10 Текстуални низи

1. Да се напише функција која од низата знаци ќе ги отстрани бланко знаците што се наоѓаат на крајот од низата.

```
#include <stdio.h>
#include <ctype.h>
#define MAXELEM 50

void RemoveTrail(char *str);
int strlen(const char *s);

int main ()
{
	char s[MAXELEM], ch;
	int i = 0;

	while((i<MAXELEM-1) && ((ch=getchar())!='\n'))
		s[i++] = ch;
	s[i] = '\0';
	printf("Vnesenata tekstualna niza e \"%s\".\n", s);
	RemoveTrail(s);
	printf("Tekstualnata niza po izvrsenite promeni e \"%s\".\n", s);
	return 0;
}

int strlen(const char *s)
{
	int n;
	for(n=0; *s!='\0' ;s++)
		n++;
	return n;
}

void RemoveTrail(char *str)
{
	int i;
	i = strlen(str);
	for (i--; i>=0 && isspace(str[i]); i--)
	{
	}
	str[i+1] = '\0';
}
/*so najavuvanje na bibliotekata <string.h> moze da ne se pisuva
funkcijata int strlen(const char *s);
```

2. Да се напише функција што ќе одреди колку пати знак се наоѓа во даден стринг. Знакот за споредување и стрингот се внесуваат од тастатура.

```
#include <stdio.h>
#define MAXELEM 50

int broiZnak(char *str, char znak);

int main ()
{
	char s[MAXELEM], znak;

	printf("Vnesete string: ");
	gets(s);
	printf("Vnesete znak koj treba da se bara vo vneseniot sting: \n");
	znak = getchar();
	printf("Vo vneseniot string \"%s\", znakot \"%c\" se pojavuva
vkupno %d pati.\n", s, znak, broiZnak(s, znak));

	return 0;
}

int broiZnak(char *str, char znak)
{
	int brojac = 0;
	while (*str !='\0')                  //while (*str)
	{
		brojac += (*str == znak);     //brojac += (*str++ == znak);
		str++;
	}
	return brojac;
}
```

3. Да се напише функција која од дадена низа знаци ќе ги исфрли знаците почнувајќи од n-тиот во должина од k знаци.

```
#include <stdio.h>
#include <string.h>
#define MAXELEM 50

void strDelete(char *str, int poz, int dolz);

int main ()
{
	char s[MAXELEM];
	int poz, dolz;

	printf("Vnesete string: ");
	gets(s);
	printf("Vnesete od koja pozicija i kolku znaka treba da se isfrlat:
\n");
	scanf("%d %d", &poz, &dolz);
	strDelete(s, poz, dolz);
	printf("Novo dobieniot string e: ");
	puts(s);
```

```
	return 0;
}
// ############ VERZIJA 1 ###############
void strDelete(char *str, int poz, int dolz)
{
	char *s = str + poz + dolz - 1, *d = str + poz;
	int len = strlen(str);
	for (; *s && ((poz+dolz)<len); *d++ = *s++);
	*d = 0;
}
// ############ VERZIJA 2 ###############
void strDelete(char *str, int poz, int dolz)
{
	if ((poz+dolz)<strlen(str)) strcpy(str+poz-1, str+poz+dolz-1);
	else *(str+poz-1) = 0;
}
```

4. Да се напишат функција која ќе врати подниза од зададена текстуална низа определена со позицијата и должината што како параметри се вчитуваат од тастатура. Поднизата започнува од карактерот што се наоѓа на соодветната позиција во текстуалната низа броено од лево.

```
#include <stdio.h>
#include <string.h>
#define MAXELEM 50

int main ()
{
	char s[MAXELEM], dest[MAXELEM];
	int poz, dolz;

	printf("Vnesete string: ");
	gets(s);
	printf("Vnesete pozicija i broj na znaci za podnizata: \n");
	scanf("%d %d", &poz, &dolz);
	if (poz<=strlen(s))
	{
		strncpy(dest, s+poz-1, dolz);
		printf("Novo dobienata tekstualna niza e: ");
		puts(dest);
	}
	else printf("Vnesena e nevalidna pozicija za podnizata, vnesenata
niza ima samo %d znaci.\n", strlen(s));

	return 0;
}
```

5. Да се напише функција која во стринг што и се предава како влезен параметар ќе ги промени малите букви во големи и обратно и ќе ги отфрли сите цифри.

```
#include <stdio.h>
#include <ctype.h>
#define MAXELEM 50

void promeniString(char *str);
```

```
int main ()
{
      char s[MAXELEM];

      printf("Vnesete string: ");
      gets(s);
      promeniString(s);
      printf("Novo dobieniot string e: ");
      puts(s);

      return 0;
}

void promeniString(char *str)
{
    int i = 0, j = 0;
    while (str[i] != '\0')
    {
          if (!(isdigit(str[i])))
          {
               if (islower(str[i])) str[j] = toupper(str[i]);
               else if (isupper(str[i])) str[j] = tolower(str[i]);
                  else str[j] = str[i];
               j++;
        }
        i++;
    }
    str[j] = '\0';
}
```

6. Да се напише програма која за дадена низа од знаци (внесена од тастатура) ќе провери дали е палиндром (исто се чита и од десно на лево и од лево на десно). Од внесениот збор, пред проверката дали е палиндром, да се исфрлат празните места и да не се прави разлика помеѓу мали и големи букви.

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#define MAXELEM 50

void promeniString(char *str);
int Palindrom(char *str);

int main ()
{
      char s[MAXELEM];

      printf("Vnesete string: ");
      gets(s);
      printf("Vneseniot string e \"%s\" i %s palindrom.\n", s,
Palindrom(s)? "E": "NE E");

      return 0;
}

void promeniString(char *str)
{
      char *a = str, *b = str;
```

```
	while (*a)
	{
		if (isalpha(*a))
		{
			if (isupper(*a)) *b = tolower(*a);
			else *b = *a;
			*b++;
		}
		*a++;
	}
	*b = 0;
}

// ############ VERZIJA 1 ###############

int Palindrom(char *str)
{
	int palin = 1, i, len = strlen(str);
	promeniString(str);
	for (i=0; i<len/2; i++)
		if (*(str+i) != *(str+len-1-i)) palin = 0;
	return palin;
}

// ############ VERZIJA 2 ###############

int Palindrom(char *str)
{
	char *a , *b;
	promeniString(str);
	a = str;
	b = str + strlen(str) - 1;
	while (a<b && *a++==*b--);
	return (a>=b);
}
```

Basic String Handling Functions

All the string handling functions are prototyped in:

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

The common functions are described below:

`char *stpcpy (const char *dest,const char *src)` -- Copy one string into another.
`int strcmp(const char *string1,const char *string2)` - Compare string1 and string2 to determine alphabetic order.
`char *strcpy(const char *string1,const char *string2)` -- Copy string2 to stringl.
`char *strerror(int errnum)` -- Get error message corresponding to specified error number.
`int strlen(const char *string)` -- Determine the length of a string.
`char *strncat(const char *string1, char *string2, size_t n)` -- Append n characters from string2 to stringl.
`int strncmp(const char *string1, char *string2, size_t n)` -- Compare first n characters of two strings.
`char *strncpy(const char *string1,const char *string2, size_t n)` -- Copy first n characters of string2 to stringl .
`int strcasecmp(const char *s1, const char *s2)` -- case insensitive version of `strcmp()`.
`int strncasecmp(const char *s1, const char *s2, int n)` -- case insensitive version of `strncmp()`.

The use of most of the functions is straightforward, for example:

```
char *str1 = "HELLO";
char *str2;
int length;

length = strlen("HELLO"); /* length = 5 */
(void) strcpy(str2,str1);
```

Note that both `strcat()` and `strcopy()` both return a copy of their first argument which is the destination array. Note the order of the arguments is *destination array* followed by *source array* which is sometimes easy to get the wrong around when programming.

The `strcmp()` function *lexically* compares the two input strings and returns:

Less than zero
-- if `string1` is lexically less than `string2`
Zero
-- if `string1` and `string2` are lexically equal
Greater than zero
-- if `string1` is lexically greater than `string2`

This can also confuse beginners and experience programmers forget this too.

The `strncat(), strncmp()` and `strncpy()` copy functions are string restricted version of their more general counterparts. They perform a similar task but only up to the first `n` characters. Note the the `NULL` terminated requirement may get violated when using these functions, for example:

```
char *str1 = "HELLO";
char *str2;
int length = 2;


(void) strcpy(str2,str1, length); /* str2 = "HE" */
```

`str2` is NOT NULL TERMINATED!! -- BEWARE

String Searching

The library also provides several string searching functions:

`char *strchr(const char *string, int c)` -- Find first occurrence of character `c` in string.
`char *strrchr(const char *string, int c)` -- Find last occurrence of character `c` in string.
`char *strstr(const char *s1, const char *s2)` -- locates the first occurrence of the string `s2` in string `s1`.
`char *strpbrk(const char *s1, const char *s2)` -- returns a pointer to the first occurrence in string s1 of any character from string `s2`, or a null pointer if no character from `s2` exists in `s1`
`size_t strspn(const char *s1, const char *s2)` -- returns the number of characters at the begining of `s1` that match `s2`.
`size_t strcspn(const char *s1, const char *s2)` -- returns the number of characters at the begining of `s1` that *do not* match `s2`.
`char *strtok(char *s1, const char *s2)` -- break the string pointed to by `s1` into a sequence of tokens, each of which is delimited by one or more characters from the string pointed to by `s2`.
`char *strtok_r(char *s1, const char *s2, char **lasts)` -- has the same functionality as strtok() except that a pointer to a string placeholder lasts must be supplied by the caller.

`strchr()` and `strrchr()` are the simplest to use, for example:

```
char *str1 = "Hello";
char *ans;

ans = strchr(str1,'l');
```

After this execution, `ans` points to the location `str1 + 2`

`strpbrk()` is a more general function that searches for the first occurrence of any of a group of characters, for example:

```
char *str1 = "Hello";
char *ans;

ans = strpbrk(str1,'aeiou');
```

Here, `ans` points to the location `str1 + 1`, the location of the first `e`.

strstr() returns a pointer to the specified search string or a null pointer if the string is not found. If s2 points to a string with zero length (that is, the string ""), the function returns s1. For example,

```
char *str1 = "Hello";
char *ans;

ans = strstr(str1,'lo');
```

will yield `ans = str + 3`.

`strtok()` is a little more complicated in operation. If the first argument is not NULL then the function finds the position of any of the second argument characters. However, the position is remembered and any subsequent calls to `strtok()` will start from this position if on these subsequent calls the first argument is `NULL`. For example, If we wish to break up the string `str1` at each space and print each token on a new line we could do:

```
char *str1 = "Hello Big Boy";
char *t1;


for ( t1 = strtok(str1," ");
      t1 != NULL;
      t1 = strtok(NULL, " ") )

printf("%s\n",t1);
```

Here we use the for loop in a non-standard counting fashion:

- The initialisation calls `strtok()` loads the function with the string `str1`
- We terminate when t1 is `NULL`
- We keep assigning tokens of `str1` to `t1` until termination by calling `strtok()` with a `NULL` first argument.

Character conversions and testing: `ctype.h`

We conclude this chapter with a related library `#include <ctype.h>` which contains many useful functions to convert and test *single* characters. The common functions are prototypes as follows:

Character testing:

`int isalnum(int c)` -- True if c is alphanumeric.
`int isalpha(int c)` -- True if c is a letter.
`int isascii(int c)` -- True if c is ASCII .
`int iscntrl(int c)` -- True if c is a control character.
`int isdigit(int c)` -- True if c is a decimal digit
`int isgraph(int c)` -- True if c is a graphical character.
`int islower(int c)` -- True if c is a lowercase letter
`int isprint(int c)` -- True if c is a printable character
`int ispunct (int c)` -- True if c is a punctuation character.
`int isspace(int c)` -- True if c is a space character.
`int isupper(int c)` -- True if c is an uppercase letter.
`int isxdigit(int c)` -- True if c is a hexadecimal digit

Character Conversion:

`int toascii(int c)` -- Convert `c` to ASCII .
`tolower(int c)` -- Convert c to lowercase.
`int toupper(int c)` -- Convert c to uppercase.