



PROGRAMARE PROCEDURALĂ

Bogdan Alexe

bogdan.alexe@fmi.unibuc.ro

Secția Informatică, anul I,
2018-2019
Cursul 10

Project

- mici erori în proiect în enunțul initial: formula pentru calculul deviației standard e incorrect scrisă (lipsește puterea 2)

- σ_S reprezintă deviația standard a valorilor intensităților grayscale a pixelilor în şablonul S : $\sigma_S = \sqrt{\frac{1}{n-1} \sum_{(i,j) \in S} (S(i,j) - \bar{S})^2}$
- $f_I(i,j)$ reprezintă valoarea intensității grayscale a pixelului de la linia i și coloana j în fereastra f_I .
- \bar{f}_I reprezintă media valorilor intensităților grayscale a pixelilor din fereastra f_I (media celor 165 de pixeli din fereastra f_I);
- σ_{f_I} reprezintă deviația standard a valorilor intensităților grayscale a pixelilor în fereastra f_I : $\sigma_{f_I} = \sqrt{\frac{1}{n-1} \sum_{(i,j) \in f_I} (f_I(i,j) - \bar{f}_I)^2}$

- formula corectă:

- σ_S reprezintă deviația standard a valorilor intensităților grayscale a pixelilor în şablonul S : $\sigma_S = \sqrt{\frac{1}{n-1} \sum_{(i,j) \in S} (S(i,j) - \bar{S})^2}$
- $f_I(i,j)$ reprezintă valoarea intensității grayscale a pixelului de la linia i și coloana j în fereastra f_I .
- \bar{f}_I reprezintă media valorilor intensităților grayscale a pixelilor din fereastra f_I (media celor 165 de pixeli din fereastra f_I);
- σ_{f_I} reprezintă deviația standard a valorilor intensităților grayscale a pixelilor în fereastra f_I : $\sigma_{f_I} = \sqrt{\frac{1}{n-1} \sum_{(i,j) \in f_I} (f_I(i,j) - \bar{f}_I)^2}$

- fișierul sursa grayscale.c nu mergea bine pe windows, am adăugat fflush după fiecare fwrite
- materiale actualizate pe moodle

Test de laborator

- sămbătă, 15 decembrie, seria 13 între 9-11, seria 14 între 12-14;
- fără materiale, fără net, fără laptopuri;
- subiect unic;
- afișăm repartizarea pe laboratoare vineri seara pe moodle;
- dacă nu puteți da testul dintr-un motiv bun mă anunțați până miercurea viitoare. Cei care nu participă la test iau 0 puncte;
- materie până la alocare dinamică (inclusiv).

Examen final

- schimbare în planul de învățământ;
- se lasă la aprecierea cadrelor didactice dacă metoda de evaluare este verificare sau examen;
- pentru verificare dăm examen scris în săptămâna 14 (weekend-ul 19-20 ianuarie);
- pentru examen putem da oricând în sesiune, 21 ianuarie – 10 februarie.

Recapitulare – cursul trecut

1. Prezentarea proiectului
2. Alocarea dinamică a memoriei
3. Clase de memorare

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: tablouri, siruri de caractere, structuri, uniuni, câmpuri de biți, enumerări, pointeri
 - Instrucțiuni de control
 - Directive de preprocessare. Macrodefiniții.
 - Funcții de citire/scriere.
 - Etapele realizării unui program C.
- Fișiere text**
 - Funcții specifice de manipulare.
- Fișiere binare**
 - Funcții specifice de manipulare.
- Funcții (1)**
 - Declarare și definire. Apel. Metode de transmitere a parametrilor. Pointeri la funcții.
- Tablouri și pointeri**
 - Aritmetică pointerilor
 - Legătura dintre tablouri și pointeri
 - Alocarea dinamică a memoriei
 - Clase de memorare
- Siruri de caractere**
 - 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.
 - Programare generică.
- Recursivitate**

Cuprinsul cursului de azi

1. Clase de memorare
2. Siruri de caractere – funcții specifice de manipulare

Clase de alocare/memorare

- Într-un program C felul în care declarăm variabilele definește modul de alocare al acestora. Clasa de alocare a unei variabile definește următoarele caracteristici:
 - locul în memorie unde se rezervă spațiu pentru variabilă;
 - durata de viață;
 - vizibilitatea;
 - modalitatea de inițializare.
- clase de alocare:
 - **auto(matic)**
 - **register**
 - **static (intern)**
 - **static extern**

Clasa de alocare static extern

- ❑ se specifică prin cuvântul cheie **extern**;
- ❑ o variabilă de tip extern este o variabilă definită într-un alt fișier sursă (extern);
- ❑ se alocă în funcție de modul de declarare din fișierul sursă.

Exemplu:

//fisier1.c

extern int i; //declara variabila i ca fiind definită in alt fisier

//fisier2.c

int i = 5; //variabila i este definită aici

O variabilă poate fi declarată în mai multe fișiere (cu extern), dar trebuie definită într-un singur fișier!

Clasa de alocare static extern

```
exempluExtern1.c
```

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

```
exempluExtern2.c
```

```
1 #include<stdio.h>
2
3 int x = 1;
4
5 void f()
6 {
7     int x = 7;
8     {
9         extern int x;
10         printf("%d \n",x);
11     }
12 }
13
14 int main()
15 {
16     f();
17     return 0;
18 }
```

Afiseaza 7. Vreau sa am acces la variabila globala x = 1. Cum fac?

Cuprinsul cursului de azi

1. Clase de memorare
2. Siruri de caractere – funcții specifice de manipulare

Şiruri de caractere

- **un sir de caractere (string)** este un tablou unidimensional cu elemente de tip char terminat cu caracterul '\0' (NUL)
- o zonă de memorie ocupată cu caractere (un caracter ocupă un octet) terminată cu un octet de valoare 0 (caracterul '\0' are codul ASCII egal cu 0).
- o variabilă care reprezintă un sir de caractere este un pointer la primul octet. Se poate reprezenta ca:
 - tablou de caractere (pointer constant):
 - `char sir1[10]; //se aloca 10 octeti`
 - `char sir2[10] = "sir2"; //se aloca 10 octeti`
 - `char sir3[] = "sir3"; //se aloca 5 octeti (se mai adauga '\0')`
 - pointer la caractere:
 - `char *sir4; //se aloca memorie numai pentru pointer`
 - `char *sir5 = "sir5"; //se aloca 8 octeti pentru pointer`

Siruri de caractere

```
exempluSiruri.c x

1 #include<stdio.h>
2 #include<string.h>
3
4 void afiseazaSir(char *t)
5 {
6     printf("Sirul %s incepe la adresa %p si are dimensiunea in memorie %lu \n",t,t,sizeof(t));
7 }
8
9 int main()
10 {
11
12     char sir1[10] = "sir1";
13     char sir2[10];
14     char sir3[] = "sir3";
15     char *sir4 = "sir4";
16
17     printf("Sirul %s incepe la adresa %p si are dimensiunea in memorie %lu \n",sir1,sir1,sizeof(sir1));
18     afiseazaSir(sir1);
19
20     printf("Sirul %s incepe la adresa %p si are dimensiunea in memorie %lu \n",sir2,sir2,sizeof(sir2));
21     printf("Sirul %s incepe la adresa %p si are dimensiunea in memorie %lu \n",sir3,sir3,sizeof(sir3));
22     printf("Sirul %s incepe la adresa %p si are dimensiunea in memorie %lu \n",sir4,sir4,sizeof(sir4));
23
24     return 0;
25 }
```

Citirea și afișarea sirurilor de caractere

□ citire:

- funcția `scanf` cu modelatorul de format `%s`:
 - atenție: dacă inputul este un sir de caractere cu spațiu citește până la spațiu
- funcția `fgets` (în loc de `gets`)
 - `char *fgets(char *s, int size, FILE *stream)`
 - `fgets(buffer, sizeof(buffer), stdin);`
 - citește și spațiile

□ afișare:

- funcția `printf` cu modelatorul de format `%s`;
- funcția `puts` (trece pe linia următoare).

Citirea și afișarea sirurilor de caractere

□ exemplu

```
001 // exempluSiruri2.c
002
003 #include<stdio.h>
004 #include<string.h>
005
006 int main()
007 {
008     char sir1[] = {'r','a','t','o','n','\0'};
009     char sir2[] = "raton";
010     printf("%s %s \n", sir1, sir2);
011
012     char *sir3 = sir1;
013     sir3[0] = 'b';
014     printf("%s %s %s\n", sir1, sir2, sir3);
015
016     char sir4[10] = "raton";
017     printf("%s \n", sir4);
018
019     for(int i = 0; i < 10; i++)
020         printf("Caracterul %c = codul ASCII %d \n", sir4[i], sir4[i]);
021     sir4[5] = 'i';
022     printf("%s \n", sir4);
023
024     char sir5[10] = "raton";
025     sir5[4] = 0;
026     printf("%s \n", sir5);
027     sir5[3] = '\0';
028     printf("%s \n", sir5);
029
030 }
```

Citirea și afișarea sirurilor de caractere

exemplu

```
exempluSiruri2.c      x

1 #include<stdio.h>
2 #include<string.h>
3
4
5 int main()
6 {
7     char sir1[] = {'r','a','t','o','n','\0'};
8     char sir2[] = "raton";
9     printf("%s %s \n", sir1, sir2);
10
11    char *sir3 = sir1;
12    sir3[0] = 'b';
13    printf("%s %s %s\n", sir1, sir2, sir3);
14
15    char sir4[10] = "raton";
16    printf("%s \n", sir4);
17
18    for(int i = 0; i < 10; i++)
19        printf("Caracterul %c = codul ASCII %d \n", sir4[i], sir4[i]);
20    sir4[5] = 'i';
21    printf("%s \n", sir4);
22
23    char sir5[10] = "raton";
24    sir5[4] = 0;
25    printf("%s \n", sir5);
26    sir5[3] = '\0';
27    printf("%s \n", sir5);
28
29    return 0;
30 }
```

Bogdan-Alexes-MacBook-Pro:curs9 bogdan\$ gcc exempluSiruri2.c
Bogdan-Alexes-MacBook-Pro:curs9 bogdan\$./a.out
raton raton
baton raton baton
raton
Caracterul r = codul ASCII 114
Caracterul a = codul ASCII 97
Caracterul t = codul ASCII 116
Caracterul o = codul ASCII 111
Caracterul n = codul ASCII 110
Caracterul = codul ASCII 0
ratoni
rato
rat

Functii predefinite pentru manipularea sirurilor de caractere

- funcții de procesare a sirurilor de caractere specifice incluse în fișierul string.h
- lungimea unui sir – funcția **strlen**
 - antet: int **strlen(const char *sir)**

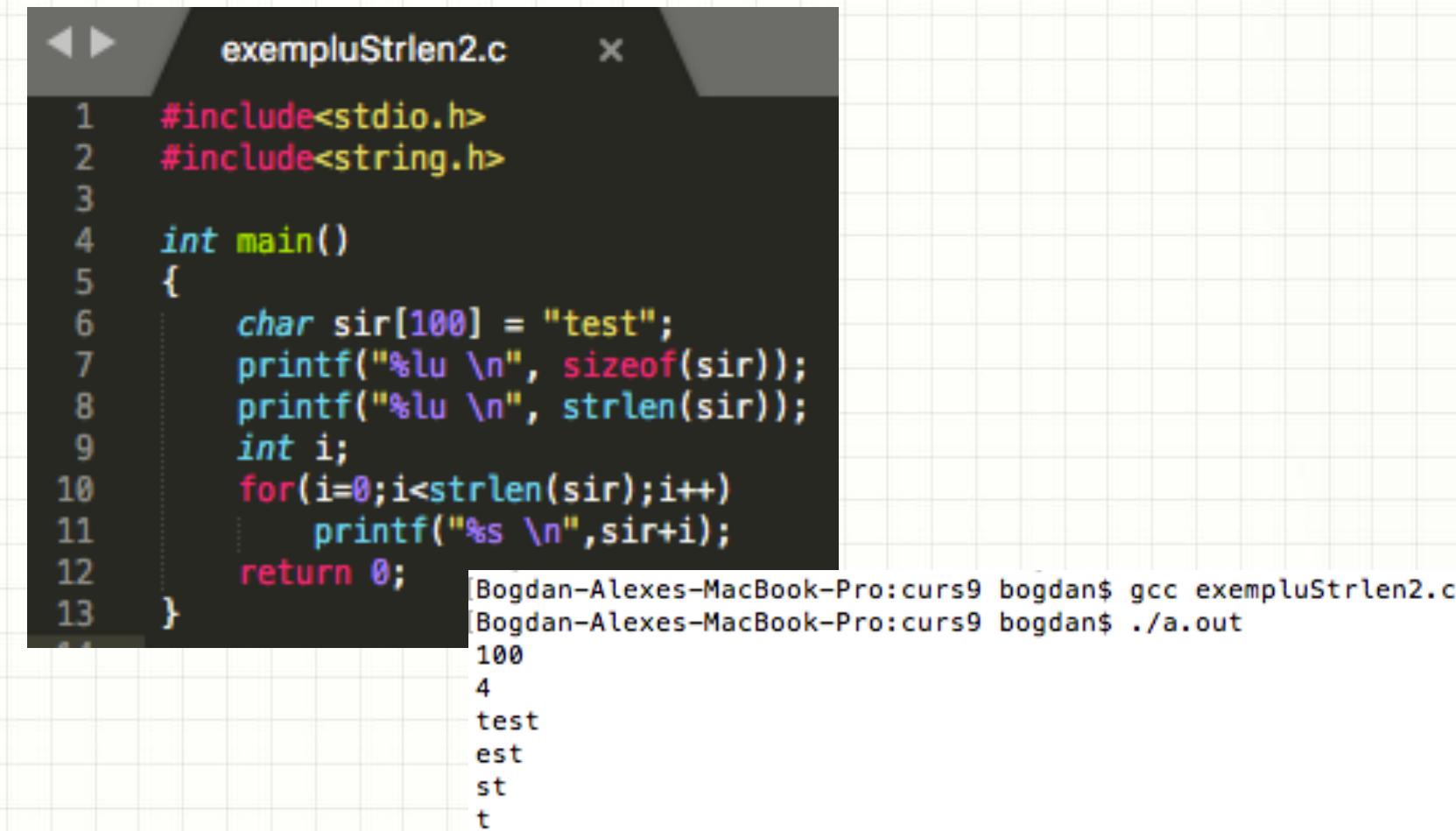
```
exempluStrlen1.c      x

1 #include<stdio.h>
2 #include<string.h>
3
4 int main()
5 {
6     char sir[100] = "test";
7     printf("%lu \n", sizeof(sir));
8     printf("%lu \n", strlen(sir));
9     return 0;
10 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ gcc exempluStrlen1.c
[Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ ./a.out
100
4
```

Funcții predefinite pentru manipularea sirurilor de caractere

- lungimea unui sir – funcția **strlen**
 - antet: **int strlen(const char *sir)**



```
exempluStrlen2.c      x
1 #include<stdio.h>
2 #include<string.h>
3
4 int main()
5 {
6     char sir[100] = "test";
7     printf("%lu \n", sizeof(sir));
8     printf("%lu \n", strlen(sir));
9     int i;
10    for(i=0;i<strlen(sir);i++)
11        printf("%s \n",sir+i);
12    return 0;
13 }
```

Bogdan-Alexes-MacBook-Pro:curs9 bogdan\$ gcc exempluStrlen2.c
Bogdan-Alexes-MacBook-Pro:curs9 bogdan\$./a.out
100
4
test
est
st
t

Funcții predefinite pentru manipularea sirurilor de caractere

- copierea unui sir
 - nu se poate copia conținutul unui sir în alt sir folosind operația de atribuire (se copiază pointerul și nu conținutul).

```
exempluSiruri3.c
```

```
1 #include<stdio.h>
2 #include<string.h>
3
4
5 int main()
6 {
7     char sir6[10] = "raton";
8     char *sir7 = "baton";//nu am voie sa fac sir7[0] = 'r';
9
10    printf("Adresa lui sir6 este %p \n", sir6);
11    printf("Adresa lui sir7 este %p \n", sir7);
12
13    sir7 = sir6;
14    puts(sir7);
15
16    printf("Adresa lui sir6 este %p \n", sir6);
17    printf("Adresa lui sir7 este %p \n", sir7);
18
19
20    return 0;
21 }
```

Functii predefinite pentru manipularea sirurilor de caractere

- copierea unui sir
 - nu se poate copia continutul unui sir în alt sir folosind operația de atribuire (se copiază pointerul și nu continutul).

```
exempluSiruri3.c
```

```
00 1 #include<stdio.h>
01 2 #include<string.h>
02 
03 
04 5 int main()
05 {
06     char sir6[10] = "raton";
07     char *sir7 = "baton";//nu am voie sa fac sir7[0] = 'r';
08 
09     printf("Adresa lui sir6 este %p \n", sir6);
10    printf("Adresa lui sir7 este %p \n", sir7);
11 
12    sir7 = sir6;
13    puts(sir7);
14 
15    printf("Adresa lui sir6 este %p \n", sir6);
16    printf("Adresa lui sir7 este %p \n", sir7);
17    return 0;
18 }
```

```
Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ gcc exempluSiruri3.c
Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ ./a.out
Adresa lui sir6 este 0x7fff52096b3e
Adresa lui sir7 este 0x10db69f74
raton
Adresa lui sir6 este 0x7fff52096b3e
Adresa lui sir7 este 0x7fff52096b3e
```

Funcții predefinite pentru manipularea sirurilor de caractere

- copierea unui sir
 - nu se poate copia conținutul unui sir în alt sir folosind operația de atribuire (se copiază pointerul și nu conținutul).

```
exempluSiruri4.c
1 #include<stdio.h>
2 #include<string.h>
3
4
5 int main()
6 {
7     char sir6[10] = "raton";
8     char sir7[10] = "baton";//acum am voie sa fac sir7[0] = 'r';
9
10    printf("Adresa lui sir6 este %p \n", sir6);
11    printf("Adresa lui sir7 este %p \n", sir7);
12
13    sir7 = sir6;
14    puts(sir7);
15
16    printf("Adresa lui sir6 este %p \n", sir6);
17    printf("Adresa lui sir7 este %p \n", sir7);
18
19    return 0;
20 }
```

Functii predefinite pentru manipularea sirurilor de caractere

- copierea unui sir
 - nu se poate copia continutul unui sir în alt sir folosind operația de atribuire (se copiază pointerul și nu continutul).

The screenshot shows a terminal window with the following content:

```
exempluSiruri4.c
1 #include<stdio.h>
2 #include<string.h>
3
4
5 int main()
6 {
7     char sir6[10] = "raton";
8     char sir7[10] = "baton";//acum am voie sa fac sir7[0] = 'r';
9
10    printf("Adresa lui sir6 este %p \n", sir6);
11    printf("Adresa lui sir7 este %p \n", sir7);
12
13    sir7 = sir6;
14    puts(sir7);
15
16    printf("Adresa lui sir6 este %p \n", sir6);
17    printf("Adresa lui sir7 este %p \n", sir7);
18
19    return 0;
20 }
```

[Bogdan-Alexes-MacBook-Pro:curs9 bogdan\$ gcc exempluSiruri4.c
exempluSiruri4.c:15:10: error: array type 'char [10]' is not assignable
 sir7 = sir6;
 ^
1 error generated.

sir7 este numele unui tablou (pointer constant). Instructiunea sir7=sir6 da eroare la compilare.

Funcții predefinite pentru manipularea sirurilor de caractere

- copierea unui sir – folosim funcțiile **strcpy** și **strncpy**
 - antet: **char* strcpy(char *d, const char* s);**
 - copiază sirul sursă **s** în sirul destinație **d**;
 - returnează adresa sirului destinație;
 - sirul rezultat are un '\0' la final;
 - antet: **char* strncpy(char *d, const char* s, int n);**
 - copiază primele **n** caractere din sirul sursă **s** în sirul destinație **d**;
 - returnează adresa sirului destinație;
 - sirul rezultatul **NU** are un '\0' la final;

Funcții predefinite pentru manipularea sirurilor de caractere

- copierea unui sir – folosim funcțiile **strcpy** și **strncpy**



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

int main()
{
    char s[10] = "exemplu";
    char t[10] = "test";
    strncpy(s,t,3);
    printf("%s \n",s);

    s[4] = 0;
    printf("%s \n",s);

    char p[100] = "nimic";
    strcpy(p,s);
    p[3] = '\0';
    printf("%s \n",p);

    return 0;
}
```

```
[Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ gcc exempluStrncpy.c
[Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ ./a.out
tesmplu
tesm
tes
```

Functii predefinite pentru manipularea sirurilor de caractere

- copierea unui sir – folosim funcțiile **strcpy** și **strncpy**
 - antet: **char* strcpy(char *d, const char* s);**
 - presupune că sirurile destinație și sursa nu se suprapun
 - dacă cele două siruri se suprapun funcția prezintă **undefined behaviour** (comportament nedefinit)

```
exempluStrncpy1.c

1 #include<stdio.h>
2 //#include<string.h>
3
4 int main()
5 {
6     char s[10] = "exemplu";
7     char t[10] = "test";
8
9     strcpy(t,t+1);
10    printf("%s \n",t);
11
12    strcpy(s+1,s);
13    printf("%s \n",s);
14
15    return 0;
16 }
```

est
eeeeeeeeeeeeeeee
P. 1 / 11 M. P. 1 / 1
...
est
e exemplu
Bogdan Alexeiu MacBook Pro (M1, 2020) bogdan

Funcții predefinite pentru manipularea sirurilor de caractere

- compararea sirurilor – funcțiile **strcmp** și **strncmp**
 - antet: `int strcmp(const char *s1, const char* s2);`
 - compară lexicografic sirurile s1 și s2;
 - returnează <0 dacă $s1 <_L s2$, 0 dacă $s1 =_L s2$ și >0 dacă $s1 >_L s2$;
 - antet: `int strncmp(const char *s1, const char* s2, int n);`
 - compară lexicografic sirurile s1 și s2 trunchiate la lungimea n
- ambele funcții sunt case sensitive
 - `strcmp("POPA","Popa")` returneaza un numar < 0 întrucât 'O' $<$ 'o' (codurile ASCII 79 respectiv 111)
 - unele implementări au funcția **stricmp** – case insensitive

Functii predefinite pentru manipularea sirurilor de caractere

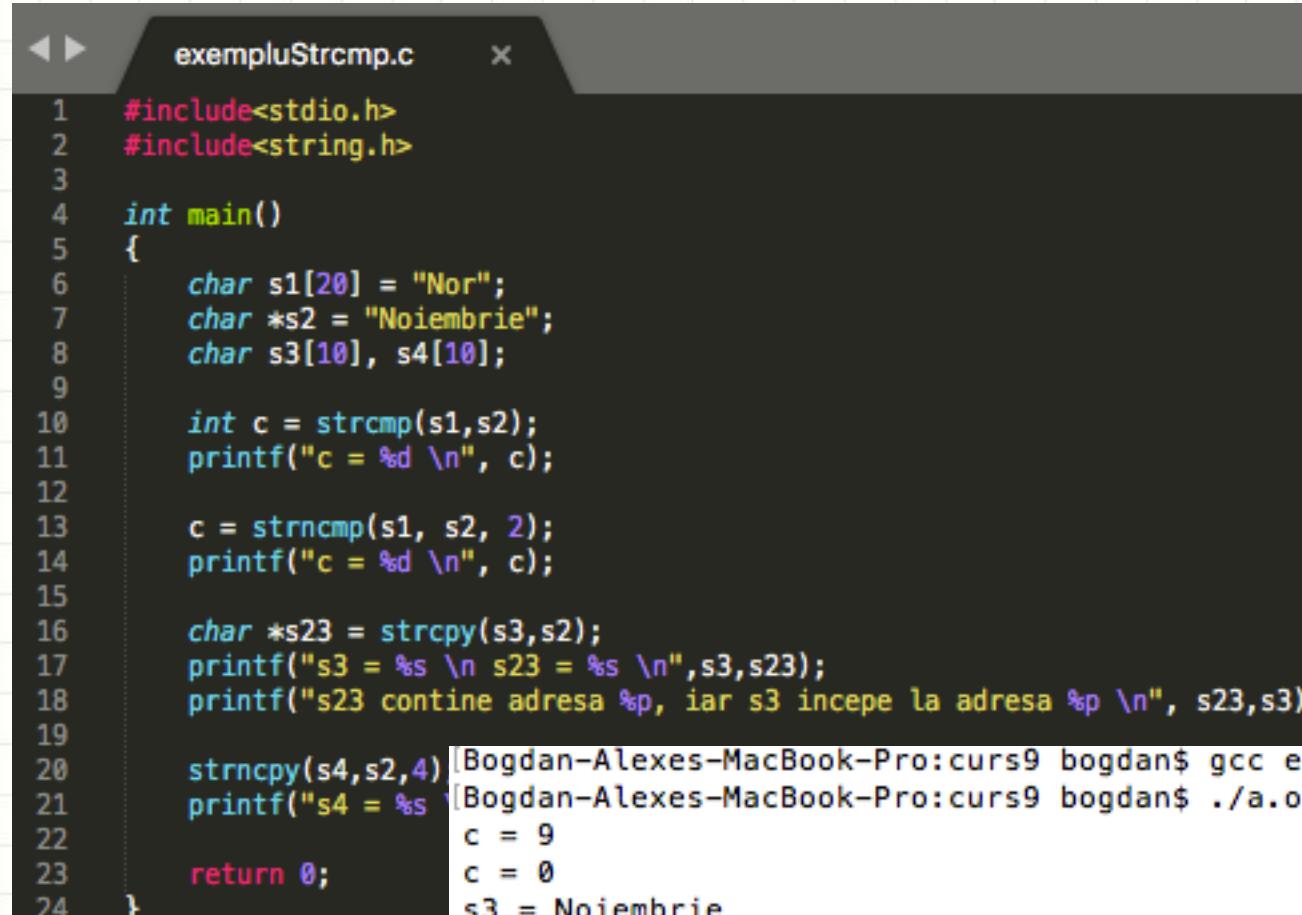
- compararea sirurilor – functiile **strcmp** si **strncpy**

```
exemplustrcmp.c      x

1 #include<stdio.h>
2 #include<string.h>
3
4 int main()
5 {
6     char s1[20] = "Nor";
7     char *s2 = "Noiembrie";
8     char s3[10], s4[10];
9
10    int c = strcmp(s1,s2);
11    printf("c = %d \n", c);
12
13    c = strncmp(s1, s2, 2);
14    printf("c = %d \n", c);
15
16    char *s23 = strcpy(s3,s2);
17    printf("s3 = %s \n s23 = %s \n",s3,s23);
18    printf("s23 contine adresa %p, iar s3 incepe la adresa %p \n", s23,s3);
19
20    strncpy(s4,s2,4);
21    printf("s4 = %s \n", s4);
22
23    return 0;
24 }
```

Functii predefinite pentru manipularea sirurilor de caractere

- compararea sirurilor – functiile **strcmp** si **strncpy**



```
exempluStrcmp.c      x

1 #include<stdio.h>
2 #include<string.h>
3
4 int main()
5 {
6     char s1[20] = "Nor";
7     char *s2 = "Noiembrrie";
8     char s3[10], s4[10];
9
10    int c = strcmp(s1,s2);
11    printf("c = %d \n", c);
12
13    c = strncmp(s1, s2, 2);
14    printf("c = %d \n", c);
15
16    char *s23 = strcpy(s3,s2);
17    printf("s3 = %s \n s23 = %s \n",s3,s23);
18    printf("s23 contine adresa %p, iar s3 incepe la adresa %p \n", s23,s3);
19
20    strncpy(s4,s2,4)
21    printf("s4 = %s \n",s4);
22    c = 9
23    c = 0
24    return 0;
}
```

```
Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ gcc exempluStrcmp.c
Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ ./a.out
c = 9
c = 0
s3 = Noiembrrie
s23 contine adresa 0x7fff57b06b26, iar s3 incepe la adresa 0x7fff57b06b26
s4 = Noie
```

Funcții predefinite pentru manipularea sirurilor de caractere

- concatenarea sirurilor – funcțiile **strcat** și **strncat**
 - antet: **char* strcat(char *d, const char* s);**
 - concatenează sirul sursă s la sirul destinație d.
 - returnează adresa sirului destinație
 - sirul rezultat are un '\0' la final
 - condiție: sirurile destinație și sursă nu se suprapun, alfel funcția prezintă **undefined behaviour**; (**strcat(s,s) =?**)

- antet: **char* strncat(char *d, const char* s, int n);**
- concatenează primele n caractere din sirul sursă s la sirul destinație d
- returnează adresa sirului destinație d
- sirul rezultat **NU** are un '\0' la final

Funcții predefinite pentru manipularea sirurilor de caractere

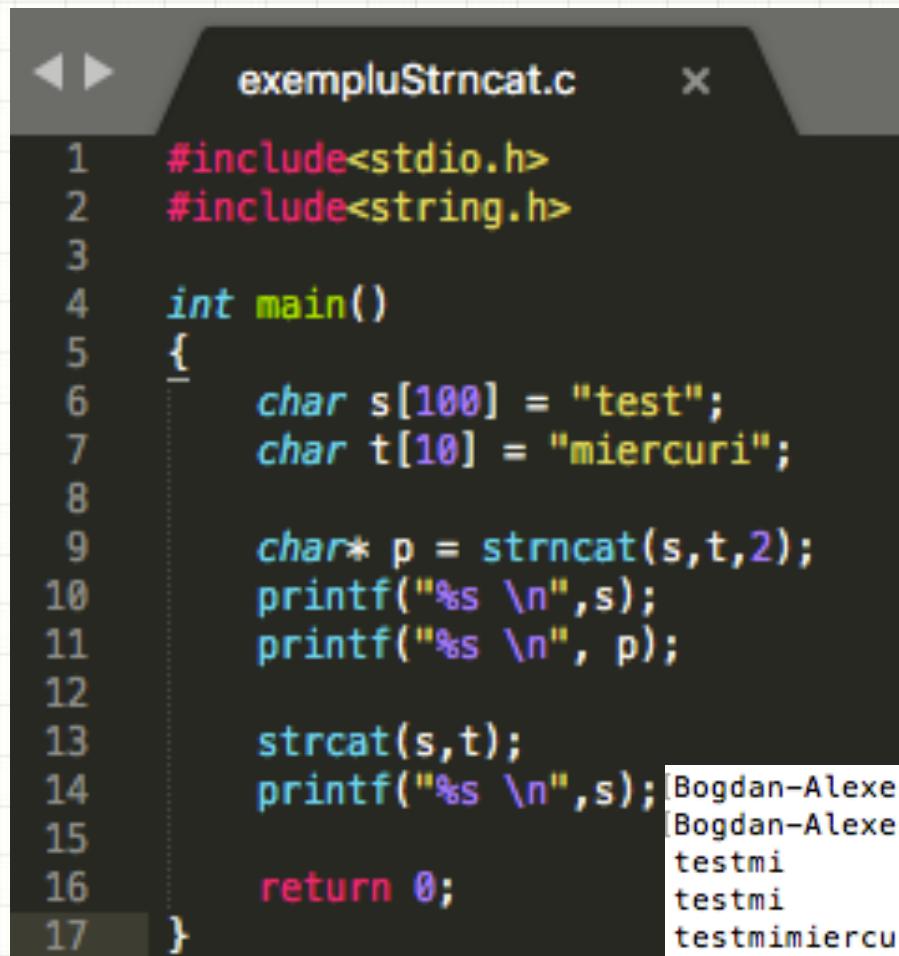
- concatenarea sirurilor – funcțiile **strcat** și **strncat**

```
exempluStrncat.c

1 #include<stdio.h>
2 #include<string.h>
3
4 int main()
5 {
6     char s[100] = "test";
7     char t[10] = "miercuri";
8
9     char* p = strncat(s,t,2);
10    printf("%s \n",s);
11    printf("%s \n", p);
12
13    strcat(s,t);
14    printf("%s \n",s);
15
16    return 0;
17 }
```

Funcții predefinite pentru manipularea sirurilor de caractere

- concatenarea sirurilor – funcțiile **strcat** și **strncat**



```
exempluStrncat.c      x

1 #include<stdio.h>
2 #include<string.h>
3
4 int main()
5 {
6     char s[100] = "test";
7     char t[10] = "miercuri";
8
9     char* p = strncat(s,t,2);
10    printf("%s \n",s);
11    printf("%s \n", p);
12
13    strcat(s,t);
14    printf("%s \n",s);
15
16    return 0;
17 }
```

```
Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ gcc exempluStrncat.c
Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ ./a.out
testmi
testmi
testmimiercuri
```

Funcții predefinite pentru manipularea sirurilor de caractere

- căutarea unui caracter într-un sir – funcțiile **strchr** și **strrchr**
 - antet: `char* strchr(const char *s, char c);`
 - cauță caracterul c în sirul s și întoarce un pointer la prima sa apariție
 - căutare de la stânga la dreapta
 - dacă nu apare caracterul c în sirul s returnează NULL

- antet: `: char* strrchr(const char *s, char c);`
- cauță caracterul c în sirul s și întoarce un pointer la prima sa apariție
- căutare de la dreapta la stânga
- dacă nu apare caracterul c în sirul s returnează NULL

Functii predefinite pentru manipularea sirurilor de caractere

- căutarea unui caracter într-un sir – funcțiile **strchr** și **strrchr**

```
exempluStrrchr.c      x

1 #include<stdio.h>
2 #include<string.h>
3
4 int main()
5 {
6     char s[] = "exemplu";
7     char litere[] = {'e','f','g'};
8     char *p;
9     int i;
10    for(i=0;i<3;i++)
11        if(strchr(s,litere[i]))
12        {
13            printf("%s \n",strchr(s,litere[i]));
14            printf("%s \n",strrchr(s,litere[i]));
15        }
16        else
17            printf("litera %c nu se gaseste in sirul %s \n",litere[i],s);
18
19
20    return 0;
21 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exempluStrrchr.c
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
exemplu
emplu
litera f nu se gaseste in sirul exemplu
litera g nu se gaseste in sirul exemplu
```

Funcții predefinite pentru manipularea sirurilor de caractere

- căutarea unui sir în alt sir – funcția **strstr**
 - antet: `char* strstr(const char *s, const char *t);`
 - cauță sirul t în sirul s și întoarce un pointer la prima sa apariție
 - căutare de la stânga la dreapta
 - dacă nu apare sirul t în sirul s returnează NULL
- exemplu: să se numere de câte ori apare un sir t într-un sir s.

Functii predefinite pentru manipularea sirurilor de caractere

- exemplu: să se numere de câte ori apare un sir t într-un sir s.

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

int main()
{
    char s[1000], t[1000];
    printf("Sirul s este : ");scanf("%s",s);
    printf("Sirul t este : ");scanf("%s",t);

    int nrAparitii = 0;
    char *p;

    p = strstr(s,t);

    while(p)
    {
        nrAparitii++;
        p = strstr(p+1,t);
    }

    printf("nrAparitii = %d \n",nrAparitii);

    return 0;
}
```

Functii predefinite pentru manipularea sirurilor de caractere

- exemplu: să se numere de câte ori apare un sir t într-un sir s.

```
01 exempluStrStr.c      x
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

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

int main()
{
    char s[1000], t[1000];
    printf("Sirul s este : ");scanf("%s",s);
    printf("Sirul t este : ");scanf("%s",t);

    int nrAparitii = 0;
    char *p;

    p = strstr(s,t);

    while(p)
    {
        nrAparitii++;
        p = strstr(p+1,t);
    }

    printf("nrAparitii = %d \n",nrAparitii);
    return 0;
}
```

```
Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exempluStrStr.c
Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
Sirul s este : abracadabra
Sirul t este : ab
nrAparitii = 2
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
Sirul s este : aaaa
Sirul t este : aa
nrAparitii = 3
```

Numără aparițiile suprapuse.
Dacă vreau aparițiile disjuncte?

Functii predefinite pentru manipularea sirurilor de caractere

- exemplu: să se numere de câte ori apare un sir t într-un sir s. Număr aparițiile disjuncte.

```
exempluStrStr.c      x
1 #include<stdio.h>
2 #include<string.h>
3
4 int main()
5 {
6     char s[1000], t[1000];
7     printf("Sirul s este : ");scanf("%s",s);
8     printf("Sirul t este : ");scanf("%s",t);
9
10    int nrAparitii = 0;
11    char *p;
12
13    p = strstr(s,t);
14
15    while(p)
16    {
17        nrAparitii++;
18        p = strstr(p+strlen(t),t);
19    }
20
21    printf("nrAparitii = %d \n",nrAparitii);
22
23    return 0;
24 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
Sirul s este : aaaa
Sirul t este : aa
nrAparitii = 2
```

Funcții predefinite pentru manipularea sirurilor de caractere

- Împărțirea unui sir în subșiruri – funcția **strtok**
 - antet: `char* strtok(char *s, const char *sep);`
 - împarte sirul s în subșiruri conform separatorilor din sirul sep
 - s = “Ana ; are . mere!!”, sep = “ ;,.!?” -> **Ana are mere**
 - string-ul inițial se trimită doar la primul apel al funcției, obținându-se primul subșir
 - la următoarele apeluri, pentru obținerea celorlalte subșiruri se trimit ca prim argument NULL
- exemplu: să se numere cuvintele dintr-o frază

Functii predefinite pentru manipularea sirurilor de caractere

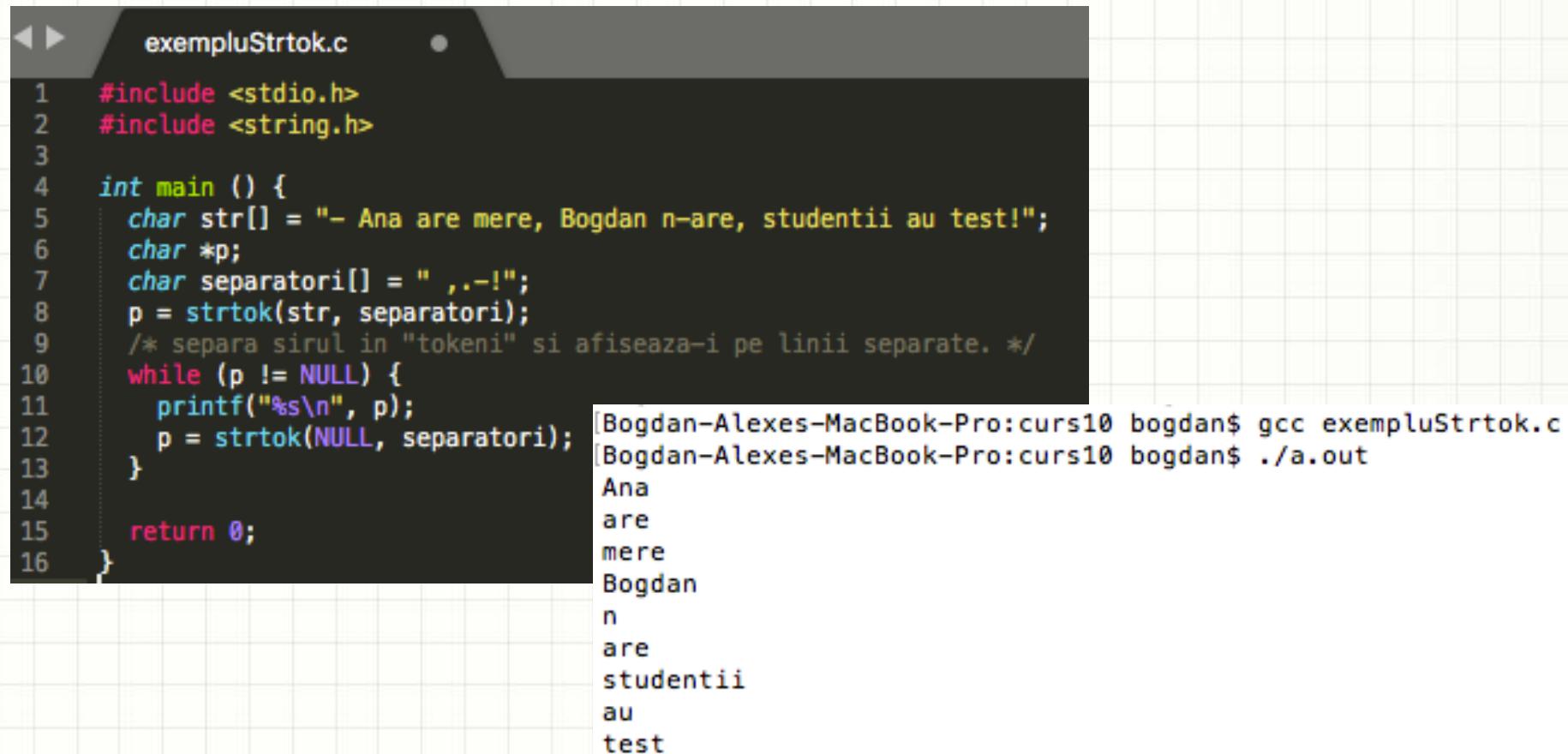
- Împărțirea unui sir în subșiruri – funcția **strtok**
- exemplu: să se numere cuvintele dintr-o frază

```
exempluStrtok.c

1 #include <stdio.h>
2 #include <string.h>
3
4 int main () {
5     char str[] = "- Ana are mere, Bogdan n-are, studentii au test!";
6     char *p;
7     char separatori[] = " ,.-!";
8     p = strtok(str, separatori);
9     /* separa sirul in "tokeni" si afiseaza-i pe linii separate. */
10    while (p != NULL) {
11        printf("%s\n", p);
12        p = strtok(NULL, separatori);
13    }
14
15    return 0;
16 }
```

Functii predefinite pentru manipularea sirurilor de caractere

- Împărțirea unui sir în subșiruri – funcția **strtok**
- exemplu: să se numere cuvintele dintr-o frază



```
exempluStrtok.c
1 #include <stdio.h>
2 #include <string.h>
3
4 int main () {
5     char str[] = "- Ana are mere, Bogdan n-are, studentii au test!";
6     char *p;
7     char separatori[] = " ,.-!";
8     p = strtok(str, separatori);
9     /* separa sirul in "tokeni" si afiseaza-i pe linii separate. */
10    while (p != NULL) {
11        printf("%s\n", p);
12        p = strtok(NULL, separatori);
13    }
14
15    return 0;
16 }
```

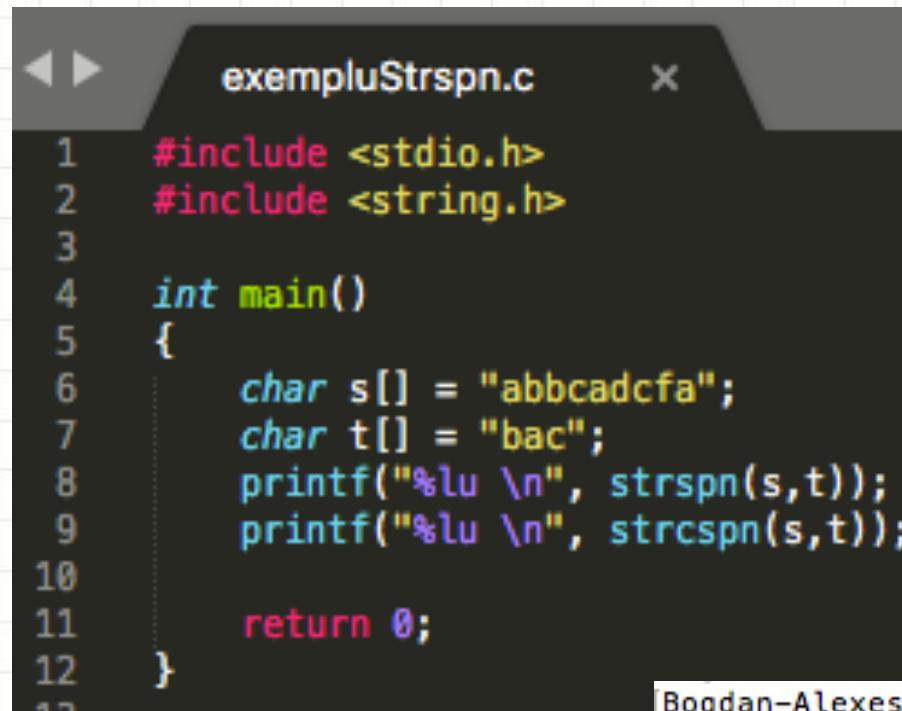
Bogdan-Alexes-MacBook-Pro:curs10 bogdan\$ gcc exempluStrtok.c
Bogdan-Alexes-MacBook-Pro:curs10 bogdan\$./a.out
Ana
are
mere
Bogdan
n
are
studentii
au
test

Funcții predefinite pentru manipularea sirurilor de caractere

- lungimea maximă a subșirului unui sir sursă **s** ce începe cu primul caracter și e format numai din caractere care apar într-un sir **t** – folosim funcțiile **strspn** și **strcspn**
 - antet: **int strspn(const char *s, const char* t);**
 - calculează lungimea maximă a subșirului din **s** ce începe cu primul caracter și e format din caractere care apar în sirul **t**; (**string span**)
 - returnează această lungime
 - antet: **int strcspn(const char *s,const char* t);**
 - calculează lungimea maximă subșirului din **s** ce începe cu primul caracter și e format din caractere care NU apar în sirul **t**; (**string complementary span**)
 - returnează această lungime

Funcții predefinite pentru manipularea sirurilor de caractere

- lungimea maximă a subșirului unui sir sursă **s** ce începe cu primul caracter și e format numai din caractere care apar într-un sir **t** – folosim funcțiile **strspn** și **strcspn**



The screenshot shows a code editor window with the file name "exempluStrspn.c". The code is as follows:

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

int main()
{
    char s[] = "abbcadcfa";
    char t[] = "bac";
    printf("%lu \n", strspn(s,t));
    printf("%lu \n", strcspn(s,t));

    return 0;
}
```

```
Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exempluStrspn.c
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
5
0
```

Functii predefinite pentru manipularea sirurilor de caractere

- funcțiile **strspn** și **strcspn** sunt folosite la validarea datelor:

```
exempluStrspn.c      x

1 #include <stdio.h>
2 #include <string.h>
3
4 int main()
{
5
6     char cifreNumar[20];
7     int nr;
8     printf("Dati numarul nr = ");scanf("%s",cifreNumar);
9
10    if (strspn(cifreNumar,"0123456789")==strlen(cifreNumar))
11    {
12        sscanf(cifreNumar,"%d",&nr);
13        printf("S-a citit numarul %d\n",nr);
14    }
15    else
16    {
17        printf("Eroare la citirea numarului \n");
18        exit(0);
19    }
20
21    return 0;
22 }
```

Dati numarul nr = 15674

Dati numarul nr = 0098

Dati numarul nr = -100

Dati numarul nr = 10bc34

Functii predefinite pentru manipularea sirurilor de caractere

- funcțiile **strspn** și **strcspn** sunt folosite la validarea datelor:

```
exemplStrspn2.c    x

1 #include <stdio.h>
2 #include <string.h>
3
4 int main()
5 {
6
7     char s[20];
8     int nr;
9     printf("Dati sirul s = ");scanf("%s",s);
10
11    if (strspn(s,"aeiouAEIOU")==strlen(s))
12        printf("S-a citit un sir numai din vocale \n");
13
14    if (strcspn(s,"aeiouAEIOU")==strlen(s))
15        printf("S-a citit un sir numai din consoane \n");
16
17    return 0;
18
19 }
```

Dati sirul s = aaaaaeeeiiiiIII00UU
S-a citit un sir numai din vocale

Dati sirul s = bcdDFTGHH
S-a citit un sir numai din consoane
^ _ " " " " " " " " " " " " " "

Funcții predefinite pentru manipularea sirurilor de caractere

- alte funcții predefinite (mai rar folosite):
 - `char* strpbrk(const char *s, const char* t);`
 - Întoarce adresa subșirului din `s` ce începe cu un caracter care se regăsește în sirul `t`; (`string pointer break`). Dacă nu găsește nici un caracter întoarce `NULL`.
 - `char* strdup(char *s);`
 - Întoarce adresa unei copii în HEAP a sirului `s`
 - `string duplicate`

Functii predefinite pentru manipularea sirurilor de caractere

The screenshot shows a terminal window with the title "exemplu.c". The code in the terminal is as follows:

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

int main()
{
    char *s="Alina";
    char t[]="Alina";
    char u[30];
    strcpy(u,"Alina");
    printf("%d %d %d\n",sizeof(s),sizeof(t),sizeof(u));
    printf("%d %d %d\n", strlen(s),strlen(t),strlen(u));
    char *ds=(char*)strdup(s);
    ds[3]='\0'; puts(ds); free(ds);
    strcpy(t,"Emil"); puts(t);
    strncpy(u+4,t+1,3); puts(u);
    u[7]='\0'; puts(u);
    strcat(u,s+2); puts(u);
    strncat(u,s,3); puts(u);
    printf("%d %d\n", strcmp(s,"Ali"),strncmp(s,"Ali",3));
    char *fs=strchr("Liliana",'a'); puts(fs);

    return 0;
}
```

The output of the program is:

8	6	30
5	5	5
Ali		
Emil		
Alinmil		
Alinmil		
Alinmilina		
AlinmilinaAli		
110	0	
ana		
-	-	-

Funcții predefinite pentru manipularea sirurilor de caractere

- conversia de la un sir la un număr și invers – funcțiile **sscanf** și **sprintf**
 - conversia de la sir la un număr poate fi făcută cu ajutorul funcției **sscanf** și descriptori de format potriviti
 - exemplu:

```
char *string="-45.8614";
double numar;
sscanf(string, "%lf", &numar);
printf("%f", numar);
```
- conversia de la un număr la sir poate fi făcută cu ajutorul funcției **sprintf** și descriptori de format potriviti
- exemplu:

```
char string[12];
int numar=897645671;
sprintf(string, "%d", numar);
printf("%s", string);
```

Funcții predefinite pentru manipularea sirurilor de caractere

- **int sscanf(char *sir, const char *format, adresa1, adresa2, ...)**

unde:

- **sir** este un sir de caractere din care se citesc datele
- **format** este un sir 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
- funcția **sscanf** întoarce numărul de câmpuri citite și depuse la adresele din listă. Dacă nu s-a stocat nici o valoare, funcția întoarce 0.
- singura deosebire față de funcția **scanf()** constă în faptul că datele sunt preluate dintr-o zonă de memorie, adresată de primul parametru (și nu de la intrarea standard = stdin).

Functii predefinite pentru manipularea sirurilor de caractere

- **int sscanf(char *sir, const char *format, adresa1, adresa2, ...)**

```
exemplSscanf.c x

1 #include <stdio.h>
2 #include <string.h>
3
4 int main()
5 {
6     int x,y,nr;
7     float f;
8     sscanf("-127","%d",&x);
9     printf("x = %d \n", x);
10    nr = sscanf("-127 -15 3.14","%d %d %f",&x,&y,&f);
11    printf("nr = %d, x = %d, y = %d, f = %f \n",nr,x,y,f);
12    sscanf("12a34","%d",&x);
13    printf("x = %d \n",x);
14
15    return 0;
16 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exemplSscanf.c
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
x = -127
nr = 3, x = -127, y = -15, f = 3.140000
x = 12
```

Funcții predefinite pentru manipularea sirurilor de caractere

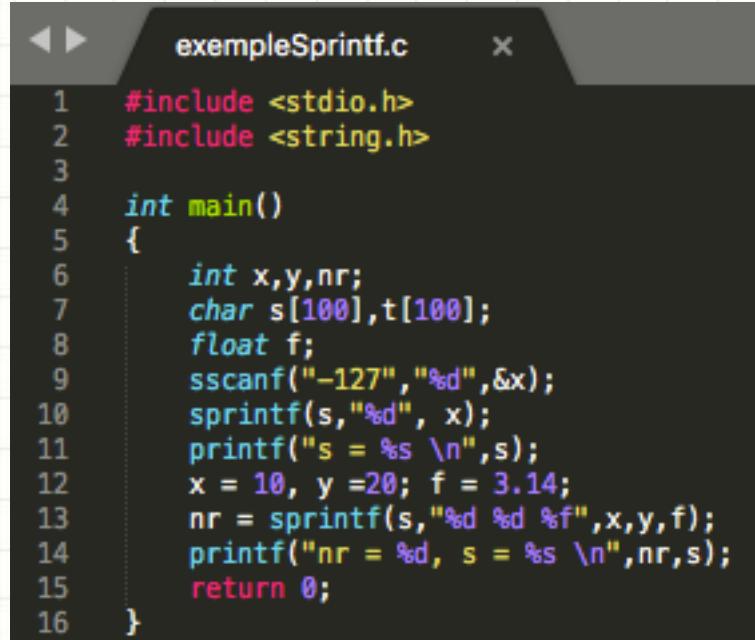
- **int sprintf(char *sir, const char *format, arg1, arg2, ...)**

unde:

- **sir** este un sir de caractere in care se scrie
- **format** este un sir de caractere ce definește textele și formatele datelor care se scriu
- **arg1, arg2,...** sunt expresii. Valorile lor se scriu in **sir** conform specificatorilor de format prezenti în format
- funcția **sprintf** întoarce numărul de octeți scriși sau -1 în caz de eșec.
- singura deosebire față de funcția **printf()** constă în faptul că datele sunt scrise într-o zonă de memorie, adresată de primul parametru (și nu la iesirea standard = stdout).

Functii predefinite pentru manipularea sirurilor de caractere

- **int sprintf(char *sir, const char *format, arg1, arg2, ...)**



```
exemplSprintf.c
1 #include <stdio.h>
2 #include <string.h>
3
4 int main()
5 {
6     int x,y,nr;
7     char s[100],t[100];
8     float f;
9     sscanf("-127","%d",&x);
10    sprintf(s,"%d", x);
11    printf("s = %s \n",s);
12    x = 10, y =20; f = 3.14;
13    nr = sprintf(s,"%d %d %f",x,y,f);
14    printf("nr = %d, s = %s \n",nr,s);
15    return 0;
16 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exemplSprintf.c
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
s = -127
nr = 14, s = 10 20 3.140000
```

Funcții predefinite pentru manipularea sirurilor de caractere

- funcții de clasificare (macro-uri) a caracterelor (nu a sirurilor de caractere)
- sunt în fișierul ctype.h

`islower(c)` 1 dacă $c \in \{‘a’..‘z’\}$, 0 altfel

`isupper(c)` 1 dacă $c \in \{‘A’..‘Z’\}$, 0 altfel

`isalpha(c)` 1 dacă $c \in \{‘A’..‘Z’\} \cup \{‘a’..‘z’\}$, 0 altfel

`isdigit(c)` 1 dacă $c \in \{‘0’..‘9’\}$, 0 altfel

`isxdigit(c)` 1 dacă $c \in \{‘0’..‘9’\} \cup \{‘A’..‘F’\} \cup \{‘a’..‘f’\}$, 0 altfel

`isalnum(c)` 1 dacă `isalpha(c)` || `isdigit(c)`, 0 altfel

`isspace(c)` 1 dacă $c \in \{‘ ‘, ‘\n’, ‘\t’, ‘\r’, ‘\f’, ‘\v’\}$, 0 altfel

`isgraph(c)` 1 dacă c este afișabil, fără spațiu, 0 altfel

`isprint(c)` 1 dacă c este afișabil, cu spațiu, 0 altfel

`ispunct(c)` 1 dacă `isgraph(c)` && `isalnum(c)`, 0 altfel

- conversia din literă mare în literă mică și invers se face folosind funcțiile: `tolower(c)` și `toupper(c)`.

Functii predefinite pentru manipularea sirurilor de caractere

- funcție care convertește un sir de caractere reprezentând un număr întreg, într-o valoare întreagă. Numărul poate avea semn și poate fi precedat de spații albe.

```
exempluCtype.C
```

```
1 #include<stdio.h>
2 #include<string.h>
3 #include<ctype.h>
4
5 int conversie(char *s)
6 { int i, numar, semn;
7     for(i=0; isspace(s[i]); i++);
8
9     semn = (s[i]=='-')?-1:1;
10
11    if(s[i]=='+') || s[i]=='-')
12        i++;
13
14    for(numar=0;isdigit(s[i]);i++)
15        numar=10*numar+(s[i]-'0');
16
17    return semn*numar;
18 }
19
20 int main()
21 {
22     char s[20];
23     strcpy(s, " -123456789");
24     printf("Sirul s = %s e transformat in valoarea = %d\n", s, conversie(s));
25
26     return 0;
27 }
```

Sirul s = -123456789 e transformat in valoarea = -123456789

Functii predefinite pentru manipularea sirurilor de caractere

- funcție care convertește un întreg într-un sir de caractere pentru baza 10. Întregul poate avea semn.

The screenshot shows a terminal window with two parts. The left part displays the source code for 'exempluCtype1.C'. The right part shows the terminal output with the program's execution and its result.

```
exempluCtype1.C
1 #include<stdio.h>
2 #include<string.h>
3 #include<ctype.h>
4
5 void inversare(char[]);
6
7 void conversieIA(int n, char s[]){
8     int j, semn;
9     if((semn=n)<0)
10        n=-n;
11     j=0;
12     do
13         s[j++]=n%10+'0';
14     while ((n/=10)>0);
15     if(semn<0)
16         s[j++]='-';
17     s[j]='\0';
18     inversare(s);
19 }
20
21 void inversare(char s[])
22 { int i,j;
23     char c;
24     for(i=0,j=strlen(s)-1;i<j;i++,j--)
25         c=s[i], s[i]=s[j], s[j]=c;
26 }
27
28 int main()
29 {
30     char s[20];
31     int n = -123456789;
32     conversieIA(n,s);
33     printf("Sirul s = %s \n", s);
34     return 0;
35 }
```

[Bogdan-Alexes-MacBook-Pro:curs10 bogdan\$ gcc exempluCtype1.c
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan\$./a.out
Sirul s = -123456789

Functii predefinite pentru manipularea sirurilor de caractere

- funcție care convertește un întreg fără semn într-un sir de caractere în baza 16.

```
001 // exempluCtype2.C
002
003 #include<stdio.h>
004 #include<string.h>
005 #include<ctype.h>
006
007
008 void inversare(char[]);
009 void conversieI16A(int n, char s[]){
010     int j;
011     j=0;
012     char baza16[] ="0123456789abcdef";
013     do { s[j++] = baza16[n%16]; } while ((n/=16)>0);
014     s[j]='\0';
015     inversare(s);
016 }
017
018 void inversare(char s[])
019 {
020     int i,j;
021     char c;
022     for(i=0,j=strlen(s)-1;i<j;i++,j--)
023         c=s[i], s[i]=s[j], s[j]=c;
024 }
025
026
027 int main()
028 {
029     char s[20];
030     int n = 123;
031     conversieI16A(n,s);
032     printf("Sirul s = %s \n", s);
033     return 0;
034 }
```

```
Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exempluCtype2.c
Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
Sirul s = 7b
```

Funcții predefinite pentru manipularea blocurilor de memorie

- copierea elementelor unui tablou **a** într-un alt tablou **b**:
 - nu se poate face prin atribuire (**b=a**), întrucât **a** și **b** sunt pointeri constanti;
 - copierea se face element cu element folosind instrucțiuni repetitive (for, while);
 - pentru stringuri (tablouri de caractere) avem funcțiile predefinite **strcpy** și **strncpy**:
 - **char* strcpy(char *d, const char* s);**
 - copiază sirul sursă **s** în sirul destinație **d**;
 - returnează adresa sirului destinație
 - sirul rezultat are un '\0' la final
 - **char* strncpy(char *d,const char* s, int n);**
 - copiază primele n caractere sirul sursă **s** în sirul destinație **d**;
 - returnează adresa sirului destinație
 - sirul rezultatul **NU** are un '\0' la final

Funcții predefinite pentru manipularea blocurilor de memorie

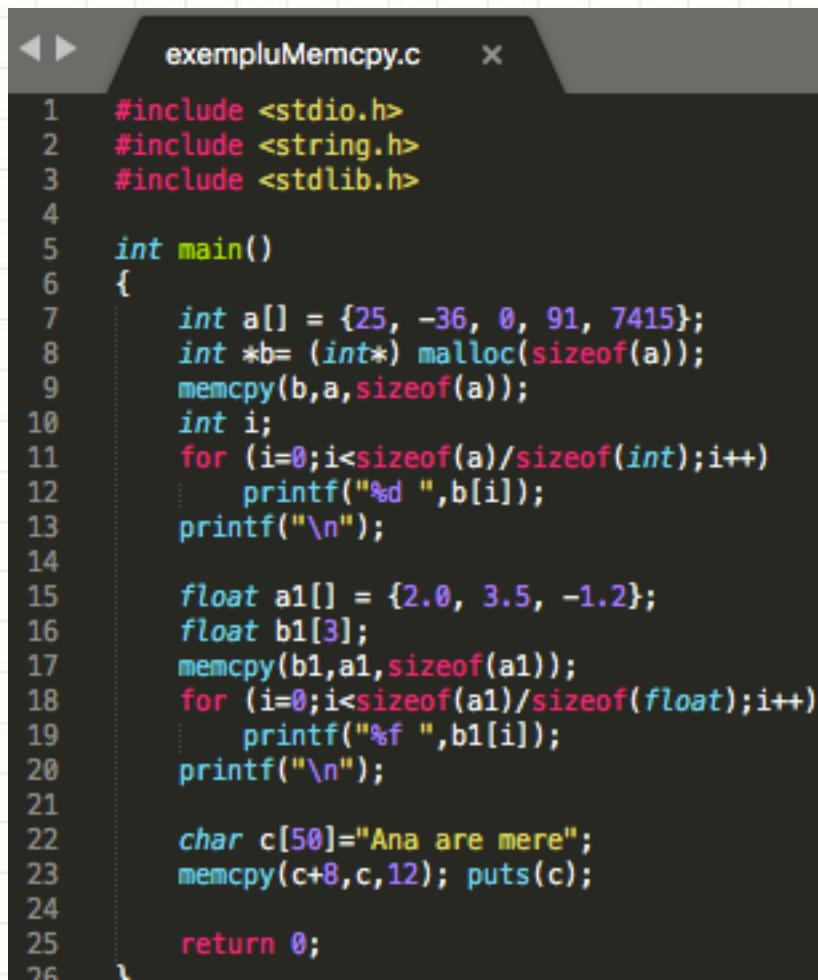
- copierea elementelor unui tablou **a** într-un alt tablou **b**:
 - nu se poate face prin atribuire ($b=a$), întrucât **a** și **b** sunt pointeri constanti;
 - copierea se face element cu element folosind instrucțiuni repetitive (for, while);
 - pe **cazul general** (a și b nu sunt neapărat tablouri de caractere) putem folosi **funcții pentru manipularea blocurilor de memorie: memcpy, memmove;**
 - lucrează la nivel de octet fără semn (unsigned char)
 - alte funcții pentru manipularea blocurilor de memorie: memcmp, memset, memchr

Funcții predefinite pentru manipularea blocurilor de memorie

- copierea unui tablou – funcțiile **memcpy** și **memmove**
 - antet: **void* memcpy(void *d, const void* s, int n);**
 - copiază primii **n** octeți din sursa **s** în destinația **d**;
 - returnează un pointer la începutul zonei de memorie destinație **d**;
 - un fel de **strcpy** extins (merge și pe alte tipuri de date, nu numai pe char-uri)
 - nu se oprește la octeți = 0 (funcția strcpy se oprește la octeți ce au valoarea 0 = sfârșit de string);
 - presupune că sirurile destinație și sursa nu se suprapun
 - dacă cele două siruri se suprapun funcția prezintă **undefined behaviour** (comportament nedefinit)

Functii predefinite pentru manipularea blocurilor de memorie

- copierea unui tablou – funcțiile **memcpy** și **memmove**
 - antet: **void* memcpy(void *d, const void* s, int n);**



The screenshot shows a code editor window with the title "exempluMemcpy.c". The code is a C program demonstrating the use of the `memcpy` function. It includes headers for stdio.h, string.h, and stdlib.h, defines main(), and uses `malloc` to allocate memory for arrays a and b. It then copies the contents of array a to array b using `memcpy`. The program then prints the elements of array b and array c. The code is color-coded for readability.

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

int main()
{
    int a[] = {25, -36, 0, 91, 7415};
    int *b= (int*) malloc(sizeof(a));
    memcpy(b,a,sizeof(a));
    int i;
    for (i=0;i<sizeof(a)/sizeof(int);i++)
        printf("%d ",b[i]);
    printf("\n");

    float a1[] = {2.0, 3.5, -1.2};
    float b1[3];
    memcpy(b1,a1,sizeof(a1));
    for (i=0;i<sizeof(a1)/sizeof(float);i++)
        printf("%f ",b1[i]);
    printf("\n");

    char c[50] = "Ana are mere";
    memcpy(c+8,c,12);
    puts(c);

    return 0;
}
```

Functii predefinite pentru manipularea blocurilor de memorie

- copierea unui tablou – funcțiile **memcpy** și **memmove**
 - antet: **void* memcpy(void *d, const void* s, int n);**

The screenshot shows a terminal window with two parts. On the left, the code file `exempluMemcpy.c` is displayed. On the right, the terminal output is shown.

`exempluMemcpy.c` content:

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

int main()
{
    int a[] = {25, -36, 0, 91, 7415};
    int *b= (int*) malloc(sizeof(a));
    memcpy(b,a,sizeof(a));
    int i;
    for (i=0;i<sizeof(a)/sizeof(int);i++)
        printf("%d ",b[i]);
    printf("\n");

    float a1[] = {2.0, 3.5, -1.2};
    float b1[3];
    memcpy(b1,a1,sizeof(a1));
    for (i=0;i<sizeof(a1)/sizeof(float);i++)
        printf("%f ",b1[i]);
    printf("\n");

    char c[50] = "Ana are mere";
    memcpy(c+8,c,12);
    puts(c);

    return 0;
}
```

Terminal Output:

```
Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exempluMemcpy.c
Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
25 -36 0 91 7415
2.000000 3.500000 -1.200000
Ana are Ana are Ana
25 -36 0 91 7415
2.000000 3.500000 -1.200000
Ana are Ana are mere
```

Funcții predefinite pentru manipularea blocurilor de memorie

- copierea unui tablou – funcțiile **memcpy** și **memmove**
 - antet: **void* memmove(void *d, const void* s, int n);**
 - copiază primii **n** octeți din sursa **s** în destinația **d**;
 - returnează un pointer la începutul zonei de memorie destinație **d**;
 - identică cu funcția **memcpy** + tratează cazurile de suprapunere dintre **d** și **s**
 - nu contează că sirurile destinație **d** și sursă **s** se suprapun
 - folosește un buffer intern pentru copiere

Functii predefinite pentru manipularea blocurilor de memorie

- copierea unui tablou – funcțiile **memcpy** și **memmove**
 - antet: **void* memmove(void *d, const void* s, int n);**



The screenshot shows a code editor window titled "exempluMemmove1.c". The code is written in C and demonstrates the use of the **memmove** function. It includes three **#include** directives for **<stdio.h>**, **<string.h>**, and **<stdlib.h>**. The **main** function contains two memory manipulation statements. The first statement moves 12 bytes from memory location **c+8** to **c**. The second statement moves 16 bytes from memory location **t + 20** to **t + 13**. Both statements use the **puts** function to print the modified strings to the console.

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

int main()
{
    char c[50] = "Ana are mere";
    memmove(c + 8, c, 12);
    puts(c);

    char t[] = "memmove este foarte folositor.....";
    memmove(t + 20, t + 13, 16);
    puts(t);

    return 0;
}
```

```
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exempluMemmove1.c
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
Ana are Ana are mere
memmove este foarte foarte folositor.....]
```

Functii predefinite pentru manipularea blocurilor de memorie

- funcție care elimină toate aparițiile unei siruri dintr-un sir

```
exempluMemmove2.c  x

1 #include <stdio.h>
2 #include <string.h>
3
4 void eliminaAparitii(char *s, char* t)
5 {
6     char *p = strstr(s,t);
7     while(p != NULL)
8     {
9         memmove(p,p+strlen(t),s + strlen(s)- (p + strlen(t)) + 1);
10        p = strstr(s,t);
11    }
12 }
13
14 int main()
15 {
16     char s[100],t[100];
17     strcpy(s,"abbbccca");
18     strcpy(t,"bc");
19     eliminaAparitii(s,t);
20     printf("%s\n",s);
21     return 0;
22 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exempluMemmove2.c
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
aa
```

Nu obțin ceea ce trebuie, unde am greșit?

Functii predefinite pentru manipularea blocurilor de memorie

- funcție care elimină toate aparițiile unei siruri dintr-un sir

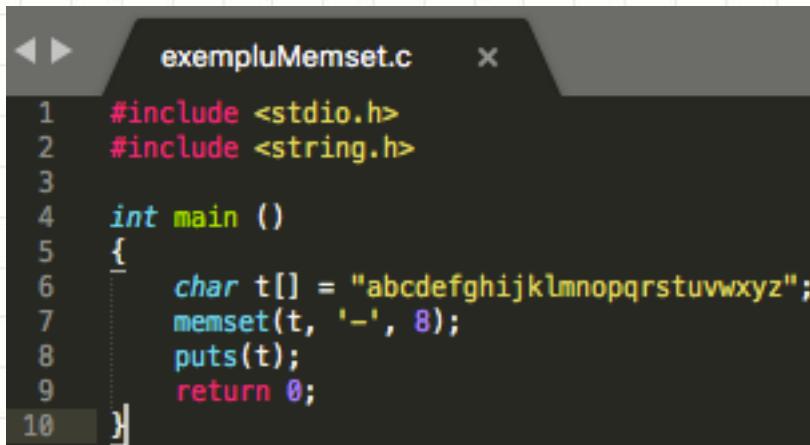
```
exempluMemmove3.c  x

1 #include <stdio.h>
2 #include <string.h>
3
4 void eliminaAparitii(char *s, char* t)
5 {
6     char *p = strstr(s,t);
7     while(p != NULL)
8     {
9         memmove(p,p+strlen(t),s + strlen(s)- (p + strlen(t)) + 1);
10        p = strstr(p,t);
11    }
12 }
13
14 int main()
15 {
16     char s[100],t[100];
17     strcpy(s,"abbbccca");
18     strcpy(t,"bc");
19     eliminaAparitii(s,t);
20     printf("%s\n",s);
21     return 0;
22 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exempluMemmove3.c
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
abbcca
```

Funcții predefinite pentru manipularea blocurilor de memorie

- setarea unor octeți la o valoare – funcția **memset**
 - antet: **void* memset(void *d, char val, int n);**
 - În zona de memorie dată de pointerul d, sunt setate primele n poziții (octeți) la valoarea dată de val. Funcția returnează șirul d.



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

int main ()
{
    char t[] = "abcdefghijklmnopqrstuvwxyz";
    memset(t, '-', 8);
    puts(t);
    return 0;
}
```

```
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exempluMemset.c
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
-----ijklmnopqrstuvwxyz
```

Funcții predefinite pentru manipularea blocurilor de memorie

- căutarea unui octet într-un tablou – funcția **memchr**
 - antet: **void* memchr(const void *d, char c, int n);**
 - determină prima apariție a octetului **c** în zona de memorie dată de pointerul **d** și care conține **n** octeți. Funcția returnează pointerul la prima apariție a lui **c** în **d** sau **NULL**, dacă **c** nu se găsește în **d**.

```
exempluMemchr.c  x
1 #include <stdio.h>
2 #include <string.h>
3
4 int main ()
5 {
6     char t[] = "abcdefghijklmnopqrstuvwxyz";
7     char *p = memchr(t, 'm', 25);
8     puts(p);
9
10    return 0;
11 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exempluMemchr.c
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
mnopqrstuvwxyz
```

Funcții predefinite pentru manipularea blocurilor de memorie

- compararea a două tablouri pe octeți – funcția **memcmp**
 - antet: `int memcmp(const void *s1, const void * s2, int n);`
 - compara primii **n** octeți corespondenți începând de la adresele **s1** și **s2**.
 - returnează 0 dacă octetii sunt identici, ceva mai mic decât 0 dacă **s1 _L s2**, ceva mai mare decât 0 dacă **s1 >sub>L</sub> s2**

Functii predefinite pentru manipularea blocurilor de memorie

- compararea a două tablouri pe octeți – funcția **memcmp**
 - antet: **int memcmp(const void *s1, const void * s2, int n);**
 - compara primii **n** octeți corespondenți începând de la adresele **s1** și **s2**.

```
exempluMemcmp.c  x

1 #include <stdio.h>
2 #include <string.h>
3
4 int main ()
5 {
6     char t[] = "Ana are mere!!!";
7     char s[] = "Ana are pere!!!";
8     int i = memcmp(t,s,6);
9     printf("%d \n",i);
10    i = memcmp(t,s,strlen(t));
11    printf("%d \n",i);
12
13    int v[] = {1,2,4,5,6};
14    int w[] = {1,2,3,7,8};
15    i = memcmp(v,w,8);
16    printf("%d \n",i);
17    i = memcmp(v,w,sizeof(v));
18    printf("%d \n",i);
19
20    return 0;
21 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exempluMemcmp.c
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
0
-3
0
1
```

Functiile din string.h

<code>char* strcpy(char* d,const char* s)</code>	copiază sirul s în d , inclusiv '\0', întoarce d
<code>char* strncpy(char* d,const char* s, int n)</code>	copiază n caractere din sirul s în d , completând eventual cu '\0', întoarce d
<code>char* strcat(char* d,const char* s)</code>	concatenează sirul s la sfârșitul lui d , întoarce d
<code>char* strncat(char* d,const char* s, int n)</code>	concatenează cel mult n caractere din sirul s la sfârșitul lui d , completând cu '\0', întoarce d
<code>int strcmp(const char* d, const char* s)</code>	compară sirurile d și s , întoarce -1 dacă d < s , 0 dacă d == s și 1 dacă d > s
<code>int stricmp(const char* d, const char* s)</code>	compară sirurile d și s (ca și <code>strcmp()</code>) fără a face distincție între litere mari și mici
<code>int strncmp(const char* d, const char* s, int n)</code>	similar cu <code>strcmp()</code> , cu deosebirea că se compară cel mult n caractere
<code>char* strchr(const char* d,char c)</code>	caută caracterul c în sirul d ; întoarce un pointer la prima apariție a lui c în d , sau NULL
<code>char* strrchr(const char* d,char c)</code>	întoarce un pointer la ultima apariție a lui c în d , sau NULL
<code>char* strstr(const char* d, const char* s)</code>	întoarce un pointer la prima apariție a subșirului s în d , sau NULL
<code>char* strpbrk(const char* d, const char* s)</code>	întoarce un pointer la prima apariție a unui caracter din subșirul s în d , sau NULL

Functiile din string.h

<code>int strspn(const char* d, const char* s)</code>	întoarce lungimea prefixului din d care conține numai caractere din s
<code>int strcspn(const char* d, const char* s)</code>	întoarce lungimea prefixului din d care conține numai caractere ce nu apar în s
<code>int strlen(const char* s)</code>	întoarce lungimea lui s ('\'0' nu se numără)
<code>void* memcpy(void* d, const void* s,int n)</code>	copiaza n octeți din s în d ; întoarce d
<code>void* memmove(void* d, const void* s,int n)</code>	ca și <code>memcpy</code> , folosită dacă s și d se întrepătrund
<code>void* memset(void* d,const int c, int n)</code>	copiază caracter c în primele n poziții din d
<code>int memcmp(const void* d, const void* s,int n)</code>	compară zonele adresate de s și d
<code>char* strtok(const char* d, const char* s)</code>	caută în d subșirurile delimitate de caracterele din s ; primul apel întoarce un pointer la primul subșir din d care nu conține caractere din s următoarele apeluri se fac cu primul argument NULL , întorcându-se de fiecare dată un pointer la următorul subșir din d ce nu conține caractere din s ; în momentul în care nu mai există subșiruri, funcția întoarce NULL

Implementarea diverselor funcții din string.h

- ❑ putem scrie propriile funcții care realizează operații pe șiruri de caractere
- ❑ câteva exemple:
 - ❑ lungimea unui șir -> strlen
 - ❑ copierea unui șir în alt șir -> strcpy
 - ❑ compararea lexicografică a două șiruri -> strcmp
- ❑ implementări cu tablouri și pointeri

Lungimea unui sir

```
lungimeSir.c      x

1 #include <stdio.h>
2 #include<string.h>
3
4 int lungimeSir1(char *s)
5 {
6     int lungime = 0, i;
7     for(i = 0; s[i]; i++)
8         lungime++;
9     return lungime;
10 }
11
12 int lungimeSir2(char *s)
13 {
14     char *p=s;
15     while (*p)
16         p++;
17     return p-s;
18 }
19
20
21 int main()
22 {
23     char s[1000];
24     strcpy(s,"Azi e miercuri");
25     printf("lungime sir = %d \n",lungimeSir1(s));
26     printf("lungime sir = %d \n",lungimeSir2(s));
27     return 0;
28 }
```

Ce se intampla daca pun
while (*p++)?

```
Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc lungimeSir.c
Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
lungime sir = 14
lungime sir = 14
```

Copierea unui sir în alt sir

```
copiereSir.C x

1 #include <stdio.h>
2
3 char* copiazaSir1(char *d, char* s)
4 {
5     int i=0;
6     while (d[i] = s[i])
7         i++;
8     return d;
9 }
10
11 void copiazaSir2(char *d, char* s)
12 {
13     int i;
14     while (*d = *s)
15     {
16         d++;
17         s++;
18     }
19 }
20
21
22 int main()
23 {
24     char s[1000], t[1000];
25     copiazaSir1(s,"Azi e miercuri");
26     copiazaSir1(t,s);
27     printf("Sirul t este: %s \n",t);
28     copiazaSir2(t,s);
29     printf("Sirul t este: %s \n",t);
30     return 0;
31 }
```

```
-----  
Sirul t este: Azi e miercuri  
Sirul t este: Azi e miercuri  
-----
```

Compararea lexicografică a două siruri

```
comparareSiruri.C x

1 #include <stdio.h>
2 #include <string.h>
3
4 int comparaSiruri1(char* s1, char *s2)
5 {
6     int j;
7     for(j=0; s1[j]==s2[j]; j++)
8         if(s1[j]=='\0')
9             return 0;
10    return s1[j]-s2[j];
11 }
12
13 int comparaSiruri2(char* s1, char *s2)
14 {
15     for(; *s1==*s2; s1++,s2++)
16         if(*s1=='\0')
17             return 0;
18     return *s1-*s2;
19 }
20
21
22 int main()
23 {
24     char s[1000], t[1000];
25     strcpy(s,"Azi e miercuri");
26     strcpy(t,"Azi e joi");
27
28     int cmp = comparaSiruri1(s,t);
29     printf("%d %d \n",comparaSiruri1(s,t),comparaSiruri2(s,t));
30     return 0;
31 }
```

F / CUISS
3 3
-