Lecture13
Character and String Processing.
Fundamentals of Characters and Strings in C language

Characters:

-Building blocks of programs. Every program is a sequence of meaningfully grouped characters.

-Character variable is defined to be of type char. For example: **char ch.** One character variable size is a single byte which contains the code for the character. This code is a numeric value and depends on the character coding system being used (is machine-dependent). The most common system is ASCII (American Standard Code for Information Interchange).

-Character literal (constant character) is written by enclosing the character between a pair of single quotes. For example: **'A', '\0'.**

Strings:

-In C language does not exist data type string and therefore do not exist variables of string type but it is possible to process strings. A string in C language represents a set of characters stored in statically or dynamically allocated 1-D array of character type and terminated by null byte (null character) and such a set of characters is considered as one object - a string. If null byte is not appended at the end of a set of characters stored in array corresponding set of characters is not considered as a string. Name of statically allocated array and name of pointer for dynamic array of characters are also used as names of strings stored in these arrays. Note that the name of a string is a pointer constant (for an array allocated statically) or pointer variable (for dynamic array) both represent the address of string's first character. What is why each character of the string can be accessed by using this name and indexation or dereferencing operators.

-In C language a string literal (constant string) is written in double quotes. For example: **"Hello".** Note that the constant strings are allocated statically in the fixed memory and cannot by modified.

Assigning character and string literals.

A character literal (constant character) can be assigned to a character variable at compile or run time. For example: **char ch = 'A';** or **char ch; ch = 'A';**

A string literal which is statically allocated in the fixed memory can be assigned:

1. to a statically allocated character array or 2. to a pointer of type **char***.

Ex.1. **char color[10] = "blue";** or **char color[ ] = "blue";**

Compiler allocates statically 1-D character array, named **color**, of 10 or 5 elements and initialized it by 5 characters: **'b' ,'l', 'u', 'e'** and **'\0' (null byte).** This is possible only at compile time by indicating or not the number of elements of array. Note that the indicated number of elements of array has to be greater or equal to the number of characters of the string plus one element for null character. If number of elements is not indicated compiler itself determines the number of needed elements of array.

Ex.2. **char* pc = "blue";** or **char *pc; pc = "blue**";

In this example memory is statically allocated for one pointer variable named **pc** of **char*** type and this pointer is initialized at compile or run time (attention!) by the first byte address of the fixed memory location where is stored constant string **"blue"**.

Initializing an 1-D character array during declaration.

1-D statically allocated character array can be initialized as any array during declaration 1) by assigning to it a string literal as we just considered before in Ex.1 and also 2) by assigning to it a set of characters ended by null byte.

For example: **char color[10] = {'b', 'l', 'u', 'e', '\0'};** or **char color[ ] = {'b','l', 'u', 'e', '\0' };**

The result of these statements is absolutely the same as the result of the statements of Ex.1.

Input (reading) characters and strings from keyboard.

Input (reading) a character from keyboard is performed by using standard functions **scanf( ), getchar( )** declared in header file **stdio.h** and functions **getch( ) , getche( )** declared in header file **conio.h.** For example, after declaration statement **char ch;** character variable **ch** can be initialized by introducing (reading) a character from keyboard in 4 ways:

1. **scanf("%c", &ch);** Note using of format specification **%c** for character type variable;
2. **ch = getchar( );** Note that after pressing a character key on the keyboard it is necessary to press also the Enter key, because function **getchar( )** uses buffer memory of console as the function **scanf( )** does;
3. **ch = getch( );** Note that function **getch( )** does not use buffer memory what is way after pressing a character key its value is directly assigned to corresponding character variable **ch** without additional pressing the Enter key;
4. **ch = getche( );** Function **getche( )** defers from function **getch( )** only by showing on the screen a character (echo) introduced from keyboard as functions **scanf( )** and **getchar( )** do.

Input (reading) a string of characters from keyboard is performed by using standard function **scanf( )** and **gets( )** declared in header file stdio.h. For example, a string can be introduced from keyboard as follows:

scanf("%s", str); or **gets(str);** where **str** is the string name (name of character array or pointer for dynamic array where the string is stored).

Note using of format specification **%s** in function **scanf( )** call and absence of ampersand **&** before **str** in both function calls, because name of string is a pointer(an address). It is also important that function **scanf( )** inputs(reads) a string from keyboard only till the first space, but function **gets( )** can input a string with spaces. It is recommended before functions **scanf( ), getchar( )** and **gets( )** calls to use the function **fflush(stdin)** call for clearing the buffer memory.

Output (writing) characters and strings on the screen.

Output (writing) a character on the screen is performed by using standard functions **printf( ), putchar( )** declared in header file **stdio.h** and function **putch( )** declared in header file **conio.h.** For example:

printf("%c", ch); or **putchar(ch);** or **putch(ch);** for output on the screen a character stored in the character variable **ch;**

printf("%c", 'A'); or **putchar('A');** or **putch('A');** for output on the screen character **A**;

printf("%c",'\n'); or **putchar('\n');** or **putch('\n');** for moving the cursor at the beginning of the new line.

Output (writing) a string of characters on the screen is performed by using standard functions **printf( )** and **puts( )** declared in header file **stdio.h.** For example:

printf("%s", str); or **puts(str);** for output on the screen a string named **str** (name of character array or pointer for dynamic character array where the string is stored);

printf("Hello, my friends!\n"); or **puts("Hello, my friends!");** for output on the screen the string **Hello, my friends!** and for moving the cursor at the beginning of the new line.

printf("\n\n"); or **puts("\n");** for output one blank line and for moving the cursor at the beginning of the new line.

Character and string processing.

For character and string processing in C language can be used different functions from standard library header files **ctype.h** (character handling library for character processing), **stdlib.h** (general utilities library for string conversion functions) and **string.h** (string handling library for string processing). Descriptions and examples of using of these functions presented in the Help option of Main menu of Turbo C++ compiler.

The most useful functions for string processing from header file **string.h** are:

strlen( ) – to determine string length (returns number **n** of characters in the string **str**); Ex.: **n=strlen(str);**

strcat( ) – to join (concatenate) string **s1** and string **s2** together in one string **s1**. Ex.: **strcat(s1, s2);**

strcpy( ) – to copy (assign) contents of string **s2** to string **s1**; Ex.: **strcpy(s1, s2);**

strcmp( ) – to compare string **s1** and string **s2** (returns **k**=0 if strings are the same, **k**<0 if string **s1** < string **s2** (upper in dictionary) and **k**>0 if string **s1** > string **s2** (lower in dictionary); Ex.: **k=strcmp(s1, s2);**

stricmp( ) – this function is the same as **strcmp( )** but not case sensitive; Ex.: **k = stricmp(s1, s2);**

strrev( ) – to reverse the characters in string **str**; Ex.: **strrev(str);**

strupr( ) – to convert all characters in string **str** from lowercase to uppercase; Ex.: **strupr(str);**

strlwr( ) – to convert all characters in string **str** from uppercase to lowercase; Ex.: **strlwr(str);**

Lecturer: Mihail Kulev, associate professor