Programarea calculatoarelor

FMI

Secția Calculatoare și tehnologia informației, anul I

Cursul 8 / 25.11.2024

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.
- Etapele realizării unui program C.

❑ **Fișiere text**
- Funcții specifice de manipulare.

❑ **Funcții (1)**
- Declarare și definire. Apel. Metode de trasmitere 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

0. **Clase de memorare**
1. Șiruri de caractere
2. Funcții specifice de manipulare a șirurilor de caractere
3. Manipularea blocurilor de memorie

Variabile

D.p.d.v. structural, o variabilă este un cvadruplu format din:

- nume
- set de atribute
- referință (adresă)
- valoare

Alocarea: Acțiunea prin care perechea (nume, set de atribute) se asociază cu perechea (referință, valoare) se numește ALOCAREA variabilei respective.

Variabile dinamice

- Variabilă dinamică = o variabilă a cărei alocare o comandă explicit programatorul prin mecanisme specifice limbajului, ea fiind alocată în heap, neavând nume și putând fi manipulată exclusiv prin intermediul unui pointer asociat.

- Exemplu:

int* p = (int*) malloc (sizeof (int));

Variabila dinamică NU este p, ci este *p (dereferențierea lui p)!

- Dereferențierea unui pointer. Accesul la o variabilă dinamică se face printr-o expresie formată dintr-un operand și un operator (*p), unde operandul este numele variabilei de tip pointer, iar operatorul este cel de dereferențiere.

Variabile dinamice

- Domeniul de vizibilitate a unei variabile dinamice este de fapt domeniul de vizibilitate al pointerului asociat.

- Durata de viață a unei variabile dinamice este intervalul de timp între alocarea variabilei de către programator și dealocarea ei explicită (tot de către programator) sau terminarea programului.

Variabile

Setul de atribute conține următoarele 4 elemente:

- domeniul de vizibilitate
- durata de viață
- tipul de date
- clasa de memorie

Primele 3 atribute sunt intdependente, iar al 4-lea atribut este o rezultantă a primelor 3.

Referință și valoare

Referința = informații de adresă (locul alocării)

- alocarea statică - global data segment
- alocarea dinamică - heap, stack

Valoarea = conținutul de la adresa respectivă, interpretat conform tipului de dată asociat variabilei.

Această interpretare poate fi redefinită prin utilizarea conversiilor de tip.

Prezența conversiilor de tip slăbește caracteristica de puternic-tipizare a limbajului respectiv, permițând interpretări diferite ale aceleiași zone de memorie (ex. C - mecanismul union)

Dereferențierea reprezintă extragerea valorii de la o anumită adresă.

Variabila. Set de atribute

- Domeniu de vizibilitate (scope)= intervalul de program sursă în care variabila respectivă este recunoscută și accesibilă, deci reutilizabilă. El începe de obicei imediat după declarare și se încheie la sfârșitul blocului, subrutinei sau fișierului sursă.

- Durata de viață (extent) = intervalul de timp de execuție în care zona de memorie alocată variabilei respective este utilizată pentru această variabilă.

- Tipul de date = Structuri + operații asociate (încapsulare).

- Clasa de memorie = un atribut rezultat al primelor 3 și se referă la regimul alocării variabilelor.

Clasa de memorare

- un atribut care exprimă regimul alocării variabilei. Mai precis, se referă la momentul alocării (compilare/ execuție) și la locul alocării (global data segment/ stack/ heap)
- un atribut implicit determinat de către programul translator la nivelul fiecărui limbaj. Unele limbaje de programare oferă posibilitatea exprimării explicite a clasei de memorie.
- Clasa de memorie a unei variabile defineşte:
  - timpul de viaţă,
  - domeniul de vizibilitate şi
  - locul de stocare al acesteia.

Clasele de memorare din C/ C++:

- **auto** - clasa implicită. Memoria este alocată automat la activarea unei funcții în zona de stivă și este eliberată automat la terminarea funcției. Variabilele locale unui bloc sau funcții și argumentele formale sunt automatice/ din clasa auto.
- **register** - încearcă alocarea variabilei într-un registru al procesorului și nu locații de memorie (pentru eficiență=timp de acces mai bun)
- **static** - alocare statică a variabilelor în segmentul global de date (poate fi utilizat ca mecanism de blocare a exportului de simboluri (variabile sau funcții) la nivelul unui modul)
- **extern** - mecanismul de import simboluri în cadrul compilării separate

Variabile automatice

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

int main()
{
    auto a; double b=65.7;
    a=12;
    printf("a= %d b=%lf\n", a, b);
    {
        int x=123; char y;
        y='r';
        printf("a= %d b=%.2lf\n", a, b);
        printf("x= %d y=%c\n", x, y);
    }
    printf("a= %d b=%.2lf\n", a, b);
    //x si y sunt invizibili aici
    return 0;
}
```

```
a= 12 b=65.700000
a= 12 b=65.70
x= 123 y=r
a= 12 b=65.70
```

Variabile externe

- sunt cele declarate în afara oricărei funcţii fără nici un specificator de stocare sau cele redeclarate cu specificatorul extern,
- sunt variabile cu vizibilitate globală, fiind vizibile din orice punct al fişierului sursă aflat după locul declaraţiei şi au timpul de viaţă nelimitat.
- sunt alocate la intrarea în execuţie a programului într-o zonă de memorie special destinată lor şi sunt iniţializate în mod implicit cu zero.
- Pentru a fi vizibile şi din funcţii definite înaintea declaraţiei lor, variabilele externe pot fi anunţate cu specificatorul extern:

Variabile externe

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

int main()
{
    int n; float pi=3.14;
    printf("n=");scanf("%d",&n);
    printf("%f",pi*n*n);
    return 0;
}
extern float pi;

```

Variabile statice

- sunt declarate cu specificatorul **static**
- sunt alocate la intrarea în execuţie a programului în zona de memorie a variabilelor externe, şi la fel ca acestea, au tipul de viaţă cât al programului şi sunt iniţializate implicit cu zero.
- diferă de variabilele externe numai prin restrângerea vizibilităţii lor:
  - domeniul de vizibilitate al unei variabile statice globale (declarată în exteriorul funcţiilor) nu poate depăşi fisierul sursă, iar
  - domeniul de vizibilitate al unei variabile statice locale (declarată în interiorul unei funcţii) este restrâns la blocul în care este declarată.
- variabilele statice locale îşi conservă valoarea între apelurile succesive al funcţiei.
- utilizarea variabilelor statice locale în locul variabilelor globale face posibilă definirea unei “stări interne” care se conservă între apelurile succesive ale unei funcţii.

Variabile statice

Exemplu: Implementarea unui generator standard de numere pseudo-aleatoare bazat pe congruenţa liniară $x_{n+1}=a*x_n+c$ modulo y, cu a = 10, c = 1, $y=2^6-1$ și x=13.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  double nextTerm(){
5      static const double modulo=63e0;
6      static const unsigned a=10u;
7      static const unsigned c=2u;
8      static unsigned x=13;
9      x=a*x+c;
10     return x/modulo;
11 }
12 int main()
13 {
14     for(int i=0;i<20;i++)
15         printf("%2u %16.3f\n", i, nextTerm());
16     return 0;
17 }
18
```

Variabila x a fost declarată statică pentru a-şi păstra valoarea între apeluri.
Cele trei constante au fost declarate statice pentru a nu fi reiniţializate în mod inutil la fiecare apel şi a mări astfel durata de execuţie.

Variabile statice vs automatice

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int x=1;
6      for(int i=0;i<5;i++)
7      {
8          auto a=78;
9          static b;
10         printf("a= %d b=%d x=%d i=%d\n",a,b,x,i);
11         a++;b++;x++;
12     }
13     return 0;
14 }
15
```

```
a=78 b=0 x=1 i=0
a=78 b=1 x=2 i=1
a=78 b=2 x=3 i=2
a=78 b=3 x=4 i=3
a=78 b=4 x=5 i=4
```

Variabila automatică la intrarea în bloc se inițializează, iar la ieșirea din bloc își pierde valoarea.
Variabila statică se inițializează o singură dată, implicit cu 0 și își păstrează valoarea de la o secvență la alta.

Variabile

- Variabilele statice pot fi inițializate numai cu valori constante (pentru că se face la compilare).
- Variabilele auto pot fi inițializate cu rezultatul unor expresii (pentru că se face la execuție).
- Toate valorile externe (și statice) sunt automat inițializate cu valori zero (inclusiv vectorii).

Variabile dinamice

- Memoria se alocă la execuție în zona heap alocată programului, dar numai la cererea explicită, prin apelarea unor funcții de bibliotecă (malloc, calloc, realloc).
- Memoria este eliberată la cerere, prin apelarea funcției free.
- Variabilele dinamice nu au nume, deci nu se pune problema clasei de memorare (atribut al variabilelor cu nume)

Cuprinsul cursului de azi

0. Clase de memorare
1. **Șiruri de caractere**
2. Funcții specifice de manipulare a șirurilor de caractere
3. Manipularea blocurilor de memorie

Șiruri de caractere

- **un șir de caractere (string)** este un tablou unidimensional cu elemente de tip char terminat cu caracterul ‘\0’ (NULL)
- 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 șir 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] = “exemplu”; //se aloca 10 octeti
    - char sir3[] = “exemplu”; //se aloca 8 octeti
  - pointer la caractere:
    - char *sir4; //se aloca memorie numai pentru pointer
    - char *sir5 = “exemplu”; //se aloca 8 octeti (se adauga ‘\0’ la final)

Citirea și afișarea șirurilor de caractere

- **citire:**
  - funcția **scanf** cu modelatorul de format %s:
    - *atenție:* dacă inputul este un șir 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 șirurilor de caractere

- exemplu

```
4  int main() {
5      char sir1[] = {'r','a','t','o','n','\0'};
6      char sir2[] = "raton";
7      printf("%s %s\n",sir1,sir2);
8
9      char *sir3 = sir1; sir3[0] = 'b';
10     printf("%s %s %s\n",sir1,sir2,sir3);
11
12     char sir4[100] = "raton";
13     printf("s\n",sir4);
14     for(int i=0;i<10;i++)
15         printf("%c %d\n",sir4[i],sir4[i]);
16     sir4[5] = 'i'; printf("%s\n",sir4);
17
18     char sir5[10] = "raton";
19     sir5[4] = 0; printf("%s\n",sir5);
20     sir5[3] = '\0'; printf("%s\n",sir5);
21
22     return 0;
23 }
```

Citirea și afișarea șirurilor de caractere

- exemplu

```c
int main() {
    char sir1[] = {'r','a','t','o','n','\0'};
    char sir2[] = "raton";
    printf("%s %s\n",sir1,sir2);

    char *sir3 = sir1; sir3[0] = 'b';
    printf("%s %s %s\n",sir1,sir2,sir3);

    char sir4[100] = "raton";
    printf("s\n",sir4);
    for(int i=0;i<10;i++)
        printf("%c %d\n",sir4[i],sir4[i]);
    sir4[5] = 'i'; printf("%s\n",sir4);

    char sir5[10] = "raton";
    sir5[4] = 0; printf("%s\n",sir5);
    sir5[3] = '\0'; printf("%s\n",sir5);

    return 0;
}
```

```
raton raton
baton raton baton
s
r 114
a 97
t 116
o 111
n 110
 0
 0
 0
 0
 0
ratoni
rato
rat
```

Cuprinsul cursului de azi

1. Clase de memorare
2. Șiruri de caractere
3. **Funcții specifice de manipulare a șirurilor de caractere**
4. Manipularea blocurilor de memorie

Funcții predefinite pentru manipularea șirurilor de caractere

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

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

int main() {
    char sir[100] = "test";
    printf("%d \n", sizeof(sir));
    printf("%d \n", strlen(sir));
    return 0;
}
```

Output

```
100
4
```

Funcții predefinite pentru manipularea șirurilor de caractere

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

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

int main() {
    char sir[100] = "test";
    printf("%d \n", sizeof(sir));
    printf("%d \n", strlen(sir));
    int i;
    for(i=0;i<strlen(sir);i++)
        printf("%s \n", sir+i);
    return 0;
}
```

```
100
4
test
est
st
t
```

Funcții predefinite pentru manipularea șirurilor de caractere

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

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

int main()
{
    char sir6[10]="raton";
    char *sir7="baton";
    printf("adresa lui sir6 este %d\n", sir6);
    printf("adresa lui sir7 este %d\n", sir7);

    sir7=sir6;
    puts(sir7);

    printf("adresa lui sir6 este %d\n", sir6);
    printf("adresa lui sir7 este %d\n", sir7);

    return 0;
}
```

```
adresa lui sir6 este 1748660558
adresa lui sir7 este 4202500
raton
adresa lui sir6 este 1748660558
adresa lui sir7 este 1748660558
```

Funcții predefinite pentru manipularea șirurilor de caractere

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

```
 4  int main()
 5  {
 6      char sir6[10]="raton";
 7      char sir8[20]="baton";
 8      printf("adresa lui sir6 este %d\n", sir6);
 9      printf("adresa lui sir8 este %d\n", sir8);
10
11      sir8=sir6;
12      puts(sir8);
13
14      printf("adresa lui sir6 este %d\n", sir6);
15      printf("adresa lui sir8 este %d\n", sir8);
16
17      return 0;
18  }
```

```
11      error: incompatible types in assignment
```

sir8 este numele unui tablou (pointer constant). Instructiunea sir8=sir6 da eroare la compilare.

Funcții predefinite pentru manipularea șirurilor de caractere

2. copierea unui șir – funcțiile **strcpy** și **strncpy**

- antet: **char* strcpy(char *d, char* s)**
  - copiază șirul sursă **s** în șirul destinație **d**;
  - returnează adresa șirului destinație
  - șirul rezultat are ‘\0’ la final

- antet: **char* strncpy(char *destinatie, char* sursa, int n)**
  - copiază primele **n** caractere din șirul sursă **s** în șirul destinație **d**;
  - returnează adresa șirului destinație
  - șirul rezultatul NU are ‘\0’ la final

Funcții predefinite pentru manipularea șirurilor de caractere

- **copierea unui șir** – > funcțiile **strcpy** și **strncpy**

```c
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);
    printf("%s\n", s);

    return 0;
}
```

```
tesmplu
tesm
tesm
```

Funcții predefinite pentru manipularea șirurilor de caractere

- **copierea unui șir** – > funcția **strcpy**
  - antet: **char* strcpy(char *d, char* s);**
  - presupune că șirurile destinație **d** și sursa **s** nu se suprapun
    - dacă cele două șiruri se suprapun funcția prezintă *undefined behaviour* (comportament nedefinit)

```
 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
eexemmlu
```

Soluție: funcțiile memcpy, memmove

Funcții predefinite pentru manipularea șirurilor de caractere

3. Compararea șirurilor – funcțiile **strcmp** și **strncmp**

- antet: **int strcmp(const char *s1, const char* s2);**
  - compară lexicografic șirurile **s1** și **s2**;
  - returnează: <0 dacă s1 $<_L$ s2
    0 dacă s1 $=_L$ s2
    >0 dacă s1 $>_L$ s2

- antet: **int strncmp(const char *s1, const char* s2, int n);**
  - compară lexicografic șirurile **s1** și **s2** trunchiate la lungimea **n**

- ambele funcții sunt case sensitive
  - Ex. strcmp(“POPA”,”Popa”) returnează un număr < 0 întrucât ‘O’ < ‘o’ (codurile ASCII 79 respectiv 111)
- unele implementări au funcția **stricmp** – **case insensitive**

Funcții predefinite pentru manipularea șirurilor de caractere

- **compararea șirurilor** – funcțiile **strcmp** și **strncmp**

```
4  int main() {
5      char s1[20] = "Nor", *s2 = "Noiembrie", s3[10], s4[20];
6      int c = strcmp(s1,s2);
7      printf("c=%d \n", c);
8      c>0?printf("%s > %s",s1,s2):((c==0)?printf("%s =%s",s1,s2): printf
           ("%s< %s",s1,s2));
9      printf("\n");
10
11     c = strncmp(s1,s2,2);
12     printf("c=%d \n", c);
13     char *s23=strcpy(s3,s2);
14     printf("Sirul s3 este = %s \n", s3);
15     printf("Sirul s23 este = %s \n", s23);
16     printf("Sirul s23 pointeaza catre adresa %d \n", s23);
17     printf("Adresa lui s3 este = %d \n", s3);
18
19     strncpy(s4,s2,4);
20     printf("Primele 4 litere in sirul copiat s4 = %s \n", s4);
21     return 0;
22 }
```

Funcții predefinite pentru manipularea șirurilor de caractere

- **compararea șirurilor** – funcțiile **strcmp** și **strncmp**

```
4  int main() {
5      char s1[20] = "Nor", *s2 = "Noiembrie", s3[10], s4[20];
6      int c = strcmp(s1,s2);
7      printf("c=%d \n", c);
8      c>0?printf("%s > %s",s1,s2):((c==0)?printf("%s =%s",s1,s2): printf
           ("%s< %s",s1,s2));
9      printf("\n");
10
11     c = strncmp(s1,s2,2);
12     printf("c=%d \n", c);
13     char *s23=strcpy(s3,s2);
14     printf("Sirul s3 este = %s \n", s
15     printf("Sirul s23 este = %s \n",
16     printf("Sirul s23 pointeaza catre
17     printf("Adresa lui s3 este = %d \
18
19     strncpy(s4,s2,4);
20     printf("Primele 4 litere in sirul copiat s4 = %s \n", s4);
21     return 0;
22 }
```

```
c=9
Nor > Noiembrie
c=0
Sirul s3 este = Noiembrie
Sirul s23 este = Noiembrie
Sirul s23 pointeaza catre adresa 72695238
Adresa lui s3 este = 72695238
Primele 4 litere in sirul copiat s4 = Noie
```

Funcții predefinite pentru manipularea șirurilor de caractere

4. concatenarea șirurilor – funcțiile **strcat** și **strncat**

- antet: **char* strcat(char *d, const char* s);**
  - concatenează șirul sursă **s** la șirul destinație **d**
  - returnează adresa șirului destinație
  - șirul rezultat are ‘\0’ la final
  - *condiție:* șirurile 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 șirul **s** la șirul **d**
  - returnează adresa șirului destinație d
  - șirul rezultat NU are ‘\0’ la final

Funcții predefinite pentru manipularea șirurilor de caractere

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

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

int main() {
    char s[100] = "test";
    char t[10]= "joi";

    char *p=strncat(s,t,2);
    printf("%s \n", s);
    printf("%s \n", p);

    strcat(s,t);
    printf("%s \n", s);
    return 0;
}
```

```
testjo
testjo
testjojoi
```

Funcții predefinite pentru manipularea șirurilor de caractere

4. căutarea unui caracter într-un șir – funcțiile **strchr** și **strrchr**

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

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

Funcții predefinite pentru manipularea șirurilor de caractere

- **căutarea unui caracter într-un șir – strchr și strrchr**

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

int main() {
    char s[] = "exemplu";
    char litere[]= {'e','f','g'};
    char *p;
    for(int i=0;i<3;i++)
        if(p=strchr(s,litere[i]))
        {
            printf("%s \n", strchr(s,litere[i]));
            printf("%s \n", strrchr(s,litere[i]));
        }
        else
            printf("litera %c nu se afla in sirul %s \n",litere[i],s);
    return 0;
}
```

```
exemplu
emplu
litera f nu se afla in sirul exemplu
litera g nu se afla in sirul exemplu
```

Funcții predefinite pentru manipularea șirurilor de caractere

5. căutarea unui șir în alt șir – funcția **strstr**

- antet: **char* strstr(const char *s, const char *t);**
  - caută șirul **t** în șirul **s** și întoarce un pointer la prima sa apariție
  - căutare de la stânga la dreapta
  - dacă nu apare șirul **t** în șirul **s**, returnează NULL
- exemplu: să se numere de câte ori apare un șir t într-un șir s.

Funcții predefinite pentru manipularea șirurilor de caractere

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

```
4  int main() {
5      char s[1000], t[1000];
6      printf("Sirul s este: ");scanf("%s",s);
7      printf("Sirul t este: ");scanf("%s",t);
8
9      int nrAparitii = 0;
10     char *p;
11     p = strstr(s,t);
12
13     while(p)
14     {
15         nrAparitii++;
16         p=strstr(p+1,t);
17     }
18     printf("nrAparitii = %d \n", nrAparitii);
19
20     return 0;
21 }
```

```
Sirul s este: abracadabra
Sirul t este: ab
nrAparitii = 2
```

```
Sirul s este: aaaaa
Sirul t este: aa
nrAparitii = 4
```

Funcții predefinite pentru manipularea șirurilor de caractere

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

```
4  int main() {
5      char s[1000], t[1000];
6      printf("Sirul s este: ");scanf("%s",s);
7      printf("Sirul t este: ");scanf("%s",t);
8
9      int nrAparitii = 0;
10     char *p;
11     p = strstr(s,t);
12
13     while(p)
14     {
15         nrAparitii++;
16         p=strstr(p+strlen(t),t);
17     }
18     printf("nrAparitii = %d \n", nrAparitii);
19
20     return 0;
21 }
```

```
Sirul s este: aaaaa
Sirul t este: aa
nrAparitii = 2
```

Funcții predefinite pentru manipularea șirurilor de caractere

6. împărțirea unui șir în subșiruri – funcția **strtok**

- antet: **char* strtok(char *s, const char *sep);**
  - împarte șirul **s** în subșiruri conform separatorilor din șirul **sep**
  - s = “Ana ; are . mere!!”, sep = “ ,.!?” -> **Ana are mere**
  - string-ul inițial se trimite doar la primul apel al funcției, obținându-se primul subșir
  - la următoarele apeluri, pentru obținerea celorlate subșiruri se trimite ca prim argument NULL

- exemplu: să se numere cuvintele dintr-o frază

Funcții predefinite pentru manipularea șirurilor de caractere

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

```c
int main() {
    char s[1000];
    printf("Sirul s este: ");fgets(s,1000,stdin);

    int nrCuvinte = 0;
    char *p;
    char separatori[] = "() ,.!?;:\n";

    p=strtok(s,separatori);
    while(p)
    {
        printf("%s \n",p);
        nrCuvinte++;
        p=strtok(NULL,separatori);
    }
    printf("nrCuvinte = %d \n", nrCuvinte);

    return 0;
}
```

Funcții predefinite pentru manipularea șirurilor de caractere

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

```c
int main() {
    char s[1000];
    printf("Sirul s este: ");fgets(s,1000,stdin);

    int nrCuvinte = 0;
    char *p;
    char separatori[] = "() ,.!?;:\n";

    p=strtok(s,separatori);
    while(p)
    {
        printf("%s \n",p);
        nrCuvinte++;
        p=strtok(NULL,separatori);
    }
    printf("nrCuvinte = %d \n", nrCuvinte);

    return 0;
}
```

```
Sirul s este: Ana are 3 mere. Bogdan are un mar.
    Dana are mere?
Ana
are
3
mere
Bogdan
are
un
mar
Dana
are
mere
nrCuvinte = 11
```

Funcții predefinite pentru manipularea șirurilor de caractere

6. conversia de la un șir la un număr și invers – funcțiile **sscanf** și **sprintfs** și descriptori de format potriviți:

- **conversia de la șir la un număr :** funcția **scanf**
- exemplu:

```
char *string="-45.8614";
double numar;
sscanf(string, "%lf", &numar);
printf("%f", numar);
```

- **conversia de la un număr la șir** : funcția **sprintf**
- exemplu:

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

Funcții predefinite pentru manipularea șirurilor de caractere

7. funcții de **clasificare a caracterelor** (*nu a șirurilor de caractere*)
   - sunt în fișierul **ctype.h**

Prototip	Descriere
int isdigit(int c)	Returnează **true** dacă **c** este cifră și **false** altfel
int isalpha(int c)	Returnează **true** dacă **c** este literă și **false** altfel
int islower(int c)	Returnează **true** dacă **c** este literă mică și **false** altfel
int isupper(int c)	Returnează **true** dacă **c** este literă mare și **false** altfel
int tolower(int c)	Dacă **c** este literă mare, **tolower** returnează **c** ca și literă mică. Altfel, **tolower** returnează argumentul nemodificat
int toupper(int c)	Dacă **c** este literă mică, **toupper** returnează **c** ca și literă mare. Altfel, **toupper** returnează argumentul nemodificat
int isspace(int c)	Returnează **true** dacă **c** este un caracter *white-space* — *newline* (**'\n'**), *space* (**' '**), *form feed* (**'\f'**), *carriage return* (**'\r'**), *horizontal tab* (**'\t'**), *vertical tab* (**'\v'**) — și **false** altfel

Funcții predefinite pentru manipularea șirurilor de caractere

- funcții de clasificare a caracterelor (nu a șirurilor de caractere)
- sunt în fișierul **ctype.h**

```
char *t="9 Portocale\n3 Pere";
printf("%s\n",t);
printf("%d\n",isdigit(t[0]));
printf("%d\n",isalpha(t[1]));
printf("%d\n",islower(t[2]));
printf("%d\n",isupper(t[2]));
printf("%d\n",isspace(t[11]));
printf("%d\n",isspace(t[1]));
printf("%c\n",tolower(t[2]));
printf("%c\n",toupper(t[4]));
puts(t);
```

Rezultate afișate:

```
9 Portocale
3 Pere
1
0
0
1
8
8
p
R
9 Portocale
3 Pere
```

Cuprinsul cursului de azi

1. Clase de memorare
2. Șiruri de caractere
3. Funcții specifice de manipulare a șirurilor de caractere
4. **Manipularea blocurilor de memorie**

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 constanți;
  - 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 șirul sursă **s** în șirul destinație **d**;
    - returnează adresa șirului destinație
    - șirul rezultat are un '\0' la final
  - **char* strncpy(char *d, char* s, int n);**
    - copiază primele **n** caractere șirul sursă **s** în șirul destinație **d**;
    - returnează adresa șirului destinație
    - șirul 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 constanți;
  - copierea se face element cu element folosind instrucțiuni repetitive (for, while);
- pe cazul general (a și b nu sunt neaparat 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 șirurile destinație și sursa nu se suprapun
      - dacă cele două șiruri se suprapun funcția prezintă *undefined behaviour* (comportament nedefinit)

Funcții predefinite pentru manipularea blocurilor de memorie

1. copierea unui tablou – funcția **memcpy**
   - antet: **void* memcpy(void *d, const void* s, int n);**

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

```
25 -36 0 91 7415
2.00 3.50 -1.20
Ana are Ana are mere
```

Funcții predefinite pentru manipularea blocurilor de memorie

2. 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 șirurile destinație **d** și sursă **s** se suprapun
  - folosește un buffer intern pentru copiere

Funcții predefinite pentru manipularea blocurilor de memorie

2. copierea unui tablou – funcția **memmove**

- antet: **void* memmove(void *d, const void* s, int n);**

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

int main() {
    char t[] = "memmove este foarte folositor!.........";
    printf("%d\n",strlen(t));

    memmove(t+20,t+13,18);
    puts(t);
    printf("%d\n",strlen(t));

    return 0;
}
```

```
40
memmove este foarte foarte folositor!...
40
```

Funcții predefinite pentru manipularea blocurilor de memorie

- funcție care elimină toate aparițiile unui șir **t** într-un șir **s**

```
12 int main() {
13     char s[100],t[100];
14     strcpy(s,"abbbccca");
15     strcpy(t,"bc");
16     eliminaAparitii(s,t);
17     printf("%s\n",s);
18     return 0;
19 }
```

abb**bc**cca
Trebuie sa obțin “abbcca”

FUNCȚII PREDEFINITE PENTRU MANIPULAREA BLOCURILOR DE MEMORIE

- funcție care elimină toate aparițiile unui șir **t** într-un șir **s**

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

FUNCȚII PREDEFINITE PENTRU MANIPULAREA BLOCURILOR DE MEMORIE

- funcție care elimină toate aparițiile unui șir **t** într-un șir **s**

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

```
aa
```

Nu obțin ceea ce trebuie.
Unde am greșit?

FUNCȚII PREDEFINITE PENTRU MANIPULAREA BLOCURILOR DE MEMORIE

- funcție care elimină toate aparițiile unui șir **t** într-un șir **s**

```
4  void eliminaAparitii(char *s,char* t){
5      char *p = strstr(s,t);
6      while(p){
7          memmove(p,p+strlen(t),s+strlen(s)-(p+strlen(t))+1);
8          p =strstr(p,t);
9      }
10 }
11
12 int main() {
13     char s[100],t[100];
14     strcpy(s,"abbbccca");
15     strcpy(t,"bc");
16     eliminaAparitii(s,t);
17     printf("%s\n",s);
18     return 0;
19 }
```

```
abbcca
```

Funcții predefinite pentru manipularea blocurilor de memorie

3. 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** poziţii (octeți) la valoarea dată de **val**
  - Funcţia returnează şirul **d**

Funcții predefinite pentru manipularea blocurilor de memorie

- **setarea unor octeți la o valoare** – funcția **memset**
  - antet: **void* memset(void *d, char val, int n);**

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

int main() {
    char s[] = "Nu vrem sa vina vacanta!";
    memset(s,'-',8);
    puts(s);

    return 0;
}
```

```
--------sa vina vacanta!
```

Funcții predefinite pentru manipularea blocurilor de memorie

4. căutarea unui octet într-un tablou – funcția **memchr**

- antet: **void* memchr(const void *d, char c, int n)**
- detemină prima apariție a octetului **c** în zona de memorie dată de pointerul **d** și care conține **n** octeți.
- Funcţia returnează pointerul la prima apariție a lui c în d sau NULL, dacă c nu se găsește în d.

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);**

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

int main() {
    char s[] = "Nu vrem sa vina vacanta!";
    char *p = memchr(s,'v',strlen(s));
    puts(p);

    return 0;
}
```

```
vrem sa vina vacanta!
```

Funcții predefinite pentru manipularea blocurilor de memorie

5. compararea a două tablouri pe octeți – funcția **memcmp**

- antet: **int memcmp(const void *s1, const void * s2, int n)**
  - compară primii **n** octeți corespondenți începând de la adresele **s1** și **s2**.
  - Returnează:
    - 0 dacă octeții sunt identici
    - ceva mai mic decât 0 dacă **s1 <$_L$ s2**
    - ceva mai mic decât 0 dacă **s1 >$_L$ 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);

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

```
0
-3
0
1
```

Funcții predefinite pentru manipularea blocurilor de memorie

Funcția	Descriere
void* memcpy (void *dest, void *src, unsigned cnt)	Funcția copiază cnt octeți din zona de memorie src în dest (src și dest trebuie să fie disjuncte) și returnează prin numele său adresa destinație dest.
void* memmove (void *dest, void *src, unsigned cnt)	Funcția copiază cnt octeți din zona de memorie src în dest (nu neapărat disjuncte)și returnează prin numele său adresa sursă src.
void* memchr (void *src, int c, unsigned cnt)	Funcția caută valoarea c în primii cnt octeți din zona de memorie src și returnează prin numele său adresa octetului c sau NULL dacă c nu a fost găsit.
void* memset (void *dest, int c, unsigned cnt)	Funcția scrie valoarea c în primii cnt octeți din zona de memorie dest și returnează prin numele său adresa destinație dest.
int memcmp (void *src1, void *src2, unsigned cnt)	Funcția compară în ordine cel mult cnt octeți din zonele de memorie src1 și src2. Funcția returnează prin numele său valoarea întreagă 0 dacă informația din src1 este identică cu cea din src2; valoarea întreagă -1 dacă primul octet diferit din src1 este mai mic decât octetul corespunzător din src2; valoarea întreagă 1 dacă primul octet diferit din src1 este mai mare decât octetul corespunzător din src2

Cursul 8

0. Clase de memorare
1. Șiruri de caractere
2. Funcții specifice de manipulare
3. Funcții pentru manipulare blocuri de memorie

Cursul 9

1. Fișiere binare