Unit 6: Variables – Char and String

1. Introduction

We can declare string variables as follows:

```
string myVariable;
```

String variables can hold any characters such as letters and digits. For instance, you can put 'John' in a string variable, but not in an integer variable. Value assignment is same as integer variables:

```
myVariable = "John";
```

puts 'John' in myvariable. Similarly, we can initialize the value during decleration:

```
string myVariable = "John";
```

NOTE:

Don't forget the double quotes (") before and after `John`.

Example 1: Write a program that reads your name from the keyboard and prints it on the screen 5 times as 'Your name is John.' if the name is 'John'.

Sample:

input:	*output*:
John	Your name is John. Your name is John. Your name is John. Your name is John. Your name is John.

Solution:

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

int main()
{
  string name;
  cin >> name;
  for (int counter = 1; counter <= 5; ++counter)
  {
    cout << "Your name is " << name << "." << endl;
  }
}
```

Exercise 1: Write a program that reads your name and a number n from the keyboard and prints your name on the screen n times, shifting one space each time.

Sample:

input:	*output*:

Jane
5
	Jane
	 Jane
	  Jane
	  Jane
	  Jane

2. String Operations

We can also put numbers in strings. However, they are not values, and you cannot make arithmetic operations on them.

```
1.   #include <iostream>
2.   using namespace std;
3.
4.   int main()
5.   {
6.     string number1 = "21", number2 = "34";
7.     int sum;
8.     sum = number1 + number2;
9.     cout << sum << endl;
10.  }
```

In the above code, the compiler will give an error at line 8. You cannot add the numbers in strings because they are considered to be just a set of characters, not integer values.

On the other hand, we can do some operations with strings. We have already seen, how to assign a value in a string such as:

```
string name;
name = "Jane";
```

We can also use addition (+) and (+=) operators to concatenate strings:

```
string name = "Jane";
string str = "Hello " + name;
```

The string `str` will have `"Hello Jane"` in it.

Example 2: Write a program that reads a name and a number n from the keyboard and prints the name on the screen n times as follows:

Sample:

input:	*output*:
Jane 5	* Jane ! ** Jane !! *** Jane !!! **** Jane !!!! ***** Jane !!!!!

Solution:

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

int main()
{
  string name;
  int n;
  cin >> name >> n;
  name = " " + name + " ";
  for (int counter = 1; counter <= n; ++counter)
  {
    name = "*" + name + "!";
    cout << name << endl;
  }
}
```

The above code adds one more '*' in the front and '!' at the end of the string each time.

To check whether two strings are equal or not equal, we use '==' or '!=' operators as in integer variables.

Example 3: Write a program that reads two numbers and an arithmetic operator (+, –, *, /) from the keyboard. The program applies the operation to the numbers and prints the result on the screen:

Sample:

input:	*output*:
5 21 *	105

Solution:

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

int main()
{
  int num1, num2, result;
```

```
  string op;                  // operator

  cin >> num1 >> num2 >> op;
  if (op == "+") result = num1 + num2;
  if (op == "-") result = num1 - num2;
  if (op == "*") result = num1 * num2;
  if (op == "/") result = num1 / num2;
  cout << result << endl;
}
```

We can also use '<', '<=', '>=', and '>' operators to compare the alphabetical order of the strings.

```
if (string1 < string2)
  cout << "string1 is before string2";
else
  cout << "string1 is not before string2";
```

The code fragment above prints '`string1 is before string2`' if `string1` is alphabetically comes before `string2`. Suppose `string1` is '`try it`' and `string2` is '`trial`' then the program prints '`string1 is not before string2`'.

NOTE:

Be careful that capital letters come before lower-case letters in computers. For instance, if `string1` is '`Zinc`' and `string2` is '`about`' then the program prints '`string1 is before string2`' since 'Z' is before 'a'.

Exercise 2: Sort given three strings according to alphabetical order. Read input from *words.txt* and write the output to *sorted.txt*.

Sample:

input:	*output*:
`hello`	`Try`
`about`	`about`
`Try`	`hello`

3. Single Character Variables

There is another variable type called `char` that refers to only a single character. Character variable is declared as follows:

```
char myChar;
```

Same as the other variable types, we use '=' for assignment:

```
char myChar = 'J';
```

and '==' and '`!=`' for equality check:

```
if (myChar != 'J')
  cout << "The letter is not J!" << endl;
```

NOTE:

Be careful about the single quote (') before and after the character instead of double quote in strings. Single quote means a single character as a character type.

Example 4: Remember the example given above. Write a program that reads two numbers and an arithmetic operator (+, –, *, /) from the keyboard. We can write the same code using character variable instead of string since operator corresponds to a single character. The only difference is the declaration of the variable and the single quotes in the `if` statement condition checks.

Solution:

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

int main()
{
  int num1, num2, result;
  char op;

  cin >> num1 >> num2 >> op;
  if (op == '+') result = num1 + num2;
  if (op == '-') result = num1 - num2;
  if (op == '*') result = num1 * num2;
  if (op == '/') result = num1 / num2;
  cout << result << endl;
}
```

Similarly, '<', '<=', '>=', and '>' operators are used to compare the alphabetical order of the characters.

4. Strings as Character Arrays

We can in fact consider strings as an array of characters. That is, we can reach a specific character of a string similar to the arrays as follows:

```
cout << name[3] << endl;
```

The code above will print the 4th character of the string name on the screen. Note that the first character of the string is located at the position 0 – the same as arrays.

We can also change the character at a specific location in a similar way:

```
name[3] = 'z';
```

The code above will change the 4th character of the string to 'z'. Be careful about the single quote (') before and after the character since you refer to a single character at a specific location of a string.

5. String Functions

Strings have different useful functions. One of them is *length* function that gives the number of characters in a string. It can be used as follows:

```
string str = "Hello world!";
int theLength = str.length();
cout << theLength << endl;
```

In the code above, we wrote the name of the string variable then a dot and '`length()`' to get the size of the string. The code fragment above will write '12' on the screen (including the space and exclamation mark).

Example 5: Write a program that outputs the middle character of a string. If it doesn't have, write '`No middle character!`' on the screen.

Sample:

input:	*output*:
`Trial`	`i`

input:	*output*:
`Helloworld`	`No middle character!`

Solution:

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

int main()
{
  string str;

  cin >> str;

  // if the size of the string is even
  if (str.length() % 2 == 0)
    cout << "No middle character!" << endl;
  else
  {
    int middle = str.length() / 2;
    cout << str[middle] << endl;
  }
}
```

Exercise 3: Write a program reads a string from the keyboard and writes the reverse of it on the screen.

Sample:

input:	*output*:
`Trial`	`lairT`

Exercise 4: Write a program reads a string and an integer *n* from the keyboard and writes the reverse of it *n* times on the screen.

Sample:

input:	*output*:
`Trial` `5`	`lairT` `lairT` `lairT` `lairT` `lairT`

Exercise 5: Write a program reads a string *s* and three integers *a*, *b* and *n* from the keyboard, and writes the substring of *s* that starting from *a* th position and continues *b* characters, *n* times on the screen.

Sample:

input:	*output*:
`Helloworld!` `4 5 5`	`oworl` `oworl` `oworl` `oworl` `oworl`

In the example above, the substring starting from 4th position is '`oworl`'.

Strings have another function called *substr* provides the substring starting from a given position of the string. We can write the simplified code in the previous exercise as follows:

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

int main()
{
  string s, substring = "";
  int a, b, n;
```

```
  cin >> s >> a >> b >> n;
  substring = s.substr(a, b);

  // write the substring n times
  for (int counter = 1; counter <= n; ++counter)
  {
    cout << substring << endl;
  }
}
```

Here, '`s.substr(a, b)`' gives the substring of *s* starting from the a^{th} position and length of *b*.

Exercise 6: Write a program reads two string *s1*, *s2* and two integers *a*, *n* from the keyboard. Insert *s2* at *a* th position of *s1* then write the new string *n* times on the screen.

Sample:

input:	*output*:
`HelloWorld!`	`HelloNewWorld!`
`New 5 3`	`HelloNewWorld!`
	`HelloNewWorld!`

Strings have a function called *insert* that inserts a given string at the given position of the string. We can write the simplified code in the previous exercise as follows:

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

int main()
{
  string s1, s2;
  int a, n;

  cin >> s1 >> s2 >> a >> n;
  s1 = s1.insert(a, s2);

  // write the substring n times
  for (int counter = 1; counter <= n; ++counter)
  {
    cout << s1 << endl;
  }
}
```

Here, '`s1.insert(a, s2)`' inserts the string *s2* into the a^{th} position of *s1*.

Exercise 7: Write a program reads a string *s* and three integers *a*, *b*, *n* from the keyboard. Delete *b* characters in *s* starting at position *a* then write the new string *n* times on the screen.

Sample:

input:	*output*:
`HelloWorld!`	`Herld!`
`2 5 3`	`Herld!`
	`Herld!`

Strings have a function called *erase* that erases given number of characters in a given string starting from the given position of the string. We can write the simplified code in the previous exercise as follows:

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

int main()
{
  string s;
  int a, b, n;

  cin >> s >> a >> b >> n;
  s = s.erase(a, b);

  // write the substring n times
  for (int counter = 1; counter <= n; ++counter)
  {
    cout << s << endl;
  }
}
```

Here, '`s.erase(a, b)`' erases *b* character starting from the position *a*.

Exercise 8: Write a program reads two strings *s1*, *s2* and three integers *a*, *b*, *n* from the keyboard. Replace *b* characters in *s1* starting at position *a* with the string *s2* then write the new string *n* times on the screen.

Sample:

input:	*output*:
`HelloWorld!`	`NewWorld!`
`New 0 5 3`	`NewWorld!`
	`NewWorld!`

Strings have a function called *replace* that makes the replacement given above. We can write the simplified code in the previous exercise as follows:

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

int main()
{
  string s1, s2;
  int a, b, n;

  cin >> s1 >> s2 >> a >> b >> n;

  s1 = s1.replace(a, b, s2);

  // write the substring n times
  for (int counter = 1; counter <= n; ++counter)
  {
    cout << s1 << endl;
  }
}
```

Here, '`s1.replace(a, b, s2)`' replaces *b* character starting from the position *a* with the string *s2*.

Exercise 9: Write a program reads two strings *s1*, *s2* from the keyboard then writes the position of s2 in s1 if s2 is found in s1. Otherwise it writes '`not found!`'.

Sample:

input:	*output*:
`HelloWorld!` `loW`	`3`

input:	*output*:
`HelloWorld!` `Low`	`not found!`

NOTE:

Be careful that strings are case sensitive as shown in the sample input/output above.

Strings have a function called *find* that search a string in a given string. We can write the simplified code in the previous exercise as follows:

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



```
int main()
{
  string s1, s2;
  int found;

  cin >> s1 >> s2;
  found = s1.find(s2);

  if (found == -1)
    cout << "not found!" << endl;
  else
    cout << found << endl;
}
```

Here, '`s1.find(s2)`' searches for *s2* in the string *s1*. If *s2* is found in *s1*, it gives the position in *s1*, otherwise it gives '-1'.

6. Array of Strings

Remember that we declared arrays that are composed of integer values:

```
int myIntArray[10];
```

declares an array of 10 integers. Similarly, we can declare arrays with string values:

```
string myStrArray[10];
```

declares an array of 10 strings. We can read from and write to the array locations in the same way.

Example 6: Read an integer *n* then *n* words from the input, and write the words in reverse order. There will be maximum 100 strings in the input file.

Sample:

input:	*output*:
4	About
HelloWorld!	TryIt!
Hello	Hello
TryIt!	HelloWorld!
About	

Solution:

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

int main()
{
  int n;
  string words[100];
```

```
  // read n words from the input
  cin >> n;
  for (int counter = 0; counter < n; ++counter)
  {
    cin >> words[counter];
  }
  // write words on the screen in reverse order
  for (int counter = n - 1; counter >= 0; --counter)
  {
    cout << words[counter] << endl;
  }
}
```

A specific position of a string in a string array can be reached as follows:

```
cout << myStrings[5][3] << endl;
```

prints the character at position 3 of string 5 in myStrings array on the screen.

String functions can be also used in string arrays. For instance;

```
cout << myStrings[5].substr(3, 6) << endl;
```

prints the substring that starts at position 3 and has length 6 of string 5 in myStrings array on the screen.

Exercise 10: Write a program reads an integer *n* then *n* words and a string *s* from the input. Erase *s* from any word it contains *s*. Write the new words on the screen. There will be maximum 100 strings in the input file.

Sample:

input:	*output*:
`4`	`HelWorld!`
`HelloWorld!`	`wer`
`lower`	`trial`
`trial`	`Hel`
`Hello`	
`lo`	

The string '`lo`' is erased from the words '`HelloWorld!`', '`lower`', and '`Hello`'.

