|  |
| --- |
| **Programming Fundamentals** |
| **(CL 214)** |
| **LABORATORY MANUAL** |
| **Spring 2021** |
| **C:\Users\Aamer\Desktop\nu-new.png**  **LAB 01** |
| **C-strings**  **Engr. Shahid Qureshi** |
| **Engr. Sana Saleh** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | \_\_\_\_\_\_\_\_\_\_ | | | \_\_\_ |
| STUDENT NAME | | ROLL NO | | | SEC |
|  | | | | | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | | |
| LAB ENGINEER SIGNATURE & DATE | | | | | |
| **MARKS AWARDED: /10** | | | | | |
|  | | | | | |
|  | | | | | |
| **NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES (NUCES), ISLAMABAD** | | | | | |
|  | | | | | |
| Last Edited by: | Engr. Sana Saleh | | Version: | 2.01 | |
| Prepared by: | Engr. Sana Saleh | | Date: | 12 Aug, 2017 | |
| Verified by: | Engr. Shahid Qureshi | | Date: | 19 Jan, 2019 | |

|  |  |
| --- | --- |
| **LAB 01** | ***C-Strings*** |

**Lab Objectives:**

1. To learn about C-*strings and strings* datatype.
2. To learn and revise the concepts of character array.

**Software Required:**

* Visual Studio 2010

**Introduction:**

1. ***Strings:***
   1. **C-String (Character Array):**

Strings are character arrays. A bunch of characters most often only make sense if they form words and words only make sense if they are part of a sentence. Essentially, character arrays are used for storage of words and sentences.

The most commonly used term for character arrays is C-strings. However, there is a subtle difference between character arrays and C-strings. Recall that a string is a sequence of zero or more characters, and strings are enclosed in double quotation marks. In C++, C-strings are null terminated; that is, the last character in a C-string is always the null character. A character array might not contain the null character, but the last character in a C-string is always the null character. As you will see, the null character should not appear anywhere in the C-string except the last position. Also, C-strings are stored in (one-dimensional) character arrays.

* + 1. **Initialization at Declaration:**

Array of characters can be initialized with some predetermined sequence of characters. To initialize character array with word “Hello”, we can do it just like any other array:

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

The above character array” myword” reserved 6 bytes. The end character of array is null character. Once a character array is declared and initialized, we can’t reassign values to it by following method:

|  |
| --- |
| myword = {‘B’’y’,’e’,’\0’}; //INVALID  myword[] = {‘B’’y’,’e’,’\0’}; //INVALID  myword[] = "Bye"; //INVALID |

Above methods are invalid. If you want to reassign values to character array use following method:

|  |
| --- |
| myword[0]=’B’;  myword[1]=’y’;  myword[2]=’e’;  myword[3]=’\0’; |

Character arrays have an additional way of initialization, in addition to common methods of array initialization. Without specifying size, a *char* array is assigned a *string* at declaration. In this case, compiler automatically allocates space for array. In following code contents of an array are printed using *cout*.

void main ()

{

char cMyname [] = "abc"; //declaration

cout<<cMyname<<endl;

cout<<”Null character is stored”<<cMyname[3]<<endl;

}

* + 1. **Null Character:**

Length of words and sentences is not fixed. When strings are stored in character arrays, they may not take up the whole space. Question arises how does the compiler know when the character array ends? The answer is that every character array end (or must end) with a NULL character (or `\0' in ASCII). In above example, cMyname was allocated 4 bytes by compiler to end the character array with a `\0' character.

* + 1. **String Object:**

In C++, string object can be created for holding strings. Unlike using char arrays, string objects has no fixed length, and can be extended as per your requirement. An example is shown below:

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  // Declaring a string object  string str;  cout << "Enter a string: ";  getline(cin, str);  cout << "You entered: " << str << endl;  return 0; |

* + 1. **C-String Input Functions:**

There are fhree ways to take input from user in a string or character array.

>>operator

cin.get (Use only for C-string)

getline

**cin (>> operator)**

The >> operator may be used when programmer wants to read the next non-blank space characters entered by the user into a character or character array. Any printable characters that follow the first space will be ignored and will not be stored in the variable.

**cin.get() Function**

The unformatted get member function works like the >> operator with two exceptions. First, the **get function includes white-space characters**, whereas the extractor excludes white space. Second, the get function is less likely to cause a tied output stream (cout, for example) to be flushed. Syntax is as follow:

***cin.get(char\_array\_name, char\_array\_length)***

A variation of the get function specifies a buffer address and the maximum number of characters to read. This is useful for limiting the number of characters sent to a specific variable, as this example shows:

|  |
| --- |
| #include <iostream.h>  void main()  {  char line[25];  cout << " Type a line\n>";  cin.get( line, 25 );  cout << ' ' << line;  } |

In this example, you can type up to 24 characters and a terminating character. Any remaining characters can be extracted later.

**cin.getline() Function**

The getline member function is similar to the get function. Both functions allow a third argument that specifies the terminating character for input. The default value is the newline character. **Both functions reserve one character for the required terminating character.** However, get leaves the terminating character in the stream and getline removes the terminating character. Syntax of cin.getline for character array is given below:

***cin.getline(char\_array\_name, char,array\_length)***

Syntax for string is given below:

***getline(cin, string\_name)***

Example of program using cin.getline() is as follow:

|  |
| --- |
| #include <iostream.h>  void main()  {  char line[100];  cout << " Type a line" << endl;  cin.getline( line, 100, 't' );  cout << line;  } |

* + 1. **C-String Manipulation Functions:**

Some of the C-string manipulation functions are described below:

|  |  |
| --- | --- |
| **Function** | **Effect** |
| *strcpy\_s*(s1, s2) | Copies the string s2 into the string variable s1. The length of s1 should be at least as large as s2 |
| *strlen\_s*(s) | Returns the length of the string s, excluding the null character. |
| *strcat\_s(s1,s2)* | Appends a copy of the source string to the destination string. The terminating null character in destination is overwritten by the first character of source, and a null-character is included at the end of the new string formed by the concatenation of both in destination. |
| *strcmp\_s(s1,s2)* | Returns a negative number if string s1 is less than string s2, returns zero if the two strings are equal, and returns a positive number is string s1 is greater than string s2. |

**Concatenation of C++ C-string:**

The concatenation of C++ C-string using *strcat\_s* function of string is as follow.

**Example 1:**

|  |
| --- |
| #include <iostream>  using namespace std;  int main ()  {  char str[80];  strcpy\_s (str,"These ");  strcat\_s (str,"strings ");  strcat\_s (str,"are ");  strcat\_s (str,"concatenated.");  cout<<(str);  return 0;  } |

Program Code

|  |
| --- |
| These strings are concatenated. |

Console Output

**Example 2:**

Following code use all of the C-string functions described above. Run this code.

|  |
| --- |
| #include<iostream>  #include<iomanip>  using namespace std;  void main()  {  char string1[20];  char string2[20];  char string3[] = "Hello";  cout << "Enter the first string\n";  cin >> string1;  cout << "Enter the second string\n";  cin >> string2;  cout << setw(20)<<"Comparing both strings\n";  int compare = strcmp(string1, string2); //String cmp  if (compare==1)  cout << "First string is greater than string 2\n";  else if (compare == 0)  cout << "Both strings are same/equal\n";  else  cout << "String 2 is larger as compared to string 1\n";  cout << setw(20)<<"\nUsing string Copy Function\n";  cout << "Copy string3 to string 1\n";  strcpy\_s(string1, string3); //String copy    cout << setw(20) << "\nUsing string Length Function\n";  cout << "Length of string 1 is " << strlen(string1) << endl;  cout << "Length of string 2 is " << strlen(string2) << endl;  cout << setw(20) << "\nUsing string Concatenation Function\n";  strcat\_s(string1, " ");  strcat\_s(string1, string3);  cout << string1 << endl;  } |

Program Code

1. **Exercise:**
2. Dry run this code and print the output.

|  |  |
| --- | --- |
| #include<iostream>  using namespace std;  void main()  {  char cA[] = { 'A', '\0' };  char cA1[] = { 'B'};  char cA2[] = { 'a' };  char cA3[] = { 'n' };  char cA11[] = { "B" };  char cA22[] = { "a" };  char cA33[] = { "n" };  char data[50] = "Fruit name is ";  char final[100];  cout<<"cA value is"<<cA[0] <<" "<<cA[1]<< endl;  cout<<"cA1value is"<<cA1[0]<<" "<<cA1[1]<<endl;  cout<<"cA11 value is"<<cA11[0]<<" "<<cA11[1]  <<" \n";  cout << "Sizeof Function\n";  cout << "Size of cA is " << sizeof(cA) << endl;  cout << "Size of cA1 is "<< sizeof(cA1)<< endl;  cout << "Size of cA11 is "<<sizeof(cA11)<< endl;  cout << "C-string length Function\n";  cout <<"Length of cA is " <<strlen(cA)<< endl;  cout <<"Length of cA1 is "<<strlen(cA1)<< endl;  cout << "Length of cA11 is " << strlen(cA11) << endl;  } | **OUTPUT** |

Does the length of all variables is same as you expected? If not, mention the variable name whose output is different what you were expecting. Can you fix it.

|  |
| --- |
|  |

1. Perform following tasks on the code given in part A:
2. Implement the above-mentioned tasks of part A using functions. In main(), call only functions i.e. find\_size() , find\_length(), find value().
3. Copy “data” variable data to “final” variable. Use C-string concatenation to make word “banana”, only by using the variables declared above. Store it in “data” character array. Implement it using function.
4. Ask user his favorite fruit and concatenate it in “final” character array. Compare both C-strings and tell which one is greater. Also output the lengths of both character arrays.
5. Implement all parts above, without using any of the built in C-string function. Make your own function for each attribute.
6. Ask user to enter a sentence. Remove all numbers entered by user from the sentence. Implement this program, using C-string i.e. character array. Store the numbers and alphabets in separate C-string respectively.

**Bonus Marks**: For concatenation of number and words in final strings, make your own function similar to str\_cat().

1. Find the frequency of a character “a” entered by user in a C-string. Instead of using default “strlen” function, make your own function.

**Practice Task:**

1. Write a program that can print complete names of student, already stored in two different text files. One text file has first name of all students and second file has last name. Print final output on console. For concatenation of strings, make your own functions with name concatenate\_string().