Programarea Calculatoarelor

Cursul 9: Șiruri de caractere (*String*-uri)

Razvan Itu
Universitatea Tehnică din Cluj-Napoca
Departamentul Calculatoare

Șiruri de caractere (*String*-uri)

- Un șir de caractere (*string*) este memorat într-un tablou unidimensional de tip **`char`**
- Ultimul caracter din tablou este caracterul NUL (**`'\0'`**)
- Numele șirului de caractere este pointer constant la primul caracter din șir
- Exemplu:

```
char s[]="Sir de caractere";
```

În memorie se stochează pe fiecare poziție din șir codul ASCII corespunzător caracterului respectiv

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Cod ASCII (hexazecimal)	53	69	72	20	64	65	20	63	61	72	61	63	74	65	72	65	00

Șiruri de caractere (*String*-uri)

- În exemplul anterior
  - `s[i]`, cu i=0…15 reprezintă codul ASCII pentru cel de-al *i*-lea caracter din șirul de caractere
  - `s+i`, cu i=0…15 reprezintă pointer la cel (adresa celui) de-al *i*-lea caracter din șirul de caractere
  - `*(s+i)`, cu i=0…15 reprezintă codul ASCII pentru cel de-al *i*-lea caracter din șirul de caractere
  - `s[1]='u'` modifică al doilea caracter din șir, acesta devenind `"Sur de caractere"`
- Afișarea șirului `s` de caractere

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

echivalentă cu

```
puts(s);
```

Șiruri de caractere (*String*-uri)

- Pointer la un șir de caractere constant
  - Constantele sunt alocate într-o zonă specială de memorie protejată
  - Exemplu

    ```
    char *p="Sir de caractere";
    ```

    `p[1]='u';` nu este permis și apare o eroare la execuția programului

- În exemplele anterioare
  - `sizeof(s)` este 17 întrucât sunt alocați 16 octeți pentru caracterele stocate (1 octet/caracter) și încă unul pentru caracterul terminal `'\0'`
  - `sizeof(p)` este 4 întrucât reprezintă un pointer la primul caracter din șirul constant respectiv
    - Orice pointer este stocat pe 4 octeți (program pe 32 de biți!)

Șiruri de șiruri de caractere

- Tablouri care conțin *string*-uri
  - `char a[][20]={"ala", "bala", "portocala"};`
  - Alocă un tablou cu 3 șiruri de caractere de lungime 20
  - `sizeof(a)` este 60
  - Schimbarea conținutului string-urilor stocate este permisă:
    - `a[1][1]='i'` face ca `"bala"` să fie schimbat în `"bila"`
- Tablouri cu *string*-uri constante
  - `char* ap[]={"ala", "bala", "portocala"};`
  - Alocă un tablou cu 3 pointeri la primul caracter din șiruri de caractere constante
  - `sizeof(ap)` este 12 (3 pointeri a câte 4 octeți fiecare)
  - Schimbarea conținutului string-urilor stocate nu este permisă!
    - `ap[1][1]='i'` generează o eroare în timpul execuției programului

Conversia *string*-urilor la numere și invers

- Conversia de la *string* la un număr poate fi făcută cu ajutorul funcției **`sscanf`** și descriptori de format potriviți
  - Exemplu

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

- Conversia de la un număr la *string* poate fi făcută cu ajutorul funcției **`sprintf`** și descriptori de format potriviți
  - Exemplu

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

Funcții pentru manipularea caracterelor

- Câteva prototipuri din biblioteca **ctype.h**
  - În limbajul C: **false** este 0, **true** este orice diferit de 0

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 pentru manipularea caracterelor – exemplu

```
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
```

Funcții pentru manipularea *string*-urilor

- Câteva prototipuri din biblioteca **string.h**

Prototip	Descriere
size_t strlen(const char *s)	Returnează numărul de caractere conținut de *string*-ul **s** aflate înaintea caracterului terminal **'\0'**
char *strdup(const char *s)	Copiază *string*-ul **s** într-un nou string alocat dinamic. Dacă alocarea dinamică eșuează, funcția returnează pointer la **NULL**, altfel pointer la primul caracter din noul *string* duplicat generat
char *strcpy(char *s1, const char *s2)	Copiază *string*-ul **s2** în șirul de caractere **s1**. Valoarea lui **s1** este returnată
char *strncpy(char *s1, const char *s2, size_t n)	Copiază cel mult **n** caractere din *string*-ul **s2** în șirul de caractere **s1**. Valoarea lui **s1** este returnată

Funcții pentru manipularea *string*-urilor

- Câteva prototipuri din biblioteca **string.h**

Prototip	Descriere
char *strcat(char *s1, const char *s2)	Concatenează *string*-ul **s2** la *string*-ul **s1**. Primul caracter din **s2** suprascrie caracterul terminal **'\0'** din **s1**. Valoarea lui **s1** este returnată
char *strncat(char *s1, const char *s2, size_t n)	Concatenează cel mult **n** caractere din *string*-ul **s2** la *string*-ul **s1**. Primul caracter din **s2** suprascrie caracterul terminal **'\0'** din **s1**. Valoarea lui **s1** este returnată
int strcmp(const char *s1, const char *s2)	Compară conținutul *string*-urilor **s1** și **s2** la nivelul caracterelor componente. Returnează un număr negativ dacă **s1** este mai mic decât **s2** (ordinea este dată de codurile ASCII), zero dacă sunt identice și un număr pozitiv dacă **s1** este mai mare decât **s2**
int strncmp(const char *s1, const char *s2, size_t n)	Compară conținutul primelor **n** caractere din *string*-urile **s1** și **s2**. Funcția se comportă asemenea funcției **strcmp**

Funcții pentru manipularea *string*-urilor

- Câteva prototipuri din biblioteca **string.h**

Prototip	Descriere
int stricmp(const char *s1, const char *s2)	Compară conținutul *string*-urilor **s1** și **s2** la nivelul caracterelor componente, **ignorând tipul literelor (mici/mari)**. Returnează un număr negativ dacă **s1** este mai mic decât **s2** (ordinea este dată de codurile ASCII), zero dacă sunt identice și un număr pozitiv dacă **s1** este mai mare decât **s2**
int strnicmp(const char *s1, const char *s2, size_t n)	Compară conținutul primelor **n** caractere din *string*-urile **s1** și **s2 ignorând tipul literelor (mici/mari)**. Funcția se comportă asemenea funcției **stricmp**
char *strchr(const char *s, int c)	Găsește prima apariție a caracterului **c** în *string*-ul **s** Returnează pointer către acel caracter în *string* dacă a fost găsit și **NULL** în caz contrar.

Funcții pentru manipularea *string*-urilor – exemplu

```
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);
```

Rezultate afișate:

```
4 6 30
5 5 5
Ali
Emil
```

Funcții pentru manipularea *string*-urilor – cont. exemplu

```
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));
printf("%d %d\n",
      stricmp(s,"ALONA"),
      strnicmp(s,"ALI",3));
char *fs=strchr("Liliana",'a');
puts(fs);
```

Rezultate afișate:

```
Alinmil.■    :A¥t=E¥t▒→@
Alinmil
Alinmilina
AlinmilinaAli
1 0
-1 0
ana
```

Funcție pentru delimitarea unor sub-șiruri dintr-un șir de carcatere

- Funcția **strtok**

```
char *strtok(char *sir, const char *delimitatori)
```

  - Delimitează string-ul ***sir*** în sub-*string*-uri componente delimitate de unul sau mai multi ***delimitatori***
  - *String*-ul inițial se trimite doar la primul apel al funcției, obținându-se primul sub-*string*
  - La următoarele apeluri, pentru obținerea celorlalte sub-*string*-uri, se trimite ca și prim argument **NULL**
  - *String*-ul inițial este distrus în urma acestor apeluri!
  - Exemplu:

```
char sir[100];
gets(sir); //"., Ana ;.  are,,,mere;si ;  pere..."
char *subsir=strtok(sir,"., ;");
while (subsir!=NULL) {
   puts(subsir);
   subsir=strtok(NULL,"., ;");
}
```

Funcții pentru manipularea blocurilor de memorie

- Câteva prototipuri din biblioteca **string.h**

Prototip	Descriere
void *memcpy(void *s1, const void *s2, size_t n)	Copiază simplu **n** octeți începând de la adresa **s2** la adresa **s1**. Se poate ca în timpul copierii sursa să fie suprascrisă. Returnează pointer la începutul zonei de memorie **s1**
void *memmove(void *s1, const void *s2, size_t n)	Copiază **n** octeți începând de la adresa **s2** la adresa **s1** prin intermediul unei zone de memorie temporară. Se evită astfel posibilitatea ca în timpul copierii sursa să fie suprascrisă. Returnează pointer la începutul zonei de memorie **s1**
int memcmp(const void *s1, const void *s2, size_t 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<s2**, ceva mai mare ca 0 dacă **s1>s2**

Funcții pentru manipularea blocurilor de memorie

- Câteva prototipuri din biblioteca **string.h**

Prototip	Descriere
void *memchr(const void *s, int c, size_t n)	Determină prima apariție a octetului **c** (convertit la tip **unsigned char**) în primii **n** octeți care încep de la adresa **s**. Dacă **c** este găsit se returnează pointerul la **c** în **s**, alfel se returnează **NULL**
void *memset(void *s, int c, size_t n)	Copiază valoarea **c** (convertită la tipul **unsigned char**) în primii **n** octeți începând de la adresa **s**. Returnează pointer la începutul zonei de memorie **s**

Funcții pentru manipularea blocurilor de memorie – exemplu

```
int a[]={25,-36,91,7415};
int *b=(int*)malloc(sizeof(a));
memcpy(b,a,sizeof(a));
for (int i=0;i<sizeof(a)/sizeof(int);i++)
    printf("%d ",b[i]);
printf("\n");
char c[50]="Ana are mere";
char d[50]="Ana are mere";
memcpy(c+8,c,12);
puts(c);
memmove(d+8,d,12);
puts(d);
```

Rezultate afișate (compilare pe Release):

```
25 -36 91 7415
Ana are Ana are Ana
Ana are Ana are mere
```

Funcții pentru manipularea blocurilor de memorie – cont. exemplu

```
char *p=memchr(d,'m',sizeof(d));
puts(p);
printf("%d\n",memcmp(c,d,sizeof(c)));
printf("%d\n",memcmp(c,d,3));
memset(a+1,0,8);
for (int i=0;i<sizeof(a)/sizeof(int);i++)
    printf("%d ",a[i]);
```

Rezultate afișate (compilare pe Release):

```
mere
-1
0
25 0 0 7415
```