CS1100 - Lecture 17

Instructor : Jasine Babu

Teaching Assistants : Nikhila K N, Veena Prabhakaran

So far we have learned about different types integers and floats. In this lecture, we will discuss about a new data type called character.

Characters

The way of defining a character variable is `char variable_name;`. Usually *1* byte is enough to store a character variable. Characters technically can be unsigned and signed. Some examples of characters are the alphabets, digits and symbols like `<, >, @, $` which are there on the keyboard. There are also some special characters like `'\n'` (newline character) and `'\t'` (tab character). Thus, characters can be visible or invisible. A character constant is usually represented by a symbol put in single quotes.

If `x` is a variable of type `char`, a value for this character variable can be read from the terminal using one of the following instructions:

```
scanf("%c",&x);
x=getchar();
```

The instruction `getchar()` will read one character from the terminal and returns its value. For displaying characters, we can either use `printf("%c",x);` or we can use `putchar(x);`.

Since a character variable occupies 1-byte, its binary representation can be any 8-bit sequence of 1s and 0s. Character data type is technically considered as a subset of integer data type. Each character has its corresponding integer value. This integer value is the decimal equivalent of the one byte binary representation of the character. For example,

- Character constant 'A' has binary representation is `0 1 0 0 0 0 0 1` and this corresponds to integer value 65.
- Character constant 'B' has binary representation is `0 1 0 0 0 0 1 0` and this corresponds to integer value 66.
- Character constant 'a' has binary representation is `0 1 1 0 0 0 0 1` and this corresponds to integer value 97.
- Character constant 'b' has binary representation is `0 1 1 0 0 0 1 0` and this corresponds to integer value 98.
- Character constant '0' has binary representation is `0 0 1 1 0 0 0 0` and this corresponds to integer value 48.
- Character constant '1' has binary representation is `0 0 1 1 0 0 0 1` and this corresponds to integer value 49.

What are the points that should be remembered about the ordering of characters and their integer values? It will be useful to remember that in the character set, upper case letters ‘A’, ‘B’, ‘C’ ···, ‘Z’ appear consecutively in their alphabetical order. Similarly, lower case letters ‘a’, ‘b’, ‘c’ ···, ‘z’ appear consecutively in their alphabetical order. Digits ‘0’, ‘1’, ‘2’ ···, ‘9’ appear consecutively in their numerical order.

What is the relevance of having integer values for characters? Since characters form a subtype of integers, we can have arithmetic operations or comparisons performed on character variables. From the description in the above, one can notice that relations like ‘A’<‘B’, ‘B’<‘C’, ‘a’<‘b’, ‘0’<‘1’ etc. hold. Moreover, the integer value of ‘A’-‘a’ is the same as the integer value of ‘C’-‘c’ or ‘Z’-‘z’. If we take ‘D’-‘A’, we get 3 which is the difference in position of ‘D’ from ‘A’ among the capital letters.

Suppose we have an integer variable y and character variable x, then the instruction y=x; is valid. After executing the instruction y=x;, y gets the integer value corresponding to the character held by x.

The following program demonstrates using character variables in arithmetic expressions.

```
#include<stdio.h>
int main()
{

   char w,x;
   int y,z;
   x=‘A’;
   y=x;
   printf("x=%c integer value of ‘A’ is %d, x+y=%d \n", x, y, x+y);
   z=‘a’;
   printf("integer value of ‘a’ is %d \n", z);
   printf("z=%d \n", z);
   printf("‘a’ - ‘A’ is %d\n", z-x);
   w=‘T’ + (‘a’ - ‘A’);
   printf(" w is %c\n", w);
   return 0;
}
```

In the above program, x is the variable of type char and its value is ‘A’ (which is equivalent to integer value 65) and y is an integer variable. Therefore, y gets value 65 after executing the instruction y=x;. The value of the expression x+y is the sum of the integer values of x and y, which is 130. When x+y is displayed using formating text %d in printf, the value displayed will be 130. When the integer variable z is assigned value ‘a’ (which is equivalent to integer value 97) , value of z changes to 97. The value of the expression z-x is calculated by considering x as an integer. Therefore the value of z-x is 97-65=32. The expression ‘a’-‘A’ also has value 32. Since, ‘t’-‘T’ has the same value as ‘a’-‘A’, the value of the expression ‘T’ + (‘a’ - ‘A’) is the same as the integer value of character ‘t’. When this integer value is assigned to the character variable w, the value of the variable w as a character becomes ‘t’. So, when w is displayed using format text %c, it displays the symbol t.

After executing the above program,the following output will be displayed.

```
x=A integer value of 'A' is 65, x+y=130
integer value of 'a' is 97
z=97
'a' - 'A' is 32
 w is t
```

Strings

A string is an array of characters terminated by a null (\0) character. To declare a variable to hold a string it is enough to declare an array of characters of sufficient length. It should be kept in mind that, when we decide the length of the character array, the space required to store the \0 character should also be counted. For example, if we declare an array of characters `str` using the declaration `char str[20];`, We can store only 19 characters other than the \0 character, if we have to use `str` as a string. A string constant is written as a sequence of characters enclosed in double quotes. For example "hellow how are you?" is a string constant. Though there is no \0 at the end of a string, it is assumed to be implicitly present as a character after all the other characters within the quotation. Thus, to store the string "hellow how are you?", we need an array of characters of size 20, though there are only 19 visible characters in this string.

To initialize a character array `str` of size 20 with the string "hellow how are you?" we can use the following method.

```
char str[20]=''hellow how are you?'';
```

If we do the above initialization, the content of each element of the character array `str` is as follows.

```
str[0]='h'
str[1]='e'
str[2]='l'
.
.
.
str[17]='u'
str[18]='?'
str[19]='\0'
```

To display a string `str`, we can use `printf(''%s'', str);`. When this instruction is executed, characters from the beginning of the array `str` are printed until a \0 character is encountered. As per our previous declaration, when the statement `printf(''%s'', str);` is executed, the message `hellow how are you?` will be displayed.

Now, suppose we change the declaration as below.

```
char str[20]=‘‘hello!’’;
```

The contents of each element of the character array `str` is as follows.

```
str[0]=‘h’
str[1]=‘e’
str[2]=‘l’
str[3]=‘l’
str[4]=‘o’
str[5]=‘!’
str[6]=‘\0’
str[2]=JUNK
.
.
.
str[18]=JUNK
str[19]=JUNK
```

If we execute `printf(‘‘%s’’, str);` after this modified declaration, the message `hello!` will be displayed.

Suppose we change the declaration as the following

```
char str[20]=‘‘hellow!  how are you?’’;
```

Note that, this array has 20 visible characters. Hence, there is no space in this array to store the `\0` character. If we try to execute `printf(‘‘%s’’, str);` after this modified declaration, it is likely to cause a buffer overflow and a segmentation fault.

To input a string from the terminal to a character array `str`, we can use the statement `scanf(‘‘%s’’,str);`. The input is accepted to `str` only until a white space character (like a space or newline or tab) is entered. The white space character will not be accepted to `str`. If the number of characters entered up to the white space character is less than the size of the array, then after putting all other characters accepted from the terminal one by one to the array, a `\0` character is added to the array. If the number of characters entered through the terminal is greater than or equal to the size of the array, then there is no space to store the `\0` character. If this happens, then trying to print the string is likely to cause a segmentation fault as explained earlier.

Similarly, if we have a character array `str` without `\0` character in it, then trying to print the array using `printf(‘‘%s’’,str);` is likely to cause a segmentation fault.