The C++ Static Arrays Data Structure and The C++ String Data Type

In this week

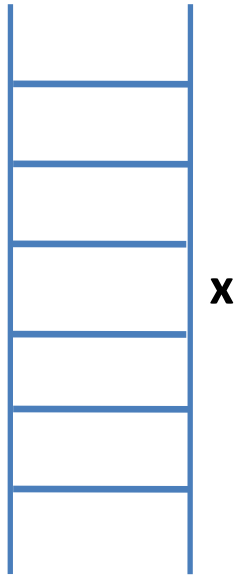
- What are C++ Static Arrays?
- Declaring, Initializing and Processing Arrays
- Passing Arrays to C++ functions
- Searching C++ Arrays
- C++ String data type

Introduction

- So far all the variables we have looked at could store **ONLY** one value at a time
- For example, in the declaration

  double x;

  the variable x can hold only one double value at a time
- Yes, we can always modify and re-modify the value of the variable **x**, but then the moment we modify its value; its old value will be completely deleted and no more available
- This makes all the variables we have seen so far to be limited to be able to hold only one value at a time
- But what if we want a variable that can hold, let's say 100 different double values, at the same time?
- This is where the concept of C++ arrays comes in!!!

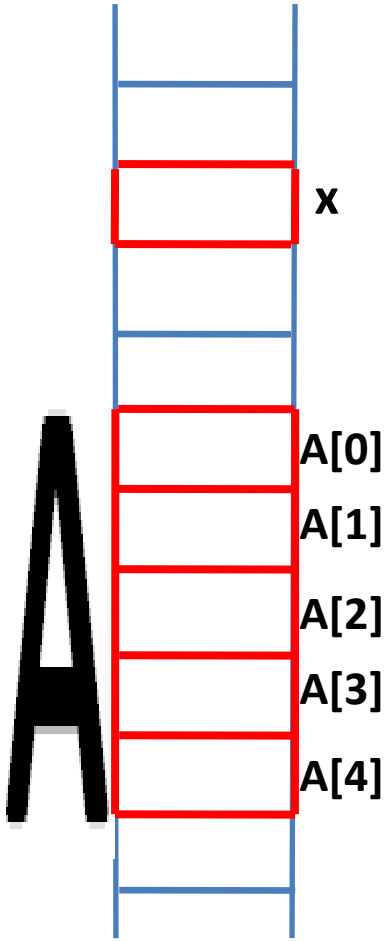


What are C++ Static Arrays?

double x; //double type variable

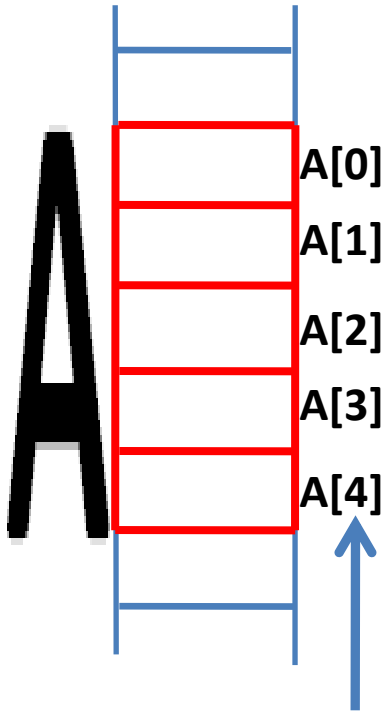
double A[5]; //double type ARRAY

- C++ Static Arrays are consecutive memory locations that share a common name, that hold certain predefined number of elements of the same data type, and whose elements are accessed via the array name and an index placed within square brackets
- The index starts at **0** for the first element, it is **1** for the second element, it is **2** for the third element, and so on so forth



What are C++ Static Arrays?

double A[5];



Size of the array (int data type)

Index of an element (int data type)

Static Array Declaration

- As shown above, an **Array** consisting of say **five** elements of type **double** is declared as follows

  double A[5];

- In the same way, an **Array** consisting of **ten** elements of type **float** is declared as follows:

  float B[10];

- Similarly an **Array** consisting of **hundred** elements of type **char** is declared as:

  char arr[100];

- Also, an **Array** consisting of one **thousand** elements of **int** is declared as:

  int res[1000];

- Finally, an **Array** consisting of a **million** elements of type **boolean** is declared as:

  bool ans[1000000];

Static Array Declaration

- In C++ the size of static arrays, that is the number of elements the array can hold, is defined at the declaration time
- The array size can not be changed once the array is declared
- The collection of data to be stored in the array must be of the same data type and must match the array declaration data type

Static Array Declaration

- In C++, the **size** of a static array can be either a **literal value** as shown earlier or an integer type **named constant** as follows:

```
const int SIZE= 30;
float  S[SIZE];
```

- Now **S** is a C++ static array with size **30** and can hold up to 30 float data
- Here the **SIZE** variable must be an integer data type **named constant** otherwise it will be a syntax error. See the following examples

```
int arraySize = 10;
int A[arraySize]; ← ← ← ← ← SYNTAX ERROR
```

OR

```
int mySize;
cout << "Enter size: ";
cin >> mySize;
int B[mySize]; ← ← ← ← ← SYNTAX ERROR
```

Processing C++ Static Arrays

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

int main()
{
    //Create an array of type int of size 10
    int A[10];

    //Start the random number generator
    srand(time(0));

    //Populate the array with random integers from [5, 25]
    for (int i = 0; i < 10; i++)
        A[i] = rand() % 21 + 5;

    //Print the elements of the array
    for (int i = 0; i < 10; i++)
        cout << "Element at index " << i << " = " << A[i] << endl;

    //Print the sum of all the elements of the array
    int s = 0;
    for (int i = 0; i < 10; i++)
        s += A[i];
    cout << "The sum of all the elements of the array is " << s << endl;
```

Processing C++ Static Arrays

```
    //Print how many even and how many odd elements there are in the array
    int e = 0, o = 0;
    for (int i = 0; i < 10; i++)
        A[i] % 2 == 0 ? e++ : o++;
    cout << "The array has " << e << " even and " << o << " odd elements" << endl;

    //Re-assign each element of the array with a user input value
    for (int i = 0; i < 10; i++)
    {
        cout << "Enter a value for the element at index " << i << ": ";
        cin >> A[i];
    }

    //Print the new elements of the array
    for (int i = 0; i < 10; i++)
        cout << "Element at index " << i << " = " << A[i] << endl;

    //Print the minimum element of the array
    int minimum = A[0];
    for (int i = 1; i < 10; i++)
        if (A[i] < minimum)
            minimum = A[i];
    cout << "The minimum element is " << minimum << endl;

    system("Pause");
    return 0;
}
```

C++ Static Arrays Definition

- In the previous example, we first declared an array of size 10 and populated/filled the array either with random integers or from user input data
- If the array data is available during the writing of the code, then we can **define** (declare and initialize) the array as follows:

float A[] = {5.4, -7.3, 2.0, 1.05, -6.54};

- Now A is a array of float of size 5. The size is determined by C++ based on the number of data.

C++ Static Arrays Definition

- The following C++ code segment demonstrates defining an array of float with 5 float data and then printing the elements of the array

```
int main()
{
    //Define an array
    float B[] = {5.4, -7.3, 0.2, 1.05, -6.54};

    //Print the elements of the array
    for (int i = 0; i < 5; i++)
        cout << "Element at index " << i << " is " << B[i] << endl;

    system("Pause");
    return 0;
}
```

Rules for C++ Static Array Definitions

- **Syntax 1: Data size determines array size**

  int A[] = {4, 9, 15, 0, 8};

  This creates an array of five integers and initializes them with the given data. The size 5 is computed by C++
- **Syntax 2: Both data and size specified**

  int A[5] = {4, 9, 15, 0, 8};

  This creates an array of five integers and initializes them with the given data
- **Syntax 3: Partial Initialization and Default Initialization**

  int A[5] = {3, 4, 5};

  This creates an array of five integers, initializes A[0], A[1], and A[2] with the given data. The remaining elements (A[3] and A[4]) are initialized to zero.
- **Syntax 4: Default initialization**

  int A[5] = {};

  This creates an array of five integers and initializes each element to zero.
- **Syntax 5: ERROR!**

  int A[5] = {1, 2, 3, 4, 5, 6, 7, 8};

  This is syntax **ERROR** because the number of the data supplied must not exceed the specified size.

C++ Static Arrays

INDEX OUT OF BOUND ERROR

- A static array **A** declared with a size, say 5, will have five elements which are **A[0], A[1], A[2], A[3],** and **A[4]**
- Attempting to access an element with **index less than 0 or index greater than 4** such as A[5] or A[6] or A[-1] or A[-3] will result to an index out of bound error and your program will crash!
- Try the following code snapshot and see what happens

```
int A[5];
for (int i = 0; i < 5; i++)
    A[i] = 3*i;
```

- **So far the program is ok! But in the following part of the code, it will get into an index out of bound error!**

```
for (int i = 0; i < 10000; i++)
    cout << "Element at index " << i << " = " << A[i] << endl;
```

Passing Array Elements to Functions

- The following example demonstrates passing elements of an array to a function by value. The function doubles the elements. **But the Array is not affected, why?**

```
void doubleArrayElements(float a, float b, float c, float d, float e)
{
    a = a * 2;
    b *= 2;
    c *= 2;
    d *= 2;
    e *= 2;
    return;
}
int main()
{
    //Define an array
    float A[] = {0.1, 0.2, 0.3, 0.4, 0.5};

    //Print the elements of the array
    cout << "Originally the elements of the array are"<< endl;
    for (int i = 0; i < 5; i++)
        cout << "Element at index " << i << " is " << A[i] << endl;

    //Call a function that will double each element of the array
    doubleArrayElements(A[0], A[1], A[2], A[3], A[4]);

    //Print the elements of the array
    cout << "The elements of the array after doubling each element are" << endl;
    for (int i = 0; i < 5; i++)
        cout << "Element at index " << i << " is " << A[i] << endl;

    system("Pause");
    return 0;
}
```

Passing C++ Static Arrays to Function

- Similarly, we can pass an array variable to a function as an argument
- However, an array argument passes to a function **NOT by value**
- Rather a static array is always passed by a special kind of parameter passing that makes it possible to change its values permanently inside functions
- Therefore any change made to a static array inside a function **ALSO CHANGES THE ARRAY IN THE MAIN PROGRAM!!!**

Passing C++ Static Arrays to Function

- The syntax for static array function argument and static array function parameter is as follows:
- **Function Parameter Declaration Syntax**

```
void doubleArrayElements(float X[ ], int size)
{
        //Function body
}
```

- Function Call Syntax

```
float A[] = {0.1, 0.2, 0.3, 0.4, 0.5};
doubleArrayElements(A, 5);
```

Passing C++ Static Arrays to Function

- The following code demonstrates array passing to functions

```
void doubleArrayElements(float X[], int size)
{
    for (int i = 0; i < size; i++)
        X[i] = 2 * X[i];
    return;
}

int main()
{
    //Define a static array
    float A[] = {0.1, 0.2, 0.3, 0.4, 0.5};

    //Print elements of the array
    cout << "Originally the elements of the array are" << endl;
    for (int i = 0; i < 5; i++)
        cout << A[i] << "   ";
    cout << endl;

    //Double the elements of the array
    doubleArrayElements(A, 5);

    cout << "After the function call the elements of the array are" << endl;
    //Print elements of the array
    for (int i = 0; i < 5; i++)
        cout << A[i] << "   ";
    cout << endl;

    system("Pause");
    return 0;
}
```

Observe the [ ] to indicate an array parameter

0.1 0.2 0.3 0.4 0.5

0.2 0.4 0.6 0.8 1.0

The const Modifier

- Now consider the following program that calls a function in order to count the number of even elements in an array. Does the program have any syntax error? Runtime error? Semantic error?

```
int countEven(int arr[], int size)
{
    //Let's not use the modulo operator. Instead let's do the actual math for checking even.
    //Note that an integer value a is even if the value of a is equal to 2*(a/2)
    int count = 0;
    for (int i = 0; i < size; i++)
    {
        if (arr[i] = 2*(arr[i]/2))
            count++;
    }
    return count;
}
int main()
{
    //Define an array
    int A[] = {1, 2, 3, 4, 5};
    //Print the elements of the array
    cout << "Originally the elements of the array are" << endl;
    for (int i = 0; i < 5; i++)
        cout << A[i] << endl;
    //Count and print how many even number elements there are in the array
    int m = countEven(A, 5);
    cout << "There are " << m << " even numbers in the array." << endl;
    //Print the elements of the array
    cout << "After the function call, the elements of the array are" << endl;
    for (int i = 0; i < 5; i++)
        cout << A[i] << endl;

    system("Pause");
    return 0;
}
```

The const Modifier

- In the previous example, not only are we getting semantically wrong results but also we are making unintended modifications to the elements of the array
- In such instances, we can **prefix the array parameter** in the function declaration with **constant** modifier and then the compiler will report a compilation error if we try to modify the array inadvertently
- The previous program with a constant modifier for the for the parameters is shown below
- This time it will have syntax error
- Once we fix the syntax error, the program will work correctly as expected

The const Modifier

```
int countEven(const int arr[], const int size)
{
    //Let's not use the modulo operator. Instead let's do the actual math for checking even.
    //Note that an integer value a is even if the value of a is equal to 2*(a/2)
    int count = 0;
    for (int i = 0; i < size; i++)
    {
        if (arr[i] = 2*(arr[i]/2))
            count++;
    }
    return count;
}
int main()
{
    //Define an array
    int A[] = {1, 2, 3, 4, 5};
    //Print the elements of the array
    cout << "Originally the elements of the array are" << endl;
    for (int i = 0; i < 5; i++)
        cout << A[i] << endl;
    //Count and print how many even number elements there are in the array
    int m = countEven(A, 5);
    cout << "There are " << m << " even numbers in the array." << endl;
    //Print the elements of the array
    cout << "After the function call, the elements of the array are" << endl;
    for (int i = 0; i < 5; i++)
        cout << A[i] << endl;

    system("Pause");
    return 0;
}
```

Passing C++ Static Arrays to Functions
Complete Example

```
void populateArray(float arr[], const int size)
{
    for (int i = 0; i < size; i++)
        arr[i] = (1.0 * rand()) / RAND_MAX * 2 - 1;
}
int countNegative(const float myArray[], const int s)
{
    int count = 0;
    for (int i = 0; i < s; i++)
        count += myArray[i] < 0 ? 1 : 0;
    return count;
}
void printArray(const float A[], const int mySize)
{
    for (int i = 0; i < mySize; i++)
        cout << A[i] << endl;
    int num = countNegative(A, mySize);
    cout << "The array contains " << num << " negative elements" << endl;
    cout << "The array contains " << mySize-num << " non-negative elements" << endl;
}
void doubleArrayElements(float X[], const int arraySize)
{
    for (int i = 0; i < arraySize; i++)
        X[i] = 2 * X[i];
}
int main()
{
    //Declare an array
    const int SIZE = 5;
    float X[SIZE];
    //Populate the array
    populateArray(X, SIZE);
    //Print the array
    cout << "Originally, the array elements are" << endl;
    printArray(X, SIZE);
    //Double the elements of the array
    doubleArrayElements(X, SIZE);
    //Print the array
    cout << "After doubling the array elements, the array elements are" << endl;
    printArray(X, SIZE);
    system("Pause");
    return 0;
}
```

A complete C++ main program along with several functions designed to create an array, populate the array, print the array and modify the array elements... as well as demonstrate constant modifier for variables and parameters.

Searching C++ Arrays

- Consider the following problem: We would like to write a function that takes an array, its size and a value as arguments; and we would like the function to search the array if it contains the value.
- If the value is found in the array, we would like to return its index; otherwise we would like to return -1 to mean the value is not found in the array.
- See the following test program

Searching C++ Arrays

```
int main()
{
    //Create an array
    const int myArraySize = 15;
    int X[myArraySize];
    //Fill the elements of the array
    for (int i = 0; i < myArraySize; i++)
    {
        X[i] = rand() % 20;
    }
    //Print the elements
    for (int i = 0; i < myArraySize; i++)
    {
        cout << X[i] << "   ";
    }
    cout << endl;
    //Search a value in the arrauy
    int s = 4;
    int m = sequentialSearch(X, myArraySize, s);
    if (m == -1)
        cout << s << " is not found in the array." << endl;
    else
        cout << s << " is found in the array at index " << m << endl;
    system("pause");
    return 0;
}
```

Searching C++ Arrays

- The function will search the array sequentially (one by one) starting from the first element going all the way to the last element. When an element matches the search value then the function will return the index of the matching element. If no element matches the search value, then the function will return -1.

```
int sequentialSearch(const int A[], const int size, const int searchValue)
{
    for (int i = 0; i < size; i++)
    {
        if (A[i] == searchValue)
        {
            //We found an element matching the search value
            //Therefore we return the index i
            return i;
        }
        else
        {
            //We have not yet found the search value
            //Therefore we continue searching
            continue;
        }
    }
    //The for loop has finished. We did not return i, means we didn't find
    //the search value. Therefore we return -1.
    return -1;
}
```

Rules of Assignment for Static Array

- A static array variable can not be re-assigned
- Example:- Given

```
int A[3] = {1,2,3};
int B[3] = {4,5,6};
```

- Then the assignment statement

```
A = B;   //Syntax Error
```

is **syntax error** because **A** can not be re-assigned

Partial Arrays

- Sometimes, we may not know in advance the exact number of values we are going to work with and therefore we may not know what size array to create beforehand
- In such cases, it is customary to create as big as reasonably possible array and then fill it partially with the values we have to work with
- For example, you may allocate an array of size 100 integers and then fill it only 64 elements and work with the 64 elements only
- Of course you can't fill the array with more than 100 elements
- Such arrays are known as Partial Arrays
- When working with partial arrays, we work with only the actual number of elements NOT the whole size
- Moreover, if we are passing a partial array to a function, then we don't need to pass the size; instead we need to pass only the actual number of elements
- Analyze the following program and explain what it does….

Partial Arrays

```
int main()
{
    const int SIZE = 50;
    float A[SIZE];

    //Read the actual number of elements from user
    //The actual number of elements must be less than or equal to SIZE
    //So we will use a loop to make sure we read a value between 1 and size
    int number_of_elements;
    do
    {
        cout << "Enter the number of elements: ";
        cin >> number_of_elements;
    }while (number_of_elements < 1 || number_of_elements > SIZE);

    //Populate the array with random floats in the range [-5.0, 5.0)
    populateArray(A, number_of_elements);

    //Print the array
    printArray(A, number_of_elements);

    //Find the element closest to zero
    float x = findElement(A, number_of_elements);
    cout << "The element nearest to zero is " << x << endl;

    system("Pause");
    return 0;
}
```

Partial Arrays

```
void populateArray(float X[], const int n)
{
    for (int i = 0; i < n; i++)
        X[i] = float(rand())/RAND_MAX * 10 - 5;
}

void printArray(const float myArray[], const int mySize)
{
    for (int i = 0; i < mySize; i++)
        cout << myArray[i] << "     ";
    cout << endl;
}

float findElement(const float arr[], const int num)
{
    float ans = arr[0]; //This is ok because there is at least one element
    for (int i = 1; i < num; i++)
        if (abs(arr[i]) < abs(ans))
            ans = arr[i];
    return ans;
}
```

The C++ string Data Type

- **Definition:-** A C++ string is a data type that can hold **ZERO** or **MORE characters** enclosed in between double quotes
- In order to work with C++ strings, we need the include directive

  #include <string>

- Now we can declare, initialize and define string variables as follows:

```
string s;                          //string declaration
s = "This is nice";                //string initialization
string  x = "CMPT $130#";          //string definition
```

Working C++ strings

- Just like any simple data type such as int, float, double, char, bool, etc; we can
  - Print strings with **cout** command,
  - Modify the values of string variables with new string values,
  - Read string values from the keyboard using **cin** command,
  - Compare strings,
  - Pass string variables to functions
  - Return strings from functions

Working C++ strings

- The following program demonstrates declaration, initialization, definition, modification, printing and reading in strings

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string s1, s2, s3 = "Nice";
    s1 = "How";
    s2 = s1;
    cout << s1 << endl;
    cout << s2 << endl;
    cout << s3 << endl;
    cout << "Enter a string: ";
    cin >> s1;
    s3 = "Bye";
    cout << s1 << endl;
    cout << s2 << endl;
    cout << s3 << endl;

    system("Pause");
    return 0;
}
```

The output will be

How
How
Nice
Enter a string: CMPT↲
CMPT
How
Bye

Empty C++ strings

- A string that contains no characters (i.e. zero characters) is called empty string

```
string s1 = "";
string s2;
```

- Now, we see that s1 is empty string
- Also s2 is empty string. That is when we declare a string variable, C++ automatically initializes it to an empty string

Length of C++ strings

- The number of characters in a string variable is known as the length of the string
- Moreover the string data type has special function (also known as a **method**) that tell us the length of a string variable

```
string s1 = “John", s2;
int a = s1.length();
cout << "The length of " << s1 << " is " << a << endl;
cout << "The length of " << s2 << " is " << s2.length() << endl;
```

- Now the value of **a** is the length of s1 which is **4**
- **Empty string has length zero! Therefore the second line of output will print 0 for the length of s2**

C++ strings concatenation

- C++ strings can be concatenated (added) together to create a new string that has all the characters of the concatenated strings
- We can also **concatenate** a character to a string
- C++ strings concatenation is performed with a + operator
- Example:-

```
string s1 = "stop";
string s2 = "now";
char c = ' ';
string s3 = s1 + c + s2 + " please";
cout << s3 << endl;
```

- The output will be **stop now please**

Passing C++ strings to functions

- Parameter passing of C++ strings follows the rule of parameter passing of simple data types such as int, doule, bool,…
- Therefore C++ strings pass to functions by value
- Hence any modification made to a string parameter in a function does not affect the string in the calling function such as main
- Analyze the following program and determine its output

Passing C++ strings to functions

```
void modifyString(string s)
{
    s = s + " now";
    return;
}

int main()
{
    string s = "stop";
    cout << s << endl;
    modifyString(s);
    cout << s << endl;

    system("Pause");
    return 0;
}
```

Returning C++ strings from functions

- We can also return C++ strings from functions
- Analyze the following program and determine its output

```
string foo(string s)
{
    s = s + " now";
    return s;
}

int main()
{
    string s1 = "stop";
    string s2 = foo(s1);
    cout << s2 << endl;
    cout << s1 << endl;

    system("Pause");
    return 0;
}
```

C++ strings processing

- C++ strings are very similar to arrays
- In fact we can think of strings as array of characters
- Of course they are not arrays because an array is a data structure but a string is a data type
- But this correspondence helps to understand working with strings better
- Just like with arrays, each character of a string variable can be accessed by the string name and an index in square bracket
- **Index starts from 0 and goes up to (length-1)**
- Each element of a string is a **char** data type!
- With such indexing, we can now access each element of the string independently so that we can print an element, modify an element by assigning it a new character, read a character from keyboard and store it in an element, modify an element, pass an element to a function, and etc

C++ strings processing

You see that you can print the characters of a string one by one using a loop or print all of them together by printing the string variable.

Can you do the same with arrays?

```
int main()
{
    string s1 = "stop";
    cout << "The characters of " << s1 << " are: ";
    for (int i = 0; i < s1.length(); i++)
        cout << s1[i] << "  ";
    cout << endl;

    string s2 = "This is nice";
    cout << "The characters of " << s2 << " are: ";
    for (int i = 0; i < s2.length(); i++)
        cout << s2[i] << "  ";
    cout << endl;

    //Modify some elements of strings
    s1[0] = 'A';
    s1[s1.length()-1] = 'm';
    cout << "After modifying s1, we get " << s1 << endl;

    system("Pause");
    return 0;
}
```

C++ strings comparison

- C++ knows how to compare strings! Comparison is done character by character on corresponding indexes. The character comparison uses the ascii code for comparison…

```
int main()
{
    string s1 = "mark";
    string s2 = "markos";
    string s3 = "Zack";
    bool a1 = s1 == s2;
    bool a2 = s1 > s2;
    bool a3 = s1 >= s2;
    bool a4 = s1 < s2;
    bool a5 = s1 <= s2;
    bool a6 = s1 != s2;
    cout << a1 << " " << a2 << " " << a3 << " " << a4 << " " << a5 << " " << a6 << endl;
    bool a7 = s3 > s1;
    cout << a7 << endl;

    system("Pause");
    return 0;
}
```

Reversing C++ strings

- Empty strings and string concatenation are useful concepts to process strings. Analyze the following program

```
string getReversedString(string s)
{
    string x = "";
    for (int i = s.length()-1; i >= 0 ; i--)
        x = x + s[i]; //You could write this as x += s[i]
    return x;
}
int main()
{
    string s1 = "stop";
    string s2 = getReversedString(s1);
    cout << s2 << endl;

    system("Pause");
    return 0;
}
```

Reversing C++ strings

- Of course instead of looping with decreasing indexes; we could loop with increasing indexes and take advantage of commutative property of string concatenation as follows

```
string getReversedString(string s)
{
    string x = "";
    for (int i = 0; i < s.length(); i++)
        x = s[i] + x; //You can NOT write this as x += s[i]
    return x;
}
```

C++ string methods

- C++ string has many more methods that give some information about a string variable
- Few of these methods are:
  - **find**:- returns the first index of a character argument in a string. Returns **the maximum value of an unsigned int data type** if the character is not found in the string*
  - **rfind**:- returns the last index of a character argument in a string. Returns **the maximum value of an unsigned int data type** if the character is not found in the string
  - **empty**:- returns true if a string is empty; otherwise returns false
  - **append**:- appends a string or a character argument to a string
  - **erase**:- deletes all the characters of a string
- *Note that the maximum value of an **unsigned int data type** is compiler dependent. In Visual C++ 2010, it is $\mathbf{2^{32}-1}$
- More methods at **www.cplusplus.com/reference/string/string/**