



PROGRAMARE PROCEDURALĂ

Bogdan Alexe

bogdan.alexe@fmi.unibuc.ro

Secția Informatică, anul I,
2016-2017

Cursul 9

Amazon TechO(n) Challenge, 9-16 decembrie

 challenge.amazontechon.com/#/teaser



AU MAI RĂMAS
03 ZILE
20 17 05
ORE MINUTE SECUNDE

Până la începerea concursului, poți consulta [regulamentul](#).

Dacă ai întrebări, contactează-ne la adresa techonchallenge@amazon.com sau pe Twitter

 [TechOnChallenge](#)

 Share  Share

Dacă ești pasionat de rezolvarea unor probleme informaticice complexe, te invităm să participe la a patra ediție a concursului "Amazon TechO(n) Challenge". Competiția are loc online, deci nu trebuie să te deplasezi către o anumită locație; totul e să ai acces la internet pe durata derulării lui. Premii:

- **Locul I: 10000 RON**
- **Locul II: 5000 RON**
- **Locul III: 2500 RON**

Autentifică-te folosind contul tău Facebook sau Google pentru a participa, iar noi îți vom reaminti pe email cu o zi înainte de începerea concursului,

Mult succes!



Development Center
Romania

Problema din 2015



Problema

Începând cu anul acesta Amazon experimentează livrarea comenzi online prin drone. Dronele sunt încărcate cu coletele ce trebuie livrate, iar sarcina lor este să ducă coletul la o anumită destinație indiferent de obstacolele care pot apărea pe traseu. Problema de anul acesta vă propune să scrieți codul pentru a ghida o astfel de dronă pe hărți arbitrate.

Organizare

1. Data examenului scris din sesiune.
2. Data testului de laborator.

Cursul trecut

1. Alocarea dinamică a memoriei.
2. Clase de alocare/memorare.
3. Siruri de caractere: funcții specifice de manipulare.

Programa cursului

- Introducere**
 - Algoritmi.
 - Limbaje de programare.
 - Introducere în limbajul C. Structura unui program C.
 - Complexitatea algoritmilor.
- Fundamentele limbajului C**
 - Tipuri de date fundamentale. Variabile. Constante. Operatori. Expresii. Conversii.
 - Instrucțiuni de control
 - Directive de preprocesare. Macrodefiniții.
 - Funcții de citire/scriere.
 - Etapele realizării unui program C.
- Tipuri deriveate de date**
 - Tablouri. Siruri de caractere.
 - Structuri, uniuni, câmpuri de biți, enumerări.
 - Pointeri.
- Funcții (1)**
 - Declarare și definire. Apel. Metode de trasmitere a parametrilor.
 - Pointeri la funcții.
- Tablouri și pointeri**
 - Legătura dintre tablouri și pointeri
 - Aritmetică pointerilor
 - Alocarea dinamică a memoriei
 - Clase de memorare
- Şiruri de caractere**
 - Funcții specifice de manipulare.
- Fișiere text și 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.
 - Programare generică.
- Recursivitate**

Cursul de azi

1. Siruri de caractere: funcții specifice de manipulare
2. Fișiere: noțiuni generale

Functii predefinite pentru manipularea sirurilor de caractere

- funcții de procesare a sirurilor de caractere specifice incluse în fișierul string.h
- lungime: **int strlen(const char *s)**
- copiere: **char* strcpy(char *d, const char* s);**
- copiere: **char* strncpy(char *d, const char* s, int n);**
- comparare: **int strcmp(const char *s1, const char* s2);**
- comparare: **int strncmp(const char *s1, const char* s2, int n);**
- concatenare: **char* strcat(char *d, const char* s);**
- concatenare: **char* strncat(char *d, const char* s, int n);**
- căutare caracter: **char* strchr(const char *s, char c);**
- căutare caracter: **char* strrchr(const char *s, char c);**
- căutare sir : **char* strstr(const char *s, const char * t);**
- împărțire în subșiruri: **char* strtok(char *s, const char *sep);**
- conversie: **sprintf, sscanf**

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, ...)**

exemplusscanf.c

```
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 }
```

```
., Exemplu, Exemplusscanf...
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 intrarea standard = stdin).

Functii predefinite pentru manipularea sirurilor de caractere

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

exempluSprintf.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 }
```

```
s = -127
nr = 14, s = 10 20 3.140000
```

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(char *s, 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(char *s, 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**

exempluStrspn.c

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main()
5 {
6     char s[] = "abbcadcf";
7     char t[] = "bac";
8     printf("%d \n", strspn(s, t));
9     printf("%d \n", strcspn(s, t));
10
11     return 0;
12 }
```

5
0

Functii predefinite pentru manipularea sirurilor de caractere

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

The screenshot shows a code editor window titled "exempluStrspn.c". The code is as follows:

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main()
5 {
6
7     char cifreNumar[20];
8     int nr;
9     printf("Dati numarul nr = ");scanf("%s",cifreNumar);
10
11    if (strspn(cifreNumar,"0123456789")==strlen(cifreNumar))
12    {
13        sscanf(cifreNumar,"%d",&nr);
14        printf("S-a citit numarul %d\n",nr);
15    }
16    else
17    {
18        printf("Eroare la citirea numarului \n");
19        exit(0);
20    }
21    return 0;
22
23 }
```

To the right of the code, there are four execution results:

- 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

```
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 = aaaaaeeeiiiiIIIOOOUU
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(char *s, 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

exemplu.c

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main()
{
5     char *s="Alina";
6     char t[]="Alina";
7     char u[30];
8     strcpy(u,"Alina");
9     printf("%d %d %d\n",sizeof(s),sizeof(t),sizeof(u));
10    printf("%d %d %d\n", strlen(s),strlen(t),strlen(u));
11    char *ds=(char*)strdup(s);
12    ds[3]='\0'; puts(ds); free(ds);
13    strcpy(t,"Emil"); puts(t);
14    strncpy(u+4,t+1,3); puts(u);
15    u[7]='\0'; puts(u);
16    strcat(u,s+2); puts(u);
17    strncat(u,s,3); puts(u);
18    printf("%d %d\n", strcmp(s,"Ali"),strncmp(s,"Ali",3));
19    char *fs=strchr("Liliana",'a'); puts(fs);
20
21    return 0;
22
23 }
```

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

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

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

exempluCtype1.C

```
1 #include<stdio.h>
2 #include<ctype.h>
3
4 int main()
5 {
6     char * t = "9 Portocale\n3 Pere";
7     printf("%s\n",t);
8     printf("%d\n",isdigit(t[0]));
9     printf("%d\n",isalpha(t[1]));
10    printf("%d\n",islower(t[2]));
11    printf("%d\n",isupper(t[2]));
12    printf("%d\n",isspace(t[11]));
13    printf("%c\n",tolower(t[2]));
14    printf("%c\n",toupper(t[4]));
15
16    return 0;
17
18 }
```

9 Portocale
3 Pere
1
0
0
1
1
p
R

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
8   semn = (s[i]=='-')?-1:1;
9
10  if(s[i]=='+') || s[i]=='-')
11    i++;
12
13  for(numar=0;isdigit(s[i]);i++)
14    numar=10*numar+(s[i]-'0');
15
16  return semn*numar;
17 }
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 în baza 10. Întregul poate avea semn.

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

Functii predefinite pentru manipularea sirurilor de caractere

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

```
exempluCtype2.C ×
1 #include<stdio.h>
2 #include<string.h>
3 #include<ctype.h>
4
5 void inversare(char []);
6 void conversieI16A(int n, char s[]){
7     int j;
8     j=0;
9     char baza16[] = "0123456789abcdef";
10    do { s[j++] = baza16[n%16]; } while ((n/=16)>0);
11    s[j]='\0';
12    inversare(s);
13 }
14
15 void inversare(char s[])
16 {
17     int i,j;
18     char c;
19     for(i=0,j=strlen(s)-1;i<j;i++,j--)
20         c=s[i], s[i]=s[j], s[j]=c;
21 }
22
23 int main()
24 {
25     char s[20];
26     int n = 123;
27     conversieI16A(n,s);
28     printf("Sirul s = %s \n", s);
29     return 0;
30 }
```

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, 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, 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 – functiile **memcpy** și **memmove**
 - antet: **void* memcpy(void *d, const void* s, int n);**

```
exempluMemcpy.c ✘
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 int main()
6 {
7     int a[] = {25, -36, 0, 91, 7415};
8     int *b = (int*) malloc(sizeof(a));
9     memcpy(b, a, sizeof(a));
10    int i;
11    for (i=0;i<sizeof(a)/sizeof(int);i++)
12        printf("%d ", b[i]);
13    printf("\n");
14
15    float a1[] = {2.0, 3.5, -1.2};
16    float b1[3];
17    memcpy(b1, a1, sizeof(a1));
18    for (i=0;i<sizeof(a1)/sizeof(float);i++)
19        printf("%f ", b1[i]);
20    printf("\n");
21
22    char c[50] = "Ana are mere";
23    memcpy(c+8, c, 12); puts(c);
24
25    return 0;
26 }
```

```
25 -36 0 91 7415
2.000000 3.500000 -1.200000
Ana are Ana are mere
```

Pe unele compilatoare in loc de
“Ana are Ana are mere” e posibil sa
obtinem altceva (undefined behaviour).

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);**

exempluMemmove.C

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main ()
5 {
6     char t[] = "memmove este foarte folositor.....";
7     memmove(t + 20, t + 13, 16);
8     puts(t);
9     return 0;
10 }
```

```
memmove este foarte foarte folositor.....
```

Funcții predefinite pentru manipularea blocurilor de memorie

- funcție care elimină toate aparițiile unui sir t într-un sir s

```
int main()
{
    char s[100],t[100];
    strcpy(s,"abbbccca");
    strcpy(t,"bc");
    eliminaAparitii(s,t);
    printf("%s\n",s);
    return 0;
}
```

Trebuie sa obtin "abbcca"

Functii predefinite pentru manipularea blocurilor de memorie

- funcție care elimină toate aparițiile unei siruri într-un sir s

```
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 }
```

aa
n

Nu obțin ceea ce trebuie, unde am greșit?

Funcții predefinite pentru manipularea blocurilor de memorie

- funcție care elimină toate aparițiile unei siruri într-un sir s

exempluMemmove2.c

```
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 }
```

abbcca

Daca pun $p = strstr(p+1, t)$
ce se intampla?

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 sirul d.

```
exempluMemset.c   
1 #include <stdio.h>  
2 #include <string.h>  
3  
4 int main ()  
5 {  
6     char t[] = "nu prea vrem sa vina vacanta!!!";  
7     memset(t, '-', 8);  
8     puts(t);  
9     return 0;  
10 }
```

-----vrem sa vina vacanta!!!

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[] = "nu prea vrem sa vina vacanta!!!";
7     char *p = memchr(t, 'm', 25);
8     puts(p);
9
10    return 0;
11 }
```

// C:\...\exempluMemchr
m sa vina vacanta!!!

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 mic decât 0 dacă **s1 >sub>L</sub> s2**

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);`
 - comparați primii **n** octeți corespondenți începând de la adresele **s1** și **s2**.

exempluMemcmp.c

```
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 }
```

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

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

```
lungime sir = 14
lungime sir = 14
```

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

Copierea unui sir în alt sir

copiereSir.C

```
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 char* copiazaSir2(char *d, char* s)
12 {
13     int i;
14     while (*d = *s)
15     {
16         d++;
17         s++;
18     }
19     return d;
20 }
21
22
23 int main()
24 {
25     char s[1000], t[1000];
26     copiazaSir1(s,"Azi e miercuri");
27     copiazaSir1(t,s);
28     printf("Sirul t este: %s \n",t);
29     copiazaSir2(t,s);
30     printf("Sirul t este: %s \n",t);
31     return 0;
32 }
```

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

Compararea lexicografică a două şiruri

comparareSiruri.C

```
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 }
32 }
```

F / CUISS

3 3

-

Tablouri de siruri de caractere

- **un sir de caractere (string)** este un tablou unidimensional cu elemente de tip char terminat cu caracterul '\0' (NUL)
- se poate reprezenta ca:
 - tablou de caractere:
 - `char sir1[10] = "exemplu"; //se aloca 10 octeti`
 - pointer la caractere:
 - `char *sir4; //se aloca memorie numai pentru pointer`
- **un tablou de siruri** = **un tablou cu elemente siruri de caractere**
- se poate reprezenta ca:
 - tablou de tablouri de caractere = tablou bidimensional
 - `char nume[4][20];`
 - tablou de pointeri la caractere
 - `char* nume[4];`

Tablouri de şiruri de caractere

```
tablouSiruri.C 
```

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main()
5 {
6     char nume[4][10] = {"Miruna", "Claudiu", "Ionut", "Paul"};
7     int i;
8     for(i=0;i<4;i++)
9         printf("%s \n", nume[i]);
10
11     printf("\n");
12
13     nume[3][0] = 'R';
14     for(i=0;i<4;i++)
15         printf("%s \n", nume[i]);
16
17     printf("\n dimensiunea lui nume este: %d \n", sizeof(nume));
18
19     return 0;
20 }
```

Miruna
Claudiu
Ionut
Paul

Miruna
Claudiu
Ionut
Raul

dimensiunea lui nume este: 40

Tablouri de şiruri de caractere

```
tablouSiruri2.C
```

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main()
5 {
6     char* nume[4] = {"Miruna", "Claudiu", "Ionut", "Paul"};
7     int i;
8     for(i=0;i<4;i++)
9         printf("%s \n",nume[i]);
10
11     printf("\n");
12
13 //nume[3][0] = 'R'; NU AM VOIE SA FAC ASTA
14
15     printf("\n dimensiunea lui nume este: %d \n",sizeof(nume));
16
17     return 0;
18 }
```

```
.....,.....
Miruna
Claudiu
Ionut
Paul
```

```
dimensiunea lui nume este: 32
```

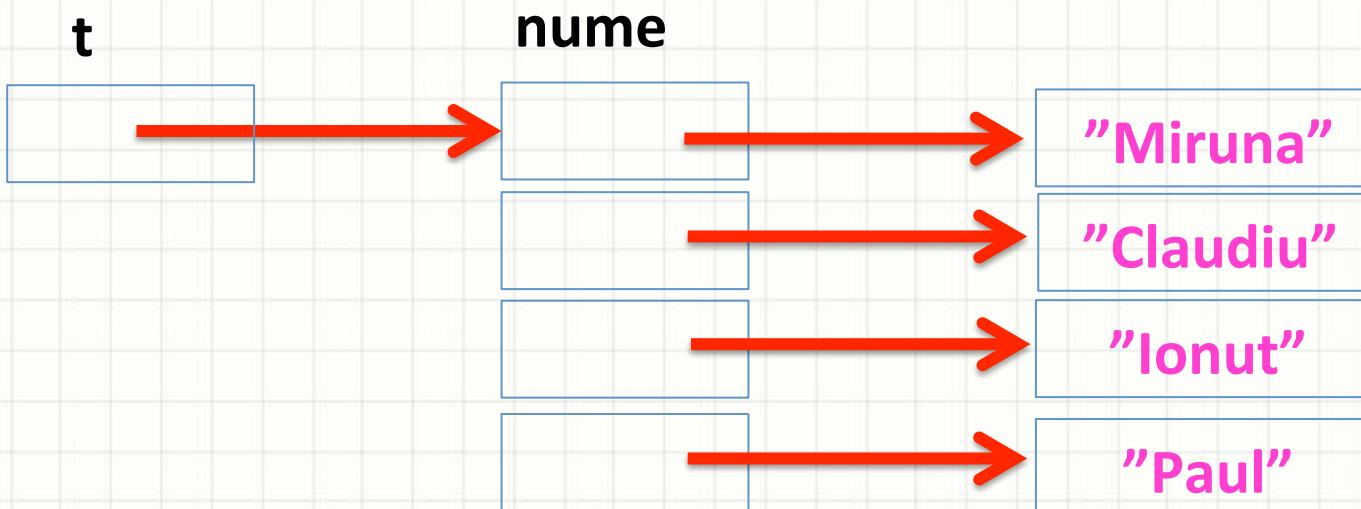
Tablouri de şiruri de caractere

```
tablouSiruri3.C ✘
1 #include <stdio.h>
2 #include <string.h>
3
4 int main()
5 {
6     char* nume[4] = {"Miruna", "Claudiu", "Ionut", "Paul"};
7     char **t = nume;
8     int i;
9     for(i=0;i<4;i++)
10         printf("%s \n", t[i]);
11
12     printf("\n");
13
14     printf("\n dimensiunea lui t este: %d \n", sizeof(t));
15
16     return 0;
17 }
```

Miruna
Claudiu
Ionut
Paul

dimensiunea lui t este: 8

Tablouri de şiruri de caractere



Tablouri de şiruri de caractere

tablouSiruri4.C 

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main()
5 {
6     char* nume[4] = {"Miruna", "Claudiu", "Ionut", "Paul"};
7     char **p = nume;
8
9     printf("Adresele pointerilor din tablou sunt:");
10    printf("%d %d %d %d \n \n", nume[0], nume[1], nume[2], nume[3]);
11    printf("Stringurile din tablou sunt:");
12    printf("%s %s %s %s \n\n", nume[0], nume[1], nume[2], nume[3]);
13
14    printf("Adresa lui nume este %d \n", nume);
15    printf("Continutul lui p este %d\n", p);
16    printf("Adresa lui nume[1] este %d \n", nume+1);
17    printf("Adresa lui nume[2] este %d \n", nume+2);
18    printf("Adresa lui nume[3] este %d \n", nume+3);
19
20
21    printf("Adresele din nume sunt:");
22    printf("%d %d %d %d \n", p[0], p[1], p[2], p[3]);
23    printf("Stringurile din nume sunt:");
24    printf("%s %s %s %s \n", p[0], p[1], p[2], p[3]);
25
26
27
28    return 0;
29 }
```

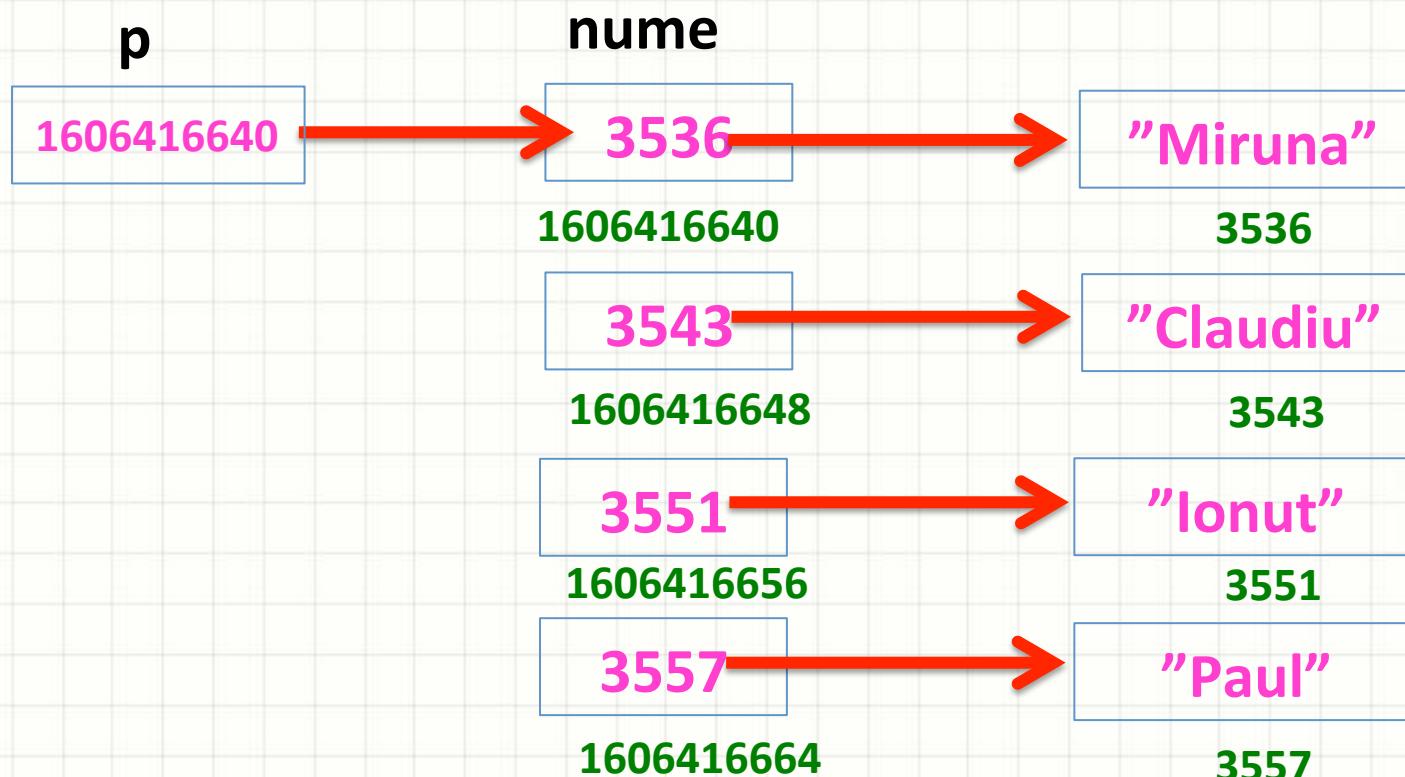
Tablouri de şiruri de caractere

tablouSiruri4.C

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main()
{
5     char* nume[4] = {"Miruna", "Claudiu", "Ionut", "Paul"};
6     char **p = nume;
7
8     printf("Adresele pointerilor din tablou sunt:");
9     printf("%d %d %d %d \n \n", nume[0], nume[1], nume[2], nume[3]);
10    printf("Stringurile din tablou sunt:");
11    printf("%s %s %s %s \n\n", nume[0], nume[1], nume[2], nume[3]);
12
13    printf("Adresa lui nume este %d \n", nume);
14    printf("Continutul lui p este %d\n", p);
15    printf("Adresa lui nume[1] este %d \n", nume+1);
16    printf("Adresa lui nume[2] este %d \n", nume+2);
17    printf("Adresa lui nume[3] este %d \n", nume+3);
18
19
20
21    printf("Adresele din nume sunt:");
22    printf("%d %d %d %d \n", p[0], p[1]Adresele pointerilor din tablou sunt:3536 3543 3551 3557
23    printf("Stringurile din nume sun
24    printf("%s %s %s %s \n", p[0], p[1]Stringurile din tablou sunt:Miruna Claudiu Ionut Paul
25
26
27
28        return 0;
29 }
```

Adresa lui nume este 1606416640
Continutul lui p este 1606416640
Adresa lui nume[1] este 1606416648
Adresa lui nume[2] este 1606416656
Adresa lui nume[3] este 1606416664
Adresele din nume sunt:3536 3543 3551 3557
Stringurile din nume sunt:Miruna Claudiu Ionut Paul

Tablouri de şiruri de caractere



Adresele pointerilor din tablou sunt: 3536 3543 3551 3557

Stringurile din tablou sunt: Miruna Claudiu Ionut Paul

Adresa lui nume este 1606416640

Continutul lui p este 1606416640

Adresa lui nume[1] este 1606416648

Adresa lui nume[2] este 1606416656

Adresa lui nume[3] este 1606416664

Adresele din nume sunt: 3536 3543 3551 3557

Stringurile din nume sunt: Miruna Claudiu Ionut Paul