Lab 3:

Strings in C++

Handling of Character String

A string is a sequence of characters. Any sequence or set of characters defined within double quotation symbols is a constant string. In C++ it is required to do some meaningful operations on strings they are:

- Reading string
- Displaying strings
- Combining or concatenating strings
- Copying one string to another.
- Comparing string & checking whether they are equal
- Extraction of a portion of a string

Strings are stored in memory as ASCII codes of characters that make up the string appended with '\0' (ASCII value of null). Normally each character is stored in one byte; successive characters are stored in successive bytes.

Character	m	y		a	g	e		i	s
ASCII Code	77	121	32	97	103	10	32	105	115

Character		2		(	t	w	o	)	\0
ASCII Code	32	50	32	40	116	119	41	0	0

Arrays of Characters

Re-Introduction to Characters

As it happens, strings are the most used items of computers. In fact, anything the user types is a string. It is up to you to convert it to another, appropriate, type of your choice. This is because calculations cannot be performed on strings. On the other hand, strings can be a little complex,

which is why we wanted to first know how to use the other types and feel enough comfortable with them.

Consider a name such as James. This is made of 5 letters, namely J, a, m, e, and s. Such letters, called characters, can be created and initialized as follows:

char L1 = 'J', L2 = 'a', L3 = 'm', L4 = 'e', L5 = 's';

To display these characters as a group, you can use the following:

cout << "The name is " << L1 << L2 << L3 << L4 << L5;

Here is such a program:

Example 2.3:

```
#include <iostream>
using namespace std;

int main()

{
   char L1 = 'J', L2 = 'a', L3 = 'm', L4 = 'e', L5 = 's';

   cout << "The name is " << L1 << L2 << L3 << L4 << L5;

      return 0;

}
```

This would produce:

Declaring and Initializing an Array of Characters

When studying arrays, we were listing the numeric members of the array between curly brackets. Here is an example:

int Number[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

Because a character is initialized by including it in single-quotes, when creating an array of characters, to initialize it, you must also include each letter accordingly. A name such as James can be initialized as follows:

char Name[6] = { 'J', 'a', 'm', 'e', 's' };

As done with other arrays, each member of this array of characters can be accessed using its index. Here is an example:

Example 3.1:

```
#include <iostream>
using namespace std;
int main()

{
   char Name[6] = { 'J', 'a', 'm', 'e', 's' };

   cout << "The name is " << Name[0] << Name[1] << Name[2] << Name[3] << Name[4];

      return 0;
}
```

The C/C++ provides another alternative. It allows you to declare and initialize the array as a whole. To do this, include the name in double-quotes. With this technique, the curly brackets that delimit an array are not necessary anymore. Here is an example:

char Name[12] = "James";

With this technique, the item between the double-quotes is called a string. It is also referred to as the value of the string or the value of the variable.

When declaring and initializing an array of characters, the compiler does not need to know the number of characters of the string. In fact, you can let the compiler figure it out. Therefore, you can leave the square brackets empty:

char Name[] = "James";

After declaring such an array, the compiler would count the number of characters of the variable, add one more variable to it and allocate enough space for the variable. The character added is called the null-terminated character and it is represented as \0. Therefore, a string such as James would be stored as follows:

J	a	m	e	s	\0

Streaming an Array of Characters

Like any other variable, before using a string, you must first declare it, which is done by type the char keyword, followed by the name of the variable, followed by square brackets. When declaring the variable, if/since you do not know the number of characters needed for the string; you must still provide an estimate number. You can provide a value large enough to accommodate the maximum number of characters that would be necessary for the variable. For a person's name, this could be 20. For the title of a book or a web page, this could be longer. Here are examples:

char Name[20];
char BookTitle[40];
char WebReference[80];
char WeekDay[4];

To request the value of an array of characters, use the **cin** extractor just like you would proceed with any other variable, without the square bracket. Here is an example:

char WeekDay[12];
cout << "Enter today's name: ";
cin >> WeekDay;

To display the value of an array of characters, use the **cout** extractor as we have used it with all other variables, without the square brackets. Here is an example:

Example 3.2:

```
#include <iostream>

using namespace std;

int main()

{
   char WeekDay[12];
   char EndMe[] = "\n";
   cout << "Enter today's name: ";
   cin >> WeekDay;
   cout << "Today is " << WeekDay;
   cout << EndMe;
    return 0;

}
```

Here is an example of running the program:

Enter today's name: Thursday

Today is Thursday

Multidimensional Arrays of Characters

C/C++ treats arrays of characters differently than it does the other arrays. For example, we have learned to declare a two-dimensional array of integers as follows:

```
int Number[2][6] = { { 31, 28, 31, 30, 31, 30 },
                     { 31, 31, 30, 31, 30, 31 } };
```

This variable is in fact two arrays and each array contains 6 integers. For a string, if you want to declare a two-dimension array of characters, the first dimension specifies the number of string in the variable. The second dimension specifies the number of characters that each string can hold. Here is an example:

```
char StudentName[4][10] = { "Hermine", "Paul", "Gertrude", "Leon" };
```

In this case, the StudentName variable is an array of 4 strings and each string can have a maximum of 9 characters (+1 for the null-terminated character). To locate a string on this variable, type the name of the array followed by its index, which is the index of the column. This means that the first string of the StudentName array can be accessed with StudentName[0]. The second would be StudentName[1], etc. This allows you to display each string on the **cout** extractor:

Example 3.3:

```
#include <iostream>

using namespace std;
int main()

{ char StudentName[4][10] = { "Hermine", "Paul", "Gertrude", "Leon" };
   cout << "Student Names";
   cout << "\nStudent 1: " << StudentName[0];
   cout << "\nStudent 2: " << StudentName[1];
   cout << "\nStudent 3: " << StudentName[2];
   cout << "\nStudent 4: " << StudentName[3];
      return 0;
}
```

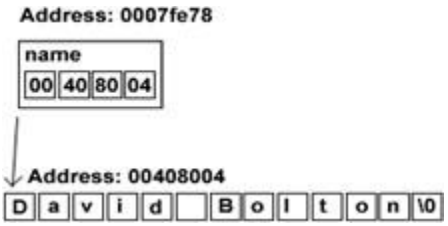
This would produce:

```
Student Names
```

Student 1: Hermine
Student 2: Paul
Student 3: Gertrude
Student 4: Leon

When declaring and initializing such an array, the compiler does not need to know the number of strings in the array; it can figure it out on its own. Therefore, you can leave the first pair square brackets empty. If you are only declaring the array but cannot initialize, then you must specify both dimensions.

Declaring a Pointer to Characters



Pointers can be quite challenging to learn. The diagram should provide a better insight.

The variable name (a pointer to a char) was located at address 0x007fe78. A pointer is 32 bits, ie. 4 bytes long and holds an address of the string which was 0x00408004. Addresses are always given in hexadecimal. The arrow shows what the pointer is pointing to, ie what is at the address in the pointer variable.
At this address there were 13 bytes holding the string "David Bolton" with a terminating NULL, the same value as '\0'.

Earlier, we declared an array as follows:

```
char EndMe[] = "";
```

The name of the variable is a pointer to the beginning of the array. For this reason, the name of the reason is sufficient to locate its value. Since in this case we do not specify the number of characters in the array, we can also just use a pointer to the array as the beginning of the array. The array can therefore be declared as follows:

```
char *EndMe = "";
```

Once again, to display the value of such an array, simply call its name on the cout extractor:

```
#include <iostream>
using namespace std;


int main()

{
```

```
    char *EndMe = "";
    cout << EndMe;
      return 0;

}
```

To request the value of an array of characters from the user, you can declare a pointer to char and initialize it with an estimate number of characters using the new operator. Here is an example:

Example 3.4:

```
#include <iostream>
using namespace std;

int main()

{

    char *StudentName = new char[20];
    cout << "Enter Sudent First Name: ";
    cin >> StudentName;
    cout << "\nStudent First Name: " << StudentName;
     return 0;

}
```

Here is an example of running the program:

Enter Sudent First Name: David
Student First Name: David

Arrays of Characters and Pointers

```
#include  <iostream>
using namespace std;
int main()

{
            double *Value;
            Value = 12.55;
            cout << "Value = " << Value;
            return 0;

}
```

On the other hand, the following declaring of an array of characters and its later initialization is perfectly legal:

Example 3.8:

```
#include  <iostream>
using namespace std;
int main()
{
            char *Country;
            Country = "Republique d'Afrique du Sud";
            cout << "Country Name: " << Country << "\n\n";
            return 0;

}
```

String Manipulation Functions

Strings as Variables

• To create a variable of type string, simply:

string fullName;

• Assignment, as always is the same:

fullName = “Aamir Hassan”;

• Or like before, we could always combine the two with an initialization:

string fullName = “Aamir Hassan”;

Input/output with Strings

• I/O with Strings is as before:

string fullName = “”;

cout << “Please enter your name: “;

cin >> fullName; //get the name

cout << “Your name is “ << fullName;

String Operators: Assignment

• Assignment (=): As with before, assign a string to a variable of type string.

string oneName = “Edwin”;

string anotherName = oneName;

• Both now hold “Edwin”

String Operators: Concatenation

• Concatenation (+): Puts a string on the end of another.

string firstName = “Edwin”;

string lastName = “Dreese”;

string fullName = firstName + “ ” + lastname;

//the += shorthand also works

string oneString = “2 + 2 = ”;

oneString += “5”;

Relational String Operators

• == and != are same as before, but the others are not exactly like usage with numbers…

• For instance, what exactly does it mean if one string is “less than” another?

String I/O

• A common problem with reading strings from user input is that it could contain white spaces. Remember that white space (e.g. space, tab, newline) is treated as termination for cin.

• Take the following code for example:

```
cout << “Enter your full name: ”;

string fullname;

//only the first name will be read in!!

cin >> fullname;
```

String Processing

• In addition to giving us a new data type to hold strings, the string library offers many

useful string processing methods.

• You can find most of them of them in the book, but here are a few useful ones.

length () and size()

• This method returns the integer length of the string. The length() and size() are the same.

• Example:

```
string s1 = “Super!”;

//the integer 6 will be output.

cout << s1.length() << endl;
```

at(index)

• This method returns the character at the specified index. Indices start from 0.

• Example:

```
string n = “Vikram”;

//the character ‘V’ will be output.

cout << n.at(0) << endl;
```

```
//the character 'm' will be output.

cout << n.at(n.size()-1) << endl;
```

Shorthand for at (index)

• As an alternative, we could have also used the following equivalent shorthand:

```
string n = "Vikram";

//the character 'V' will be output.

cout << n[0] << endl;

//the character 'm' will be output.

cout << n[n.size()-1] << endl;
```

erase(index)

• This method removes all characters from the string starting from the specified index to the end.

• The length of the new string is reset to index!

• Example:

```
string os = "Operating Systems";

os.erase(9);

//the string "Operating" is output

cout << os << endl;

//length is now 9, the index

cout << os.length() << endl;
```

find(str)

• This method returns the integer index of the first occurrence of the specified string

• Example:

```
string d = "data data data";

//0 is output

cout << d.find("data") << endl;
```

find(str, index)

• This method returns the integer index of the first occurrence of the specified string starting from the specified index.

• Returns -1 if pattern is not found.

• Example:

string d = “data data data”;

//5 is output

cout << d.find(“data”, 1) << endl;

• Why? Because by specifying a starting index of 1, we only consider

“ata data data”

insert(index, str)

• Inserts the specified string at the specified index.

• Example:

string animal = “Hippo”;

animal.insert(0, “Happy ”);

//outputs “Happy Hippo”

cout << animal << endl;

replace(index, n, str)

• Removes n characters in the string starting from the specified index, and inserts the specified

string, str, in its place.

• Example:

string transport = “Speed Boat”;

transport.replace(0, 5, “Sail”);

//outputs “Sail Boat”

cout << transport << endl;

substr(index, n)

• Returns the string consisting of n characters starting from the specified index.

• Example:

string transport = “Cruise Ship”;

//outputs “Ship”

cout << transport.substr(7, 4) << endl;

Some String Functions

- strlen()
- strcpy()
- strncpy()
- strcat()
- strncat()
- strcmp()
- strncmp()

Strlen ()

Syntax:

len = strlen(ptr);
where len is an integer and
ptr is a pointer to char

strlen() returns the length of a string, excluding the null. The following code will result in len having the value 13.

```
int len;
char str[15];

strcpy(str, "Hello, world!");
len = strlen(str);
```

Example 3.9:

```
#include <iostream>

#include <cstring>

using namespace std;

int main()

{
```

```
 int len;

char str[15];

strcpy(str, "Hello, world!");

len = strlen(str);

cout<<len<<endl;

  system("PAUSE");

 return 0;

}
```

Strcpy ()

strcpy(ptr1, ptr2);
  where ptr1 and ptr2 are pointers to char

strcpy() is used to copy a null-terminated string into a variable. Given the following declarations, several things are possible.

char S[25];
char D[25];

- Putting text into a string:
- strcpy(S, "This is String 1.");
- Copying a whole string from S to D:
- strcpy(D, S);
- Copying the tail end of string S to D:
- strcpy(D, &S[8]);

N.B. If you fail to ensure that the source string is null-terminated, very strange and sometimes very ugly things may result.

Strncpy ()

  strncpy(ptr1, ptr2, n);
   where n is an integer and
   ptr1 and ptr2 are pointers to char

strncpy() is used to copy a portion of a possibly null-terminated string into a variable. Care must be taken because the '\0' is put at the end of destination string only if it is within the part of the string being copied. Given the following declarations, several things are possible.

char S[25];

char D[25];

Assume that the following statement has been executed before each of the remaining code fragments.

- Putting text into the source string:
- strcpy(S, "This is String 1.");
- Copying four characters from the beginning of S to D and placing a null at the end:
- strncpy(D, S, 4);
- D[4] = '\0';
- Copying two characters from the middle of string S to D:
- strncpy(D, &S[5], 2);
- D[2] = '\0';
- Copying the tail end of string S to D:
- strncpy(D, &S[8], 15);

  Which produces the same result as strcpy(D, &S[8]);

Strcat ()

strcat(ptr1, ptr2);
where ptr1 and ptr2 are pointers to char

strcat() is used to concatenate a null-terminated string to end of another string variable. This is equivalent to pasting one string onto the end of another, overwriting the null terminator. There is only one common use for strcat().

Char S[25] = "world!";
Char D[25] = "Hello, ";

- Concatenating the whole string S onto D:
- strcat(D, S);

N.B. If you fail to ensure that the source string is null-terminated, very strange and sometimes very ugly things may result.

Strncat ()

Syntax: strncat(ptr1, ptr2, n);
where n is an integer and
ptr1 and ptr2 are pointers to char

strncat() is used to concatenate a portion of a possibly null-terminated string onto the end of another string variable. Care must be taken because some earlier implementations of C do not append the '\0' at the end of destination string. Given the following declarations, several things are possible, but only one is commonly used.

Char S[25] = "world!";

Char D[25] = "Hello, ";

- Concatenating five characters from the beginning of S onto the end of D and placing a null at the end:
- strncat(D, S, 5);
- strncat(D, S, strlen(S) -1);

  Both would result in D containing "Hello, world".

N.B. If you fail to ensure that the source string is null-terminated, very strange and sometimes very ugly things may result.

Strcmp ()

Syntax: diff = strcmp(ptr1, ptr2);
where diff is an integer and
ptr1 and ptr2 are pointers to char

strcmp() is used to compare two strings. The strings are compared character by character starting at the characters pointed at by the two pointers. If the strings are identical, the integer value zero (0) is returned. As soon as a difference is found, the comparison is halted and if the ASCII value at the point of difference in the first string is less than that in the second (e.g. 'a' 0x61 vs. 'e' 0x65) a negative value is returned; otherwise, a positive value is returned. Examine the following examples.

char s1[25] = "pat";
char s2[25] = "pet";

diff will have a negative value after the following statement is executed.

diff = strcmp(s1, s2);

diff will have a positive value after the following statement is executed.

diff = strcmp(s2, s1);

diff will have a value of zero (0) after the execution of the following statement, which compares s1 with itself.

diff = strcmp(s1, s1);

Strncmp ()

Syntax: diff = strncmp(ptr1, ptr2, n);
where diff and n are integers
ptr1 and ptr2 are pointers to char

strncmp() is used to compare the first n characters of two strings. The strings are compared character by character starting at the characters pointed at by the two pointers. If the first n strings are identical, the integer value zero (0) is returned. As soon as a difference is found, the comparison is halted and if the ASCII value at the point of difference in the first string is less than that in the second (e.g. 'a' 0x61 vs. 'e' 0x65) a negative value is returned; otherwise, a positive value is returned. Examine the following examples.

```
char s1[25] = "pat";
char s2[25] = "pet";
```

diff will have a negative value after the following statement is executed.

```
diff = strncmp(s1, s2, 2);
```

diff will have a positive value after the following statement is executed.

```
diff = strncmp(s2, s1, 3);
```

diff will have a value of zero (0) after the following statement.

```
diff = strncmp(s1, s2, 1);
```

Note: Before doing your lab exercises run the examples.

Exercises will be provided by the instructors in the lab.