</i>	The <i>width</i> is not specified in the <i>format</i> string, but as an additional integer value argument preceding the argument that has to be formatted.

Modelatorul precision

- formatarea se realizează pe baza descriptorilor de format

`%[flags][width][.precision][length]specifier`

<i>.precision</i>	description
<i>.number</i>	<p>For integer specifiers (<i>d</i>, <i>i</i>, <i>o</i>, <i>u</i>, <i>x</i>, <i>X</i>): <i>precision</i> specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A <i>precision</i> of 0 means that no character is written for the value 0.</p> <p>For <i>a</i>, <i>A</i>, <i>e</i>, <i>E</i>, <i>f</i> and <i>F</i> specifiers: this is the number of digits to be printed after the decimal point (by default, this is 6).</p> <p>For <i>g</i> and <i>G</i> specifiers: This is the maximum number of significant digits to be printed.</p> <p>For <i>s</i>: this is the maximum number of characters to be printed. By default all characters are printed until the ending null character is encountered.</p> <p>If the period is specified without an explicit value for <i>precision</i>, 0 is assumed.</p>
<i>.*</i>	The <i>precision</i> is not specified in the <i>format</i> string, but as an additional integer value argument preceding the argument that has to be formatted.

Modelatorul length

□ formatarea se realizează pe baza descriptorilor de format

`%[flags][width][.precision][length]specifier`

	specifiers						
<i>length</i>	<i>d i</i>	<i>u o x X</i>	<i>f F e E g G a A</i>	<i>c</i>	<i>s</i>	<i>p</i>	<i>n</i>
<i>(none)</i>	int	unsigned int	double	int	char*	void*	int*
hh	signed char	unsigned char					signed char*
h	short int	unsigned short int					short int*
l	long int	unsigned long int		wint_t	wchar_t*		long int*
ll	long long int	unsigned long long int					long long int*
j	intmax_t	uintmax_t					intmax_t*
z	size_t	size_t					size_t*
t	ptrdiff_t	ptrdiff_t					ptrdiff_t*
L			long double				

Modelatori de format pentru printf

main.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 int main(){
6
7     double numar;
8     numar = 10.1234;
9
10    printf("numar=%f\n",numar);
11    printf("numar=%10f\n",numar);
12    printf("numar=%012f\n",numar);
13
14    printf("%.4f\n",123.1234567);
15    printf("%3.8d\n",1000);
16    printf("%10d\n",1000);
17    printf("%-10d\n",1000);
18    printf("%10.15s\n","Acesta este un test simplu");
19
20    return 0;
21 }
22
```

```
numar=10.123400
numar= 10.123400
numar=00010.123400
123.1235
00001000
      1000
1000
Acesta este un
```

Funcția printf

□ exemplu:

```
printf("valoarea lui x este: %-4.2f\n", 3.14);  
printf("x=%i, y=%f, x=%o, x=%#x\n", 15, 3.14, 15, 15);  
printf("c= %c, c=%d\n", '%', '%');  
printf("sir de caractere: %s\n", "ana are mere");  
printf("\\ \\ \" \' \n");
```

```
valoarea lui x este: 3.14  
x=15, y=3.140000, x=17, x=0xf  
c= %, c=37  
sir de caractere: ana are mere  
\\ \" \'
```

Funcția scanf

□ prototipul funcției:

int scanf(const char *format ,adresa1, adresa2, ...);

unde:

- **format** este un șir de caractere ce definește textele și formatele datelor care se citesc de la tastatură
- **adresa1, adresa2,...** sunt adresele zonelor din memorie în care se păstrează datele citite după ce au fost convertite din reprezentarea lor externă în reprezentare internă.
- funcția **scanf** întoarce numărul de câmpuri citite și depuse la adresele din listă. Dacă nu s-a stocat nici o valoare, funcția întoarce 0.

Funcția scanf

- ❑ șirul de formatare (format) poate include următoarele elemente:
 - ❑ **spațiu alb**: funcția citește și ignoră spațiile albe (spațiu, tab, linie nouă) înaintea următorului caracter diferit de 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

Funcția scanf

□ exemplu:

```
main.c ✕  
1 #include <stdio.h>  
2 #include <stdlib.h>  
3  
4  
5 int main()  
6 {  
7     int n,a;  
8     float m;  
9  
10    scanf("n=%d m=%f",&n,&m);  
11    printf("n=%d\nm=%f\n",n,m);  
12  
13    printf("n=");  
14    scanf("%d",&n);  
15  
16    printf("m=");  
17    a = scanf("%f",&m);  
18    printf("a = %d \n",a);  
19  
20    return 0;  
21 }  
22
```

```
n=25 m=3.2  
n=25  
m=3.200000  
n=100  
m=i37  
a = 0
```


Funcția scanf

□ exemplu:

```
int main()
{
    int a, b;
    char c;
    for(;;)
    {
        printf("Introduceti 2 numere intregi\n");
        if(scanf("%d%d",&a,&b)==2)
            break;
        else
            while(c=getchar()!='\n' && c!=EOF);
    }
    printf("am citit 2 numere: %d si %d\n",a,b);
    return 0;
}
```

```
Introduceti 2 numere intregi
3 a
Introduceti 2 numere intregi
4 2 3
am citit 2 numere: 4 si 2
```

Cuprinsul cursului de azi

1. Funcții de citire/ scriere
- 2. Fișiere text. Funcții specifice de manipulare**
3. Funcții

Fișiere

- ❑ **fișier = șir de octeți (colecție de date) memorat pe suport extern (magnetic, optic) și identificat printr-un nume.**
- ❑ fișierele sunt entități ale sistemului de operare.
- ❑ operațiile cu fișiere se realizează de către sistemul de operare, compilatorul de C traduce funcțiile de acces la fișiere în apeluri ale funcțiilor de sistem; alte limbaje de programare fac același lucru;

Fișiere

- noțiunea de fișier este mai generală:
- **fișier = flux de date (stream) = transfer de informație binară (șir de octeți) de la o sursă spre o destinație:**
 - **citire:** flux de la tastatură (sursă) către memoria internă (destinație)
 - **afișare:** flux de la memoria internă (sursă) către periferice (monitor, imprimantă)

Fluxuri care se deschid automat în program

- ❑ **stdin (standard input)** – flux de intrare (citire)
 - ❑ asociat implicit cu tastatura
- ❑ **stdout (standard output)** – flux de ieșire (afișare)
 - ❑ asociat implicit cu ecranul
- ❑ **stderr (standard error)** – flux de ieșire (afișare) pentru erori
 - ❑ asociat implicit cu ecranul

Tipuri de fișiere

1. **fișiere text:** fiecare octet este interpretat drept caracter cu codul ASCII dat de octetul respectiv
 - octeții (caractere ASCII) sunt organizați pe linii.
 - un fișier text **poate** fi terminat printr-un caracter terminator de fișier (**EOF = CTRL-Z**). Caracterele terminatorii de linii sunt:
Windows: **CR + LF = '\r\n'** (\r are codul ASCII 13, \n are codul ASCII 10, dar **se returnează doar \n**)
 - Terminatorul nu este obligatoriu. Sfârșitul unui fișier disc poate fi detectat și pe baza lungimii fișierului (număr de octeți), memorată pe disc.
 - **Exemplu:** un fișier text în care scriem numărul întreg 123 ocupă trei octeți (codul ASCII pt 1, codul ASCII pt 2, codul ASCII pt 3)

Tipuri de fișiere

- 2. **fișiere binare:** octeții nu sunt organizați în nici un fel
 - ❑ nu există noțiunea de linie
 - ❑ **Exemplu:** un fișier binar în care scriem numărul întreg 123 ocupă 4 octeți (scrierea binară a lui 123 în baza 2 pentru un int)

Lucrul cu fișiere

- ❑ în biblioteca `stdio.h` sunt definite (pe lângă `scanf` și `printf`):
 - ❑ o structură **FILE** și
 - ❑ funcțiile necesare
 - ❑ deschiderii,
 - ❑ închiderii,
 - ❑ scrierii și
 - ❑ citirii din fișiere

Lucrul cu fișiere

- ❑ declararea unui **pointer la structura FILE** = realizarea legăturii dintre nivelele logic (variabila fișier) și fizic (numele extern al fișierului) :

FILE * f;

- ❑ Cererea de deschidere a unui fișier:
 - ❑ fișierul a putut fi deschis:
 - ❑ Pointer-ul la **FILE** nu este **NULL**;
 - ❑ Se prelucrează fișierul și se închide;
 - ❑ fișierul nu a putut fi deschis:
 - ❑ Pointer-ul la **FILE** este **NULL**;
 - ❑ Nu se poate continua cu prelucrarea;
 - ❑ Nu este necesară închiderea ;

```

main.c  untitled_2.c  untitled_3.c
1  #include <stdio.h>
2  #include <errno.h>
3  #include <string.h>
4  #define CLOSE_FILE 1
5  void verify_file(FILE *f)
6  {
7      if (f!=NULL)
8      {
9          printf("%p\n",f);
10         if (CLOSE_FILE)
11             fclose(f);
12     }
13     else
14     {
15         printf("Nu am putut deschide fisierul! Eroare %s\n",strerror(errno));
16     }
17 }
18
19 int main()
20 {
21     FILE *f1, *f2, *f3;
22     f1=fopen("untitled_2.c", "r");
23     verify_file(f1);
24     f2=fopen("untitled_3.c", "r");
25     verify_file(f2);
26     f2=fopen("untitled_30.c", "r");
27     verify_file(f2);
28     printf("Numarul maxim de fisiere deschise: %d",FOPEN_MAX);
29
30     return 0;
31 }

```

```
1 #include <stdio.h>
2 #include <errno.h>
3 #include <string.h>
4 #define CLOSE_FILE 1
5 void verify_file(FILE *f)
6 {
7     if (f!=NULL)
8     {
9         printf("%p\n",f);
10        if (CLOSE_FILE)
11            fclose(f);
12    }
13    else
14    {
15        printf("Nu am putut deschide fisierul! Eroare %s\n",strerror(errno));
16    }
17 }
18
19 int main()
20 {
21     FILE *f1, *f2, *f3;
22     f1=fopen("untitled_2.c", "r");
23     verify_file(f1);
24     f2=fopen("untitled_3.c", "r");
25     verify_file(f2);
26     f2=fopen("untitled_30.c", "r");
27     verify_file(f2);
28     printf("Numarul maxim de fisiere deschise: %d",FOPEN_MAX);
29
30     return 0;
31 }
```

0x560faa1e32a0

0x560faa1e32a0

Nu am putut deschide fisierul! Eroare No such file or directory

Numarul maxim de fisiere deschise: 16

Lucrul cu fișiere

- ❑ deschiderea unui fișier = stabilirea unui flux către acel fișier.

Se realizează folosind funcția **fopen**:

FILE *fopen(char *nume_fisier, char *mod_acces)

- ❑ **nume_fisier** = numele fisierului
- ❑ **mod_acces**= șir de 1-3 caractere ce indica tipul de acces :
 - ❑ citire "**r**", scriere "**w**", adăugare la sfârșitul fișierului "**a**";
 - ❑ "**+**" permite scrierea și citirea "**r+**", "**w+**", "**a+**";
 - ❑ **t** (implicit) sau **b**: specifică tipul de fișier (text sau binar).
- ❑ funcția **fopen** întoarce un pointer la o structura **FILE** sau în caz de eroare (fișierul nu se poate deschide) întoarce **NULL**.

Lucrul cu fișiere

Moduri de prelucrare a fișierelor text

Mod	Descriere
r	Deschiderea fișierului pentru citire. Fișierul trebuie să existe!
w	Crearea unui fișier pentru scriere. Dacă fișierul există, conținutul acestuia este șters în totalitate!
a	Deschiderea sau crearea (dacă nu există) unui fișier pentru adăugarea de conținut numai la sfârșitul acestuia
r+	Deschiderea unui fișier pentru actualizarea conținutului (citire și scriere). Fișierul trebuie să existe!
w+	Deschiderea unui fișier pentru actualizarea conținutului (citire și scriere). Dacă fișierul există, conținutul acestuia este șters în totalitate!
a+	Deschiderea unui fișier pentru citirea conținutului și adăugarea de conținut numai la sfârșitul acestuia. Dacă fișierul nu există, acesta este creat.

Pentru lucrul cu fișierul în mod binar avem variante similare:

“rb”, “wb”, “ab”, “r+b”, “w+b”, “a+b”.

Ce afișează programul?

main.c	untitled_1.txt	untitled_3.c	main.c	untitled_1.txt	untitled_3.c
<pre>1 #include <stdio.h> 2 #include <stdlib.h> 3 // #include <string.h> 4 5 int main() 6 { 7 FILE *g=fopen("untitled_1.txt","r+"); 8 char sir[10]; 9 10 if (g==NULL) 11 printf("Eroare"); 12 13 fgets(sir,10,g); 14 printf("%s",sir); 15 16 return 0; 17 }</pre>			<pre>1 abecedar</pre>		
			<pre>< ▼ ↗ ↘ abecedar ...Program finished with exit code 0 Press ENTER to exit console.</pre>		

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

```
int main()
```

```
{
```

```
FILE *g = fopen ("fisier.txt", "r+");
```

```
char c;
```

```
if (g == NULL)
```

```
    printf("\n Nu s-a putut deschide! \n");
```

```
c = fgetc(g);
```

```
printf("%c", c);
```

```
fputc('z', g);
```

```
fflush(g);
```



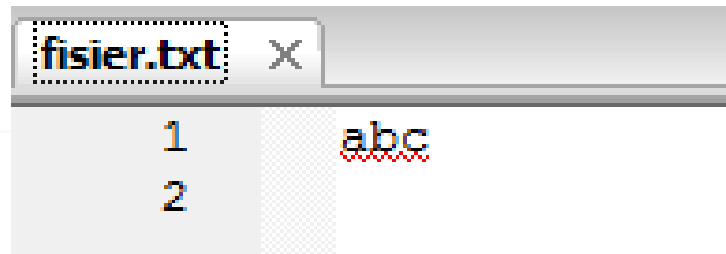
Se forteaza scrierea datelor in fisier fara
inchiderea acestuia cu **fflush(g)**

```
if (fclose(g) != 0)
```

```
    printf("\n Probleme la inchiderea fisierelor!\n");
```

```
return 0;
```

```
}
```



Ce afișează programul?

```

#include <stdlib.h>

int main()
{
    FILE *g = fopen ("fisier.txt", "r+");
    char c;

    if (g == NULL)
        printf("\n Nu s-a putut deschide! \n");

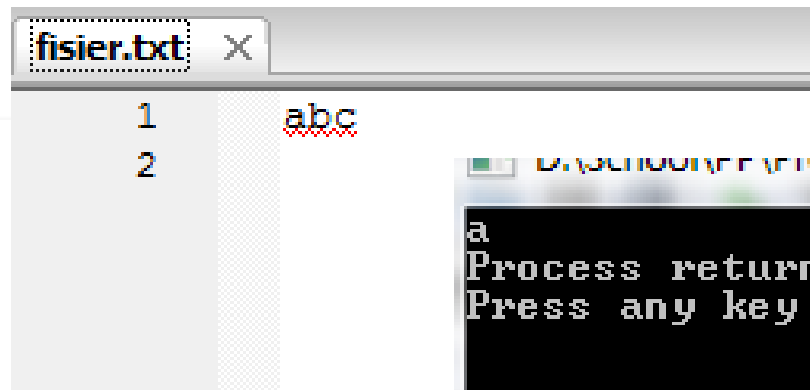
    c = fgetc(g);
    printf("%c", c);

    fputc('z', g);
    fflush(g);

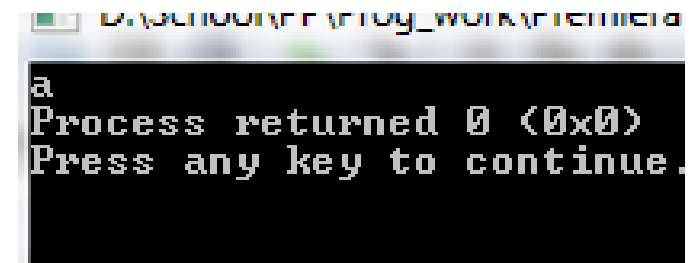
    if (fclose(g) != 0)
        printf("\n Probleme la inchiderea fisierelor!\n");

    return 0;
}

```



Rezultat:



Fișierul rămâne neschimbat?


```

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

int main()
{
    FILE *g = fopen ("fisier.txt", "r+");
    char c;

    if (g == NULL)
        printf("\n Nu s-a putut deschide! \n");

    c = fgetc(g);
    printf("%c", c);
    fclose(g);

    g = fopen ("fisier.txt", "r+");
    fputc('z', g);
    fflush(g);

    if (fclose(g) != 0)
        printf("\n Probleme la inchiderea fisierelor!\n");

    return 0;
}

```

fisier.txt		X
1	zbc	
2		

Rezultat: suprascrierea primului caracter

Lucrul cu fișiere

- ❑ Redeschiderea fișierului implică resetarea cursorului în punctul de început al fișierului!
- ❑ Adăugând + după modifierul de acces (e.g. w+, r+, a+), fișierul se deschide cu permisiuni de citire/scriere.
- ❑ Cu toate acestea:
 - a) După ce s-a citit din fișier, va trebui apelată o funcție de poziționare în fișier (**fseek**, **fsetpos**, **rewind**)
 - b) După scriere, va trebui apelată **fflush()** sau o funcție de poziționare în fișier, înainte de a citi.

=> Ce facem dacă dorim scrierea la poziția curentă a cursorului?

Ce facem dacă dorim scrierea la poziția curentă a cursorului?

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

int main()
{
    FILE *g = fopen ("fisier.txt", "r+");
    char c;

    if (g == NULL)
        printf("\n Nu s-a putut deschide! \n");

    c = fgetc(g);
    printf("%c", c);

    fseek(g, 0, SEEK_CUR); |
    fputc('z', g);
    fflush(g);

    if (fclose(g) != 0)
        printf("\n Probleme la inchiderea fisierelor!\n");

    return 0;
}
```

← **Funcția fseek** poziționează “cursorul” (virtual) de citire/scriere oriunde în interiorul fișierului

Funcția ftell returnează poziția curentă a cursorului în cadrul fișierului.

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      FILE *g = fopen ("fisier.txt", "r+");
7      char c;
8      if (g == NULL)
9          printf("\n Nu s-a putut deschide! \n");
10
11      c = fgetc(g);
12      printf("%c",c);
13
14      fseek(g,0,SEEK_CUR);
15
16      fputc('z',g);
17      if (fclose(g) != 0)
18          printf("\n Probleme la inchiderea fisierului! \n");
19
20      return 0;
21 }
```

Ce afișează programul?

Nu s-a putut deschide!

...Program finished with exit code 0
Press ENTER to exit console.

```
main.c  fisier.txt  ⋮
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      FILE *g = fopen ("fisier.txt", "r+");
7      char c;
8      if (g == NULL)
9          printf("\n Nu s-a putut deschide! \n");
10
11      c = fgetc(g);
12      printf("%c",c);
13
14      fseek(g,0,SEEK_CUR);
15
16      fputc('z',g);
17      if (fclose(g) != 0)
18          printf("\n Probleme la inchiderea fisierului! \n");
19
20      return 0;
21  }
22
```

fisier.txt	
1	abc
2	

fisier.txt	
1	azc
2	

Schimba caracterul de la pozitia curenta!

```
input
a
...Program finished with exit code 0
Press ENTER to exit console.
```

main.c

fisier.txt



```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      FILE *g = fopen ("fisier.txt", "r+");
7      char c;
8      if (g == NULL)
9          printf("\n Nu s-a putut deschide! \n");
10
11     printf("%ld \n",ftell(g));
12
13     c = fgetc(g);
14     printf("%c",c);
15
16     fseek(g,0,SEEK_CUR);
17     fputc('z',g);
18
19     printf("\n%ld",ftell(g));
20
21     if (fclose(g) != 0)
22         printf("\n Probleme la inchiderea fisierului! \n");
23
24     return 0;
25 }
```

input

0
a
2

Lucrul cu fișiere

- ❑ Închiderea unui fișier = închiderea unui flux către acel fișier. Se realizează folosind funcția **fclose**:

- ❑ Sintaxa:

int *fclose(FILE *f)

- ❑ f = pointer la FILE care realizează legătura cu fișierul pe care vrem să-l închid

- ❑ **fclose** întoarce:

- ❑ **0** dacă închiderea s-a efectuat cu succes și
- ❑ **EOF** în caz de eroare.

- ❑ Toate fișierele în care s-a scris trebuie închise.
- ❑ Dacă se realizează doar citire, fișierul nu trebuie închis.
- ❑ tastatura și imprimanta sunt considerate fișiere text. Ele nu trebuie deschise și închise.

Lucrul cu fișiere

❑ exemplu:

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

int main()
{
    FILE *g = fopen ("fisier.txt", "w");
    int a;

    if (g == NULL)
        printf("Nu s-a putut deschide!");

    fputc('a', g);

    a = fclose(g);
    printf("%d", a);

    return 0;
}
```

Ce afișează programul?

main.c	fisier.txt
1	abecedar

main.c	fisier.txt
1	z

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


Lucrul cu fișiere

- ❑ **detectarea sfârșitului de fișier.** Se poate realiza și folosind funcția **feof (find end of file)** :

- ❑ **sintaxa**

int feof(FILE *f)

- ❑ **f** = pointer la FILE corespunzătoare fișierului pe care îl prelucrez

- ❑ funcția **feof** returnează:

- ❑ **0** dacă nu s-a ajuns la sfârșitul fișierului la ultima operație de citire
- ❑ **o valoare nenulă** dacă s-a ajuns la sfârșitul fișierului.

Lucrul cu fișiere

❑ Detectarea sfârșitului de fișier

Ce afișează programul?

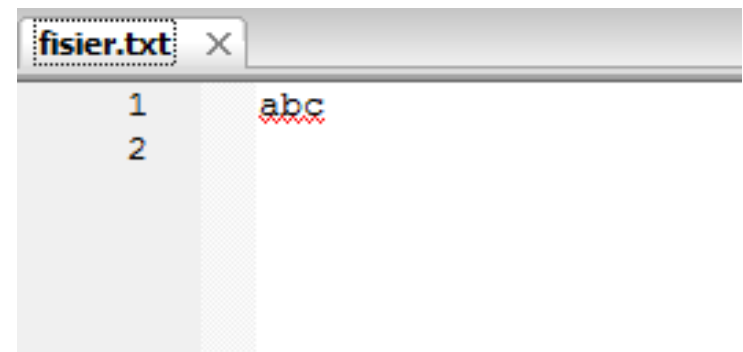
```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    FILE *g = fopen ("fisier.txt", "r+");
    char c;

    if (g == NULL)
        printf("eroare!");

    while (!feof(g))
    {
        c = fgetc(g);
        printf("%c(%d) ", c, c);
    }

    return 0;
}
```



Lucrul cu fișiere

❑ Detectarea sfârșitului de fișier

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

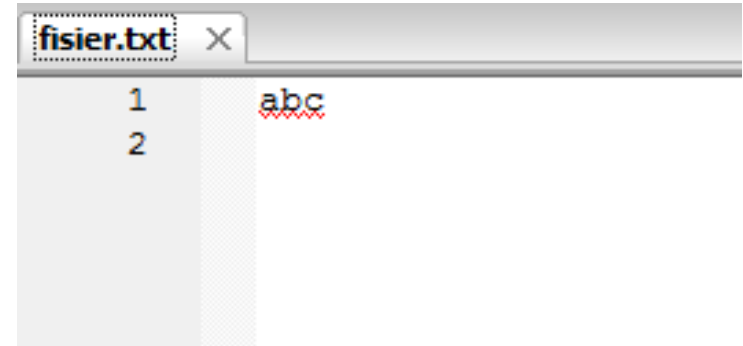
int main()
{
    FILE *g = fopen ("fisier.txt", "r+");
    char c;

    if (g == NULL)
        printf("eroare!");

    while (!feof(g))
    {
        c = fgetc(g);
        printf("%c(%d) ", c, c);

    }

    return 0;
}
```



```
a(97) b(98) c(99)
<10>  <-1>
Process returned 0 (0x0)   execution time : 0.01
Press any key to continue.
-
```

ASCII code of '\n', a non-zero value

Fișiere text. Funcții specifice de manipulare

Cuprins:

- Funcții de citire/ scriere la nivel de caracter
- Funcții de citire/ scriere la nivel de linie
- Funcții de citire/ scriere cu format
- Funcții de poziționare într-un fișier
- Alte funcții

Funcții de citire/scriere la nivel de caracter

- ❑ **int fgetc(FILE *f)** returneaza codul ASCII al caracterului citit din fișierul **f**.
 - ❑ întoarce EOF (= -1) dacă s-a ajuns la finalul fișierului sau a avut loc o eroare la citire
- ❑ **int fputc(int c, FILE *f)** scrie caracterul cu codul ASCII **c** în fișierul **f**.
 - ❑ întoarce:
 - ❑ EOF (= -1) în caz de eroare
 - ❑ codul ASCII al caracterului scris în caz de succes.

Funcții de citire/scriere la nivel de caracter

```
#include <stdlib.h>

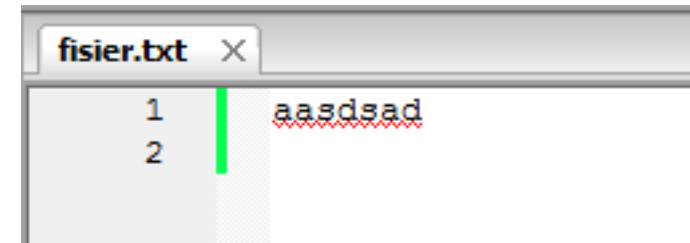
int main()
{
    char nume[30]="fisier_copy.txt";
    FILE *f = fopen ("fisier.txt", "r");
    FILE *g = fopen (nume, "w");
    char c;

    if (g == NULL)
        printf("\n Nu s-a putut deschide! \n");

    while ((c = fgetc(f)) != EOF)
        fputc(c, g);

    if ((fclose(f) != 0) || (fclose(g) != 0))
        printf("\n Probleme la inchiderea fisierelor!\n");

    return 0;
}
```



Funcții de citire/scriere la nivel de caracter

```
#include <stdlib.h>

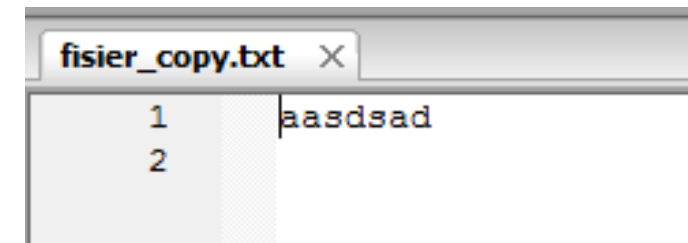
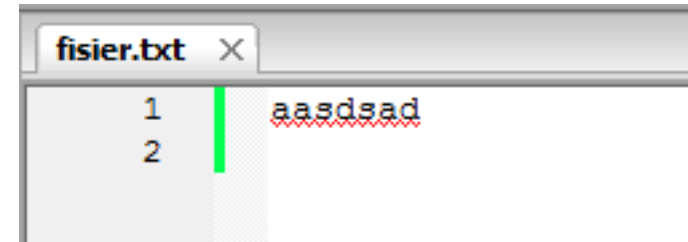
int main()
{
    char nume[30]="fisier_copy.txt";
    FILE *f = fopen ("fisier.txt","r");
    FILE *g = fopen (nume,"w");
    char c;

    if (g == NULL)
        printf("\n Nu s-a putut deschide! \n");

    while ((c = fgetc(f))!=EOF)
        fputc(c,g);

    if ((fclose(f)!=0) || (fclose(g)!=0))
        printf("\n Probleme la inchiderea fisierelor!\n");

    return 0;
}
```



Funcții de citire/scriere la nivel de caracter

Ce afișează programul?

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

```
int main()
```

```
{
```

```
FILE *f = fopen ("fisier.txt", "r");
```

```
char c;
```

```
if (f == NULL)
```

```
    printf("\n Nu s-a putut deschide! \n");
```

```
while (!feof(f))
```

```
    { c = fgetc(f);
```

```
      printf("%c(%d) ", c, c);
```

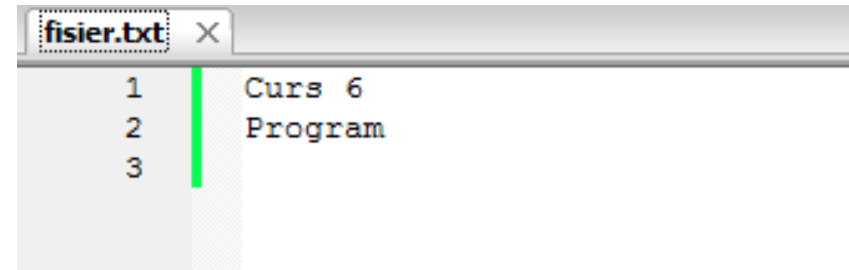
```
    }
```

```
if (fclose(f) != 0)
```

```
    printf("\n Probleme la inchiderea fisierelor!\n");
```

```
return 0;
```

```
}
```



Funcții de citire/scriere la nivel de caracter

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

int main()
{
    FILE *f = fopen ("fisier.txt", "r");
    char c;

    if (f == NULL)
        printf("\n Nu s-a putut deschide! \n");

    while (!feof(f))
    {
        c = fgetc(f);
        printf("%c(%d) ", c, c);

    }

    if (fclose(f) != 0)
    {
        printf("Eroare la inchiderea fișierului!\n");
    }

    return 0;
}
```

fisier.txt		X
1	Curs 6	
2	Program	
3		

```
C(67) u(117) r(114) s(115) (32) 6(54)
(10) P(80) r(114) o(111) g(103) r(114) a(97) m(109)
(10) (-1)
Process returned 0 (0x0)    execution time : 0.031 s
Press any key to continue.
```

Funcții de citire/scriere la nivel de caracter

❑ **char* fgets(char *sir, int m, FILE *f)**

- ❑ citește maxim **m-1** caractere sau până la **'\n'** și pune șirul de caractere în **sir** (adaugă la sfârșit **'\0'**)
- ❑ returnează adresa șirului citit
- ❑ dacă apare vreo eroare întoarce NULL

❑ **int fputs(char *sir, FILE *f)**

- ❑ scrie șirul **sir** în fișierul **f**, fără a pune **'\n'** la sfârșit
- ❑ întoarce:
 - ❑ numărul de caractere scrise, sau
 - ❑ **EOF** în caz de eroare

Funcții de citire/scriere la nivel de linie

main.c	fisier.txt	:
1	<code>#include <stdio.h></code>	
2	<code>#include <stdlib.h></code>	
3		<i>Ce afișează programul?</i>
4	<code>int main()</code>	
5	<code>{</code>	
6		
7	<code>FILE *g = fopen ("fisier.txt", "r+");</code>	
8	<code>char sir[30];</code>	
9		
10		
11	<code>if (g ==NULL)</code>	
12	<code>printf("\nNu s-a putut deschide fisierul!");</code>	
13		
14	<code>fgets(sir,20,g);</code>	
15	<code>printf("%s",sir);</code>	
16		
17	<code>if (fclose(g)!=0)</code>	
18	<code>printf("\nProbleme la inchiderea fisierului");</code>	
19		
20	<code>return 0;</code>	
21	<code>}</code>	

Funcții de citire/scriere la nivel de linie

```
main.c  fisier.txt  ⋮
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6
7      FILE *g = fopen ("fisier.txt", "r+");
8      char sir[30];
9
10
11     if (g ==NULL)
12         printf("\nNu s-a putut deschide fisierul!");
13
14     fgets(sir,20,g);
15     printf("%s",sir);
16
17     if (fclose(g)!=0)
18         printf("\nProbl
19
20     return 0;
21 }
```

```
main.c  fisier.txt  ⋮
1  abecedarABECEDARcalendarCALENDAR
<
input
abecedarABECEDARcal
...Program finished with exit code 0
Press ENTER to exit console.
```

Funcții de citire/scriere cu format

- ❑ **int fscanf(FILE *f, char *format)**

- ❑ citește din fișierul **f** folosind un format (analog cu scanf)

- ❑ **int fprintf(FILE *f, char *format)**

- ❑ scrie în fișierul **f** folosind un format (analog cu printf)

Funcții de poziționare într-un fișier

- ❑ în C ne putem poziționa pe un anumit octet din fișier. Funcțiile care permit poziționarea (cele mai importante) sunt:
- ❑ **long int ftell(FILE *f)**
 - ❑ întoarce numărul octetului curent față de începutul fișierului;
 - ❑ (dimensiunea maximă a unui fișier în C este de $2^{31}-1$ octeți ~ 2GB)
- ❑ **int fseek(FILE *f, int nr_octeti, int origine)**
 - ❑ mută pointerul de fișier **f** pe octetul numărul **nr_octeti** în raport cu **origine**
 - ❑ origine – 3 valori posibile:
 - ❑ **SEEK_SET (= 0)** - început de fișier
 - ❑ **SEEK_CUR (=1)** – poziția curentă
 - ❑ **SEEK_END (=2)** – sfârșit de fișier

Funcții de poziționare într-un fișier

□ Exemplu: aflarea dimensiunii unui fișier

Ce afișează programul?

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

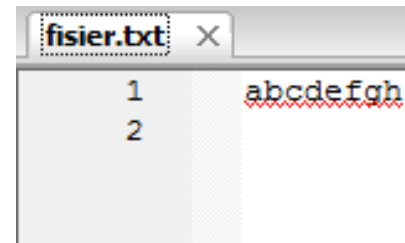
int main()
{
    FILE *g = fopen ("fisier.txt", "r+");
    char nr_linii = 0;
    long int nr_octeti;

    if (g == NULL)
        printf("\n Nu s-a putut deschide! \n");

    fseek(g, 0, SEEK_END);
    nr_octeti = ftell(g);
    printf("%ld", nr_octeti);

    if (fclose(g) != 0)
        printf("\n Probleme la inchiderea fisierelor!\n");

    return 0;
}
```



1	abcdefgh
2	

Funcții de poziționare într-un fișier

□ Exemplu: aflarea dimensiunii unui fișier

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

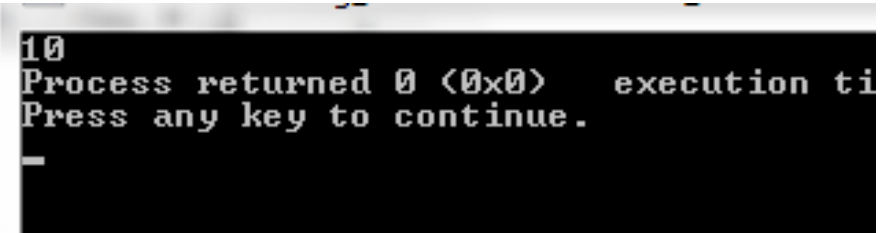
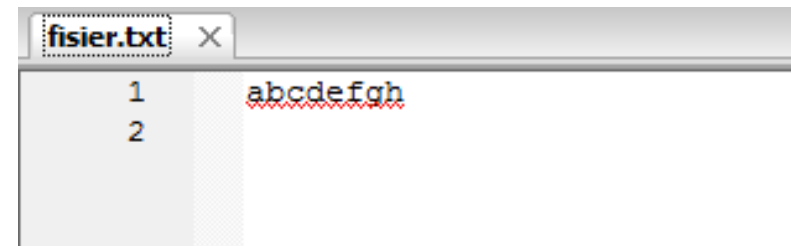
int main()
{
    FILE *g = fopen ("fisier.txt", "r+");
    char nr_linii = 0;
    long int nr_octeti;

    if (g == NULL)
        printf("\n Nu s-a putut deschide! \n");

    fseek(g, 0, SEEK_END);
    nr_octeti = ftell(g);
    printf("%ld", nr_octeti);

    if (fclose(g) != 0)
        printf("\n Probleme la inchiderea fisierelor!\n");

    return 0;
}
```



Alte funcții pentru lucrul cu fișiere

❑ **void rewind (FILE *f)**

- ❑ repoziționarea pointerului asociat fișierului la începutul său.

❑ **int remove(char * nume_fisier);**

- ❑ șterge fișierul cu numele = **nume_fisier**.
- ❑ Întoarce: **0** în caz de succes, **1** în caz de eroare;

❑ **int rename(char *nume_vechi, char *nume_nou);**

- ❑ redenumeste fișierul cu numele = **nume_vechi** cu **nume_nou**.
- ❑ întoarce **0** în caz de succes, **1** în caz de eroare;

Cuprinsul cursului de azi

1. Fișiere: noțiuni generale
2. Fișiere text. Funcții specifice de manipulare.
- 3. Funcții**

Funcții

Cuprins:

- Declaraare și definire
- Valoarea returnată de o funcție
- Prototipul și argumentele unei funcții
- Metode de trasmitere a paramerilor
- Apelul funcției și revenirea din apel

Funcții

- ❑ permit modularizarea programelor
 - ❑ variabilele declarate în interiorul funcțiilor – variabile locale (vizibile doar în interior)
- ❑ parametri funcțiilor
 - ❑ permit comunicarea informației între funcții
 - ❑ sunt variabile locale ale funcțiilor
- ❑ avantajele utilizării funcțiilor
 - ❑ divizarea problemei în subprobleme
 - ❑ managementul dezvoltării programelor
 - ❑ utilizarea/reutilizarea funcțiilor scrise în alte programe
 - ❑ elimină duplicarea codului scris

Funcții

- ❑ o funcție = bloc de instrucțiuni care nu se poate executa de sine stătător ci trebuie apelat

- ❑ sintaxa:

tip_returnat nume_functie (lista parametrilor formali)

antetul funcției
(declarare)

```
{  variabile locale  
  instructiuni;  
  return expresie;  
}
```

corpul funcției
(definire)

- ❑ lista de parametri formali poate fi reprezentata de:
 - ❑ nici un parametru:
 - ❑ **tip_returnat nume_functie ()**
 - ❑ **tip_returnat nume_functie (void)**
 - ❑ unul sau mai mulți parametri separați prin virgulă.

Valoarea returnată de o funcție

- ❑ două categorii de funcții:
 - ❑ care **returnează o valoare**: prin utilizarea instrucțiunii **return expresie**;
 - ❑ care **nu returnează o valoare**: prin instrucțiunea **return**;
(tipul returnat este **void**)
- ❑ returnarea valorii
 - ❑ poate returna:
 - ❑ orice tip standard (**void, char, int, float, double**) sau
 - ❑ definit de utilizator (**structuri, uniuni, enumerari, typedef**)
 - ❑ declarațiile și instrucțiunile din funcții sunt executate până se întâlnește
 - ❑ instrucțiunea **return**
 - ❑ acolada închisă **}** - execuția atinge finalul funcției

Valoarea returnată de o funcție

```
double f(double t)
{
    return t-1.5;
}
```

← definire de funcție

```
float g(int);
```

← declarație de funcție (prototip)

```
int main()
{
    float a=11.5;
    printf("%f\n", f(a));
    printf("%f\n", g(a));
}
```

```
float g(int z)
{
    return z+2.0;
}
```

← definire de funcție

Valoarea returnată de o funcție

Ce afișează programul?

```
double f(double t)
{
    return t-1.5;
}
```

← definire de funcție

```
float g(int);
```

← declarație de funcție

```
int main()
{
    float a=11.5;
    printf("%f\n", f(a));
    printf("%f\n", g(a));
}
```

Rezultat afișat

10.000000

13.000000

```
float g(int z)
{
    return z+2.0;
}
```

← definire de funcție

Prototipul și argumentele funcțiilor

- ❑ **prototipul** unei funcții (declararea ei) constă în specificarea antetului urmat de caracterul ;
 - ❑ nu este necesară specificarea numelor parametrilor formali
int adunare(int, int);
 - ❑ este necesară inserarea prototipului unei funcții înaintea altor funcții în care este invocată dacă definirea ei este localizată după definirea acelor funcții
- ❑ **parametri** apar în definiții
- ❑ **argumentele** apar în apelurile de funcții
 - ❑ corespondența între parametrii formali (definiția funcției) și actuali (apelul funcției) este **pozițională**
 - ❑ regula de **conversie a argumentelor**: în cazul în care diferă, tipul fiecărui argument este convertit automat la tipul parametrului formal corespunzător (ca și în cazul unei simple atribuirii)

Prototipul și argumentele funcțiilor

```
1  #include <stdio.h>
2
3  void f(char a)
4  {
5      printf("%d\n", a);
6  }
7
8  int main()
9  {
10     int a = 300;
11     float b = 305.7;
12     f(a);
13     f(b);
14     return 0;
15 }
```

Rezultatul afișat

44

49

Fișiere header cu extensia .h

- ❑ conțin **prototipuri de funcții**

- ❑ **bibliotecile standard**

- ❑ conțin prototipuri de funcții standard regăsite în fișierele *header* corespunzătoare (ex. **stdio.h**, **stdlib.h**, **math.h**, **string.h**)

- ❑ *Exemplu:* biblioteca **stdio.h** care conține și prototipul funcției **printf**:

```
int printf(const char* format, ...);
```

- ❑ se încarcă cu **#include <filename.h>**

- ❑ **biblioteci utilizator**

- ❑ conțin prototipuri de funcții și macrouri

- ❑ se pot salva ca fișiere cu extensia *.h* : ex. **filename.h**

- ❑ se încarcă cu **#include "filename.h"**

Transmiterea parametrilor către funcții

- ❑ utilizată la apelul funcțiilor
- ❑ în limbajul C transmiterea parametrilor se poate face doar prin **valoare (pass-by-value)**
 - ❑ o copie a argumentelor este trimisă funcției
 - ❑ modificările în interiorul funcției nu afectează argumentele originale
- ❑ în limbajul C++ transmiterea parametrilor apelul se poate face și prin **referință (pass-by-reference)**
 - ❑ argumentele originale sunt trimise funcției
 - ❑ modificările în interiorul funcției afectează argumentele trimise

```
1  #include <stdio.h>
2
3  void interschimba1(int x, int y)
4  {
5      int aux = x; x = y; y = aux;
6  }
7
8  void interschimba2(int& x, int& y)
9  {
10     int aux = x; x = y; y = aux;
11 }
12
13 void interschimba3(int* x, int* y)
14 {
15     int aux = *x; *x = *y; *y = aux;
16 }
17
18 int main()
19 {
20     int x=10, y =15;
21     interschimba1(x,y);
22     printf("x = %d, y = %d \n",x,y);
23     x=10, y =15;
24     interschimba2(x,y);
25     printf("x = %d, y = %d \n",x,y);
26     x=10, y =15;
27     interschimba3(&x,&y);
28     printf("x = %d, y = %d \n",x,y);
29     return 0;
30 }
```

```
1  #include <stdio.h>
2
3  void interschimba1(int x, int y)
4  {
5      int aux = x; x = y; y = aux;
6  }
7
8  void interschimba2(int& x, int& y)
9  {
10     int aux = x; x = y; y = aux;
11 }
12
13 void interschimba3(int* x, int* y)
14 {
15     int aux = *x; *x = *y; *y = aux;
16 }
17
18 int main()
19 {
20     int x=10, y =15;
21     interschimba1(x,y);
22     printf("x = %d, y = %d \n",x,y);
23     x=10, y =15;
24     interschimba2(x,y);
25     printf("x = %d, y = %d \n",x,y);
26     x=10, y =15;
27     interschimba3(&x,&y);
28     printf("x = %d, y = %d \n",x,y);
29     return 0;
30 }
```

apel prin valoare

apel prin referință
numai în C++

```
1  #include <stdio.h>
2
3  void interschimba1(int x, int y)
4  {
5      int aux = x; x = y; y = aux;
6  }
7
8  void interschimba2(int& x, int& y)
9  {
10     int aux = x; x = y; y = aux;
11 }
12
13 void interschimba3(int* x, int* y)
14 {
15     int aux = *x; *x = *y; *y = aux;
16 }
17
18 int main()
19 {
20     int x=10, y =15;
21     interschimba1(x,y);
22     printf("x = %d, y = %d \n",x,y);
23     x=10, y =15;
24     interschimba2(x,y);
25     printf("x = %d, y = %d \n",x,y);
26     x=10, y =15;
27     interschimba3(&x,&y);
28     printf("x = %d, y = %d \n",x,y);
29     return 0;
30 }
```

x = 10, y = 15
x = 15, y = 10
x = 15, y = 10

apel prin valoare

apel prin referință
numai în C++

apel prin valoare

Transmiterea parametrilor către funcții

- ❑ utilizată la apelul funcțiilor
- ❑ în limbajul C transmiterea parametrilor se poate face doar prin **valoare (pass-by-value)**
 - ❑ o copie a argumentelor este trimisă funcției
 - ❑ modificările în interiorul funcției nu afectează argumentele originale
- ❑ pentru modificarea parametrilor actuali, funcției i se transmit nu valorile parametrilor actuali, ci **adresele lor (pass by pointer)**. Funcția face o copie a adresei dar prin intermediul ei lucrează cu variabila “reală” (zona de memorie “reală”). Astfel **putem simula în C transmiterea prin referință cu ajutorul pointerilor.**

Transmiterea parametrilor către funcții

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f1(int a, int b)
5  {
6      a++;
7      b++;
8      printf("In functia f1 avem a=%d,b=%d\n",a,b);
9      return a+b;
10 }
11
12 int main() {
13     int a = 5, b = 8;
14     int c = f1(a,b);
15     printf("In functia main avem a=%d,b=%d,c=%d\n",a,b,c);
16     return 0;
17 }
18
```

Ce afișează programul?

Transmiterea parametrilor către funcții

```
main.c [X]
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f1(int a, int b)
5  {
6      a++;
7      b++;
8      printf("In functia f1 avem a=%d,b=%d\n",a,b);
9      return a+b;
10 }
11
12 int main() {
13     int a = 5, b = 8;
14     int c = f1(a,b);
15     printf("In functia main avem a=%d,b=%d,c=%d\n",a,b,c);
16     return 0;
17 }
18
```

In functia f1 avem a=6,b=9
In functia main avem a=5,b=8,c=15

Process returned 0 (0x0) execution time : 0.004 s
Press ENTER to continue.

Transmiterea parametrilor către funcții

```
main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f1(int a, int b)
5  {
6      a++;
7      b++;
8      printf("In functia f1 avem a=%d,b=%d\n",a,b);
9      return a+b;
10 }
11
12 int f2(int*a, int b)
13 {
14     *a = *a + 1;
15     b++;
16     printf("In functia f2 avem *a=%d,b=%d\n",*a,b);
17     return *a+b;
18 }
19
20 int main(){
21     int a = 5, b= 8;
22     int c = f2(&a,b);
23     printf("In functia main avem a=%d,b=%d,c=%d\n",a,b,c);
24     return 0;
25 }
```

Ce afișează programul?

Transmiterea parametrilor către funcții

```
main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f1(int a, int b)
5  {
6      a++;
7      b++;
8      printf("In functia f1 avem a=%d,b=%d\n",a,b);
9      return a+b;
10 }
11
12 int f2(int*a, int b)
13 {
14     *a = *a + 1;
15     b++;
16     printf("In functia f2 avem *a=%d,b=%d\n",*a,b);
17     return *a+b;
18 }
19                                     In functia f2 avem *a=6,b=9
20 int main(){
21     int a = 5, b= 8;
22     int c = f2(&a,b);
23     printf("In functia main avem a=%d,b=%d,c=%d\n",a,b,c);
24     return 0;
25 }
26                                     In functia main avem a=6,b=8,c=15
```

Apelul funcției și revenirea din apel

- ❑ etapele principale ale apelului unei funcții și a revenirii din acesta în funcția de unde a fost apelată:
 - ❑ argumentele apelului sunt evaluate și trimise funcției
 - ❑ adresa de revenire este salvată pe stivă
 - ❑ controlul trece la funcția care este apelată
 - ❑ funcția apelată alocă pe **stivă** spațiu pentru variabilele locale și pentru cele temporare
 - ❑ se execută instrucțiunile din corpul funcției
 - ❑ dacă există valoare returnată, aceasta este pusă într-un loc sigur
 - ❑ spațiul alocat pe stivă este eliberat
 - ❑ utilizând adresa de revenire controlul este transferat în funcția care a inițiat apelul, după acesta

English Resources

- <https://www.programiz.com/c-programming/c-file-input-output>
- <https://www.geeksforgeeks.org/basics-file-handling-c/>
- <https://ccsuniversity.ac.in/bridge-library/pdf/btech-cs/Functions%2025,26,27,28,-converted.pdf> pages 1-17
- https://users.cs.utah.edu/~germain/PPS/Topics/C_Language/c_functions.html
- <https://www.geeksforgeeks.org/c-functions/>
- <https://www.alphacodingskills.com/c/c-stdio-h.php>

Cursul 5

1. Funcții de citire / scriere
2. Fișiere text. Funcții specifice de manipulare
3. Funcții

Cursul 6

1. Pointeri la funcții
2. Legătura dintre tablouri și pointeri
3. Aritmetica pointerilor