LECTURE-13

STRING OPERATION

13.1 introduction : We have been using strings in our previous lecture without going deep into its properties. In the present lecture we will study the strings in details.

Combination of characters is called a string. For example, " My name is Vikram Singh" is a string. One can print a string as:

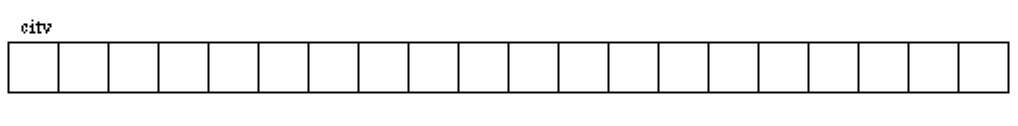
```
cout<<'\n My name is Vikram Singh";
```

As you may already know, the C++ Standard Library implements a powerful string class, which is very useful to handle and manipulate strings of characters. However, because strings are in fact sequences of characters, we can represent them also as plain arrays of char elements.

For example, the following array:

```
char city [20];
```

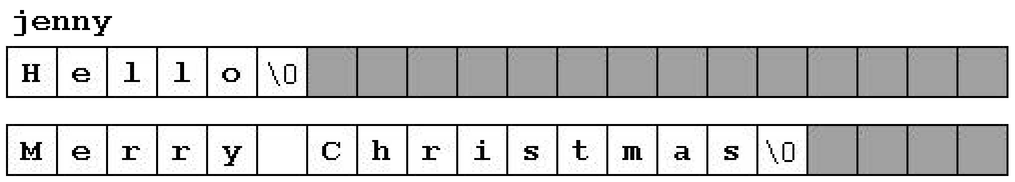
is an array that can store up to 20 elements of type char. It can be represented as:



Therefore, in this array, in theory, we can store sequences of characters up to 20 characters long. But we can also store shorter sequences. For example, 'city' could store at some point in a program either the sequence "Hello" or the sequence "Merry christmas", since both are shorter than 20 characters.

Therefore, since the array of characters can store shorter sequences than its total length, a special character is used to signal the end of the valid sequence: the *null character*, whose literal constant can be written as '\0' (backslash, zero).

Our array of 20 elements of type char, called jenny can be represented storing the characters sequences "Hello" and "Merry Christmas" as:



NB : Notice how after the valid content a null character ('\0') has been included in order to indicate the end of the sequence. The panels in gray color represent char elements with undetermined values.

13.2 Initialization of null-terminated character sequences

Because arrays of characters are ordinary arrays, they follow all their same rules. For example, if we want to initialize an array of characters with some predetermined sequence of characters we can do it just like any other array:

```
char myword[] = { 'H', 'e', 'l', 'l', 'o', '\0' };
```

In this case we would have declared an array of 6 elements of type char initialized with the characters that form the word "Hello" plus a null character '\0' at the end.

But arrays of char elements have an additional method to initialize their values: using string literals.

In the expressions we have used in some examples in previous chapters, constants that represent entire strings of characters have already showed up several times. These are specified enclosing the text to become a string literal between double quotes ("). For example:

```
"the result is: "
```

is a constant string literal that we have probably used already.

Double quoted strings (") are literal constants whose type is in fact a null-terminated array of characters. So string literals enclosed between double quotes always have a null character ('\0') automatically appended at the end.

Therefore we can initialize the array of char elements called myword with a null-terminated sequence of characters by either one of these two methods:

```
char myword [] = { 'H', 'e', 'l', 'l', 'o', '\0' };
char myword [] = "Hello";
```

In both cases the array of characters myword is declared with a size of 6 elements of type char: the 5 characters that compose the word "Hello" plus a final null character ('\0') which specifies the end of the sequence and that, in the second case, when using double quotes (") it is appended automatically.

Please notice that we are talking about initializing an array of characters in the moment it is being declared, and not about assigning values to them once they have already been declared. In fact because this type of null-terminated arrays of characters are regular arrays we have the same restrictions that we have with any other array, so we are not able to copy blocks of data with an assignment operation.

Assuming mytext is a char[] variable, expressions within a source code like:

```
mystext = "Hello";
mystext[] = "Hello";
```

would not be valid, like neither would be:

```
mystext = { 'H', 'e', 'l', 'l', 'o', '\0' };
```

The reason for this may become more comprehensible once you know a bit more about pointers, since then it will be clarified that an array is in fact a constant pointer pointing to a block of memory.

13.3 Using null-terminated sequences of characters

Null-terminated sequences of characters are the natural way of treating strings in C++, so they can be used as such in many procedures. In fact, regular string literals have this type (char[]) and can also be used in most cases.

For example, cin and cout support null-terminated sequences as valid containers for sequences of characters, so they can be used directly to extract strings of characters from cin or to insert them into cout. For example:

Program 13.1: Null-terminated sequences of characters

```
// null-terminated sequences of characters
#include <iostream>
using namespace std;

int main ()
{
  char question[] = "Please, enter your first name: ";
  char greeting[] = "Hello, ";
  char yourname [80];
  cout << question;
  cin >> yourname;
  cout << greeting << yourname << "!";
  return 0;
}
```

```
Please,
enter
your
first
name:
John
Hello,
John!
```

As you can see, we have declared three arrays of char elements. The first two were initialized with string literal constants, while the third one was left uninitialized. In any case, we have to speficify the size of the array: in the first two (question and greeting) the size was implicitly defined by the length of the literal constant they were initialized to. While for yourname we have explicitly specified that it has a size of 80 chars.

Finally, sequences of characters stored in char arrays can easily be converted into string objects just by using the assignment operator:

```
string mystring;
char myntcs[]="some text";
mystring = myntcs;
```

13.4 Initialising multiple strings

In case e have to use data involving multiple strings, e initialize string as char city [10][20]; To understand multiple string operation let us study program 13.2.

Program 13.2 : Initializing multiple strings

```
#include <conio.h>
#include <conio.h>
using namespace std;

void main()
{
 char city[5][20];int i;
 cout<<"Enter names of ten cities:";
   for (i=0; i<5; i++)
   {
           cin>> city[i];
   }
   cout <<"Names of the cities are:";
   for (i=0; i<=5; i++)
   {
           cout<< city[i];
   }
}
/*Output
Enter names of ten cities:Delhi Calcutta Ludhiana Chandigarh Amritsar
Rupnagar
Names of the cities are:DelhiCalcuttaLudhianaChandigarhAmritsar¦¦¦¦£9-
É+ ?Press
any key to continue . . .*/
```

In the above program first bracket shos e chan enter five names and second shos each name can be of length of 20 characters.

13.5 String handling Functions

C++ language is rich in library functions, but to handle or operate some strings, e ill discuss some of the functions goven below. These functions are included in header file <string.h> , which e have to include in the program. These functions are:

(i) strcat():

(ii) strcmp():

(iii) strcpy():

(iv) strlen():

(i) strcat(): The purpose of strcat() is to concatenate or combine two strings together. The general syntax used for this is as:

strcat(string1,string2);

Also e can combine more than to strings as below:

strcat(strcat(string1,string2),string3);

(ii) strcmp(): The purpose of this function is to compare two strings. The general syntax is :

strcmp(string1,string2);

strcmp will test two strings for equallity.

Returns :

< 0 if s1 is less than s2
0 if s1 == s2
> 0 if s1 is greater than s2

This function probably provides too much information by indicating which string is lexicographically greater. The net result means that the **strcmp** return code is logically incorrect because it returns a FALSE value when the strings match. Folloing program illustrates the function of strcmp():

Program 13.3: use strcmp()

```
int StringCompare(char *s1, char *s2);

main()
{
    char        One[] = "Bartman";
    char        Two[] = "Batman";

    int         Ret;

    Ret = StringCompare(One, Two);

    if (Ret == TRUE)
    {
        puts("The Strings match");
    }
    else
    {
        puts("The Strings do not match");
    }
}

/****************************************************************/

int StringCompare(char *s1, char *s2)
{
    int Ret;

    if (strcmp(s1, s2))
    {
        Ret = 0;
    }
    else
    {
        Ret = 1;
    }

    return (Ret);
}
```

(iii)strcpy(): strcpy copies a string. This function will copy the bytes stored at the location pointed to by 's2' to the location pointed to by 's1'.

```
        s1              s2
        |               |
        V               V
         - - - -        - - - --
        | | | | |       |a|b|c|\0|
         - - - -        - - - --
        ^ ^             | |
        | |             | |
         -|-------------  |
           ---------------
```

Code given below illustrates the functioning of strcpy().

```
library:   string.h

Prototype: char strcpy(char *s1, const char *s2);

Syntax:
           char string2[20]="red dwarf";
           char string1[20]="";
           strcpy(string1, string2);
```

(iv) strlen():strlen will give you the length of a string, NOT including the '\0' terminator. It should not be confused with the sizeof operator which returns the size of a variable (that could hold a string).

```
Library:   string.h
```

```
Prototype: size_t strlen(const char *s);

Syntax:    size_t size;
           char string[20]="red dwarf";
           size = strlen(string);
```

Program 13.4 : Example of strlen().

```
/*********************************************************************
 *
 * Purpose: Reverse characters in a string.
  *
 *********************************************************************/

void reverse(char s[]);

main()
{
  char text[80]="martin";

  printf("string is %s\n", text);
  reverse(text);
  printf("string is %s\n", text);
}

void reverse(char s[])
{
  int c, i, j;

  for (i=0, j=strlen(s)-1; i < j;i++, j--)
  {
    c = s[i];
    s[i] = s [j];
    s[j] = c;
  }
}
```