Programarea calculatoarelor

caractere şi şiruri de caractere; funcţii de clasificare pentru caractere (ctype.h); funcţii pentru şiruri de caractere (string.h); citire şi scriere

În limbajul C caracterele individuale sunt reprezentate folosind tipul de date **char**. Fiecărui caracter i se atribuie un cod numeric (**ASCII** – American Standard Code for Information Interchange), cu ajutorul căruia se pot codifica 128 de caractere. În tabela de mai jos pentru fiecare caracter se dau codul său zecimal şi cel hexazecimal (în baza de numeraţie 16). Se poate constata că unele caractere nu sunt reprezentabile direct ci ele sunt „caractere de control" gen „spaţiu", „linie nouă", „TAB", „ENTER", etc:

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Tipul de date **char** este tratat ca un număr întreg şi astfel se pot realiza cu el orice operaţii numerice posibile numerelor întregi. Afişarea/citirea unui singur caracter se face cu placeholder-ul **%c** în *printf/scanf*. La *printf*, dacă se foloseşte %d se va afişa codul ASCII al caracterului, iar dacă se foloseşte %c se va afişa reprezentarea sa grafică (ex: o literă). Caracterele (tip de date *char*) se scriu între apostroafe:

```
char c='A';     // atribuie valoarea 'A' (codul ASCII 65) variabilei c
```

Se poate citi un caracter şi cu funcţia „**getchar**()", care returnează un caracter citit de la tastatură, sau se poate scrie cu „**putchar**(c)".

Atenţie: Între **apostroafe** se scrie doar **un singur caracter** şi tipul rezultat este **char.** Între **ghilimele** se scrie un **şir de caractere** şi tipul rezultat este **char[]** (vector de caractere).

Exemplul 10.1: Să se citească un caracter de la tastatură şi să se afişeze dacă este cifră, literă mică, literă mare sau altceva. În caz că e litera mică să se afişeze litera mare corespunzătoare şi invers.

```
#include <stdio.h>

int main()
{
  char c;
  printf("introduceti un caracter: ");
  scanf("%c", &c);
  if (c >= '0' && c <= '9')
     printf("este o cifra");
  else if (c >= 'a' && c <= 'z')
     printf("este o litera mica -> %c", c - 'a' + 'A');
  else if (c >= 'A' && c <= 'Z')
     printf("este o litera mare -> %c", c - 'A' + 'a');
  else
     printf("este altceva");
  return 0;
}
```

Pentru a se testa apartenenţa la o anumită clasă de caractere s-au folosit operaţii clasice de testare a apartenenţei la un interval, la fel ca pentru numere. Pentru transformarea din numere mici în numere mari şi invers s-au folosit tot operaţii numerice: prima oară s-a scăzut din *c* începutul intervalului din care face parte, pentru a se obţine astfel deplasamentul (offset-ul) lui *c* faţă de începutul de interval, iar apoi s-a adăugat la acest offset începutul noului interval în care vrem să-l translatăm.

Deoarece astfel de operaţii pe caractere (clasificare, transformare) sunt destul de frecvente, limbajul C oferă o bibliotecă de funcţii pe caractere accesibilă prin intermediul antetului **<ctype.h>** (*char types*), printre care enumerăm:

Funcţie	**Efect**
toupper(c)/**tolower**(c)	returnează litera mare/mică, corespunzătoare literei *c*
isdigit(c)	**true** dacă *c* este o cifră (’0’..’9’)
isalpha(c)	**true** dacă *c* este o literă, mică sau mare (’a’..’z’ sau ’A’..’Z’)
islower(c)	**true** dacă *c* este o literă mică (’a’..’z’)
isupper(c)	**true** dacă *c* este o literă mare (’A’..’Z’)

Folosind aceste funcţii, exemplul anterior se poate scrie:

```
#include <ctype.h>
#include <stdio.h>
int main()
{
  char c;
  printf("introduceti un caracter: ");
  scanf("%c", &c);
  if (isdigit(c))
     printf("este o cifra");
```

```
  else if (islower(c))
     printf("este o litera mica -> %c", toupper(c));
  else if (isupper(c))
     printf("este o litera mare -> %c", tolower(c));
  else
     printf("este altceva");
  return 0;
}
```

Observaţie: Se poate constata că tabela ASCII, fiind iniţial gândită de americani, nu conţine coduri pentru caractere naţionale, altele decât cele englezeşti (ex: româneşti, germane, chirilice, chinezeşti, greceşti, etc). Pentru reprezentarea acestor caractere se foloseşte o altă codificare numită **UNICODE**, care atribuie un cod fiecărui caracter existent în lume, inclusiv caracterelor matematice, muzicale, etc. În limbajul C aceste coduri, care nu încap în tipul **char** (1 octet) se reprezintă folosind tipul **wchar_t** („**wide char type**”, 2-4 octeţi). Funcţiile pentru ele sunt în antetul **<wctype.h>** şi au forma **iswalpha**, **iswupper**, **towupper**, etc. Pentru tipărire/citire există funcţiile **wprintf/wscanf**. În acest laborator ne vom ocupa doar de variantele ASCII ale acestor funcţii, codificate cu caractere de tip **char**.

Şiruri de caractere

Un şir de caractere (string) este reprezentat în C prin vectori cu elemente de tip *char*. **Prin convenţie, sfârşitul unui şir de caractere se consideră ca fiind caracterul cu codul 0 (’\0’).** Funcţiile C folosesc acest caracter pentru a şti când s-a ajuns la sfârşit de şir. Se poate astfel folosi un vector mai mare (ex: de 100 caractere) pentru a se ţine în el un şir de caractere de doar 9 litere.

Atenţie: a nu se confunda terminatorul de şir (’\0’, cod ASCII 0) cu cifra ’0’ (cod ASCII 48).

Folosind strict noţiunile de la vectori, putem iniţializa un şir de caractere astfel:

```
char sir1[]={‘s’,’a’,’l’,’u’,’t’,’\0’};
```

Limbajul C oferă o posibilitate mai simplă de a scrie şiruri de caractere, folosind **ghilimele**:

```
char sir1[]=”salut”;
```

Când se folosesc ghilimele, compilatorul adaugă automat terminatorul ‘\0’ la sfârşitul şirului, deci acesta nu mai trebuie specificat. Deoarece vectorii sunt în marea majoritate a cazurilor identici cu pointerii, putem rescrie iniţializarea ca fiind:

```
char *sir1=”salut”;
```

Constatăm astfel că în C, atunci când scriem caractere între ghilimele, tipul de date este ”*char[]*” (chiar dacă avem un singur caracter, ex: ”#”), iar acel şir are automat adăugat terminatorul de şir. Când scriem un caracter între apostroafe, tipul de date este “*char*” şi avem de-a face întotdeauna cu un singur caracter, chiar dacă pentru a-l scrie este necesar să folosim secvenţe escape (ex: ‘\\’).

Afişarea şirurilor de caractere se poate face cu *printf*, folosind placeholderul %s, sau cu funcţii gen *puts(s)*, care trece automat la linie nouă după ce a afişat şirul *s*.

Citirea unor şiruri de caractere se face în mai multe feluri, cu efecte diferite:

- pentru a citi un singur cuvânt, până la primul caracter cu funcţie de spaţiu (spaţiu, TAB, ENTER, etc), se foloseşte placeholderul %s în *scanf*. În acest fel putem de exemplu citi mai multe valori diferite, aflate pe aceeaşi linie.
- pentru a citi o linie întreagă, până la apăsarea tastei ENTER, se foloseşte funcţia *fgets(sir,nr_max_caractere,stdin)*, cu următoarele argumente:

- *sir* - destinaţia unde vor fi citite caracterele (un vector de caractere)
- *nr_max_caractere* - lungimea maximă a şirului care poate fi stocat în *sir*, incluzând terminatorul de şir. Întotdeauna vor fi citite maximum *nr_max_caractere-1*, pentru a rămâne loc pentru terminator, care este adăugat automat. După caracterele citite, *fgets* va pune automat în şir (dacă mai este loc) caracterul \n (newline). Astfel, în marea majoritate a cazurilor, dacă citim date cu *fgets*, trebuie să ţinem cont de faptul că la sfârşitul şirului există un \n.
- *stdin* - este o variabilă predefinită în C, care indică tastatura ca fiind sursa caracterelor citite (*standard input*). Când se va discuta despre fişiere, *stdin* va putea fi înlocuit cu un fişier, pentru a citi din acel fişier.

Observaţie: În C există şi funcţia *gets(sir)*, care într-un fel este o variantă simplificată a lui *fgets*. *gets* citeşte de la tastatură o linie, fără a ţine cont de lungimea ei şi o depune în *sir*. Această funcţie nu este recomandat a fi folosită, deoarece ea nu testează dacă şirul citit începe în vectorul destinaţie, putând astfel apărea depăşiri de vector, dacă se citesc prea multe caractere. Din acest motiv, compilatorul va emite o atenţionare la folosirea lui *gets*.

Funcţia **scanf** poate şi ea duce la depăşire de vector când este folosită cu %s şi este recomandat să se precizeze numărul maxim de caractere ce se pot citi: %20s

Exemplul 10.2: Să se scrie un program care citeşte o linie de text, îi converteşte toate caracterele la litere mari şi apoi afişează textul rezultat.

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

int main()
{
  char s[100];
  int i;
  printf("textul: ");
  fgets(s, 100, stdin);     // se citesc maxim 99 de caractere+\0; dacă sunt mai puţine, la sfarsit se va depune şi \n
  for (i = 0; s[i]; i++){      // se itereaza atata timp cat inca nu s-a ajuns la terminatorul de sir
     s[i] = toupper(s[i]);
  }
  printf("text final: %s", s);
  return 0;
}
```

Variabila *s* va conţine şirul de caractere citit. Din definirea ei se constată că în ea vor putea încăpea maximum 99 de caractere şi terminatorul. Iterarea într-un şir de caractere se face testând prezenţa terminatorului. Deoarece terminatorul are valoarea numerică 0, deci fals, când se ajunge la acesta, iterarea se opreşte, fără ca terminatorul să fie inclus în procesare. În final *i* va fi poziţionat chiar pe poziţia terminatorului. Acest aspect este important dacă vrem să numărăm caracterele din şir, dacă vrem să adăugăm noi caractere, etc.

Aplicaţia 10.1: Să se citească o linie de la tastatură. Linia conţine cuvinte care sunt formate doar din litere, cuvintele fiind despărţite prin orice alte caractere ce nu sunt litere. Să se capitalizeze prima literă din fiecare cuvânt şi să se afişeze şirul rezultat.

Exemplul 10.3: Să se scrie un program care citeşte pe rând 2 nume. Cele două nume vor fi concatenate într-un alt şir de caractere cu „ şi ” între ele şi rezultatul va fi afişat.

```
#include <stdio.h>

int main()
{
  char s1[30], s2[30];                    // şirurile de intrare
```

```
  char s[100];                              // şirul destinaţie
  char sep[] = " si ";
  int i, j;
  printf("nume1: ");
  fgets(s1, 30, stdin);
  printf("nume2: ");
  fgets(s2, 30, stdin);
  j = 0;                                    // index in destinatie
  for (i = 0; s1[i] && s1[i] != '\n'; i++){ // copiaza caracterele din s1 pana la aparitia \0 sau \n
     s[j++] = s1[i];
  }
  for (i = 0; sep[i]; i++){                 // concateneaza separatorul
     s[j++] = sep[i];
  }
  for (i = 0; s2[i] && s2[i] != '\n'; i++){ // concateneaza caracterele din s2
     s[j++] = s2[i];
  }
  s[j] = '\0';                               // adauga terminatorul
  printf("text final: %s", s);
  return 0;
}
```

Deoarece operaţiile cu şiruri de caractere sunt frecvente, în C există o bibliotecă de funcţii pe şiruri de caractere, accesibilă prin intermediul antetului **<string.h>.** Aceste funcţii în general încep cu „str” (de la **string**), urmat de un sufix care este prescurtarea unui verb ce defineşte acţiunea realizată. Printre aceste funcţii enumerăm:

Funcţie	**Efect**
strcpy(dst,src)	copiază şirul „src” peste „dst”, inclusiv terminatorul. Pentru a se ţine minte care este sursa şi care este destinaţia, se poate face comparaţia cu o atribuire, deci destinatia este în stânga: dst=src;
strcat(s1,s2)	concatenează „s2” la sfârşitul lui „s1” şi adaugă terminatorul
strlen(s)	returnează dimensiunea caracterelor şirului „s”, fără terminator. ATENŢIE! Funcţia returnează un unsigned long, ca urmare, NU COMPARAŢI NUMERE NEGATIVE cu strlen(s)!!
strcmp(s1,s2)	compară s1 cu s2 în ordine alfabetică, ţinând cont de diferenţa între litere mari şi mici. Dacă: • s1 este înaintea lui s2 alfabetic, returnează o valoare negativă (<0) • s1 este după s2 alfabetic, returnează o valoare strict pozitivă (>0) • s1 este identic cu s2, returnează 0

stricmp(s1,s2)	la fel ca **strcmp**(s1,s2), dar nu ţine cont de litere mari şi mici
strchr(s,c)	Caută caracterul c în şirul s. Dacă l-a găsit, returnează un pointer la poziţia sa, altfel returnează NULL.
strstr(sir,subsir)	Caută subşirul în şir. Dacă l-a găsit, returnează un pointer la poziţia sa, altfel returnează NULL.
strcspn(s1,s2)	returnează câte caractere există de la începutul lui *s1* până la prima apariţie în s1 a oricărui caracter din s2. De obicei s2 este o listă de separatori (spaţiu, virgulă, …) şi atunci se returnează poziţia primului separator găsit. Dacă în s1 nu a fost găsit niciun caracter din s2, se returnează dimensiunea lui s1.

Folosind aceste funcţii, exemplul de mai sus poate fi rescris astfel:

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

int main()
{
   char s1[30], s2[30];
   char s[100];
   char sep[] = " si ";
   printf("nume1: ");
   fgets(s1, 30, stdin);
   s1[strcspn(s1, "\n")] = '\0';                  // elimină posibilul \n, punand terminatorul de sir peste el
   printf("nume2: ");
   fgets(s2, 30, stdin);
   s2[strcspn(s2, "\n")] = '\0';
   strcpy(s, s1);                                 // seteaza sirul rezultat ca fiind primul nume
   strcat(s, sep);                                // concateneaza separatorul
   strcat(s, s2);                                 // concateneaza al doilea nume
   printf("text final: %s", s);
   return 0;
}
```

Dacă se va consulta documentaţia C pentru aceste funcţii, se va constata că ele sunt definite în următoarea manieră:

int **strcmp**(const char *s1, const char *s2)

const înseamnă că acel argument este constant pentru funcţie, adică funcţia nu are dreptul să-l modifice.

În general implementarea acestor funcţii este simplă şi putem la rândul nostru să scriem funcţii similare, dacă avem nevoie de ele. De exemplu, putem implementa *strcat(s1,s2)* în felul următor:

```
// s1 va fi modificat prin concatenarea la el a caracterelor din s2
// s2 va fi doar citit, deci poate fi const
char *mystrcat(char *s1,const char *s2){
    char *p=s1;
```

```
  while(*p)p++;                    // trece peste toate caracterele din s1
  // p este pozitionat pe terminatorul lui s1
  // itereaza toate caracterele din s2 si le concateneaza la s1
  for( ; *s2 ; p++, s2++)*p=*s2;
  // p este pozitionat pe prima pozitie de dupa caracterele concatenate
  *p='\0';                         // adauga terminatorul
  return s1;                       // strcat trebuie intotdeauna sa returneze s1
}
```

Pentru a se memora mai multe şiruri se pot folosi matrici, fiecare linie din matrice memorând câte un şir: „char siruri[10][30]” (*siruri* poate conţine maxim 10 şiruri de caractere, fiecare cu maxium 29 de caractere+terminator). În acest caz, *siruri[i]* este şirul de caractere de la poziţia *i* (de fapt este un pointer la linia *i* din matrice, aşa cum s-a discutat la laboratorul cu funcţii)

Exemplul 10.4: Se va citi un n<=10. Se vor citi apoi *n* persoane, fiecare definită cu *nume* şi *vârstă* (întreg). Să se sorteze persoanele în ordine alfabetică şi să se afişeze.

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

int main()
{
  char nume[10][30], saux[30];          // vector de nume, sir auxiliar
  int varste[10];                       // vector de varste
  int n, i, schimbare, vaux;
  printf("n: ");
  scanf("%d", &n);
  getchar();                            // citire pentru a elimina '\n' ramas la intrare
  for (i = 0; i < n; i++){              // introducere persoane
     printf("nume %d: ", i);
     fgets(nume[i], 29, stdin);
     printf("varsta %d: ", i);
     scanf("%d", &varste[i]);
     getchar();                         // citire pentru a elimina '\n' ramas la intrare
  }
  do{
     schimbare = 0;
     for (i = 1; i < n; i++){
        if (strcmp(nume[i - 1], nume[i]) > 0){
           strcpy(saux, nume[i - 1]);   // interschimbare nume
           strcpy(nume[i - 1], nume[i]);
           strcpy(nume[i], saux);
           vaux = varste[i - 1];        // interschimbare varste
           varste[i - 1] = varste[i];
           varste[i] = vaux;
           schimbare = 1;
        }
     }
  } while (schimbare);
  for (i = 0; i < n; i++){              // afisare persoane
     printf("%s %d\n", nume[i], varste[i]);
  }
  return 0;
}
```

Când se citeşte cu *scanf* un număr, *scanf* sare peste caracterele de tip spaţiu (spaţiu, ENTER, TAB, etc) de dinaintea numărului. Numărul este apoi citit şi caracterul ENTER de după număr rămâne în aşteptare (în bufferul de intrare, fără a fi consumat), deoarece el este considerat doar terminator şi nu face parte din număr. Dacă după aceasta se citeşte un alt număr este în regulă, deoarece noul *scanf* va sări la rândul său peste caracterele de tip spaţiu existente, deci şi peste ENTER-ul rămas în aşteptare. Dacă însă după ce s-a citit un număr se va citi un şir de caractere (ex: *fgets*), ENTER-ul rămas în aşteptare va termina automat încă de la început (înainte de a se citi orice caracter) şirul ce se doreşte a fi citit şi deci va rezulta un şir vid. Pentru a se evita aceasta, după fiecare citire de număr cu *scanf* s-a folosit *getchar* pentru a se consuma ENTER-ul rămas după citirea numărului respectiv. Puteţi testa cum funcţionează programul de mai sus, cu cele 2 *getchar* comentate, pentru a înţelege mai bine acest comportament.

Aplicaţii propuse

Aplicaţia 10.2*: Se va citi un n<=10. Se vor citi apoi pe rând n nume. Se cere ca în final să fie afişat numele cel mai lung şi lungimea sa.*

Aplicaţia 10.3: Se citeşte un n<=10 şi apoi *n* nume de persoane, fiecare nume putând apărea de mai multe ori. Să se afişeze de câte ori apare fiecare nume.

Aplicaţia 10.4: Se citeşte un text care este alcătuit doar din litere. Să se afişeze pentru fiecare literă de câte ori apare în text.

Aplicaţia 10.5: Se citeşte *n* din intervalul [3,10] şi apoi *n* nume de persoane. Să se concateneze primele *n-1* nume folosind „, ”, ultimul nume cu „ si ”, iar apoi să se adauge „invata.”. Să se afişeze propoziţia rezultată.
Exemplu: n=3, numele: Ion Ana Maria
Propoziţie rezultata: Ion, Ana şi Maria invata.

Aplicaţia 10.6: Se va citi un n<=10. Se vor citi apoi *n* produse, fiecare definit prin *nume* (un şir de caractere), *cantitate* (real) şi *preţ unitar* (real). Unele produse pot să fie date de mai multe ori, cu valori diferite. Să se calculeze pentru fiecare produs cantitatea şi preţul total, iar în final să se calculeze şi preţul global pentru toate produsele. Exemplu:

Intrare	Ieşire
3 mere 2.5 4 pere 2 7.5 mere 3 5	mere 5.5, 25 pere 2, 15 preţ global: 40