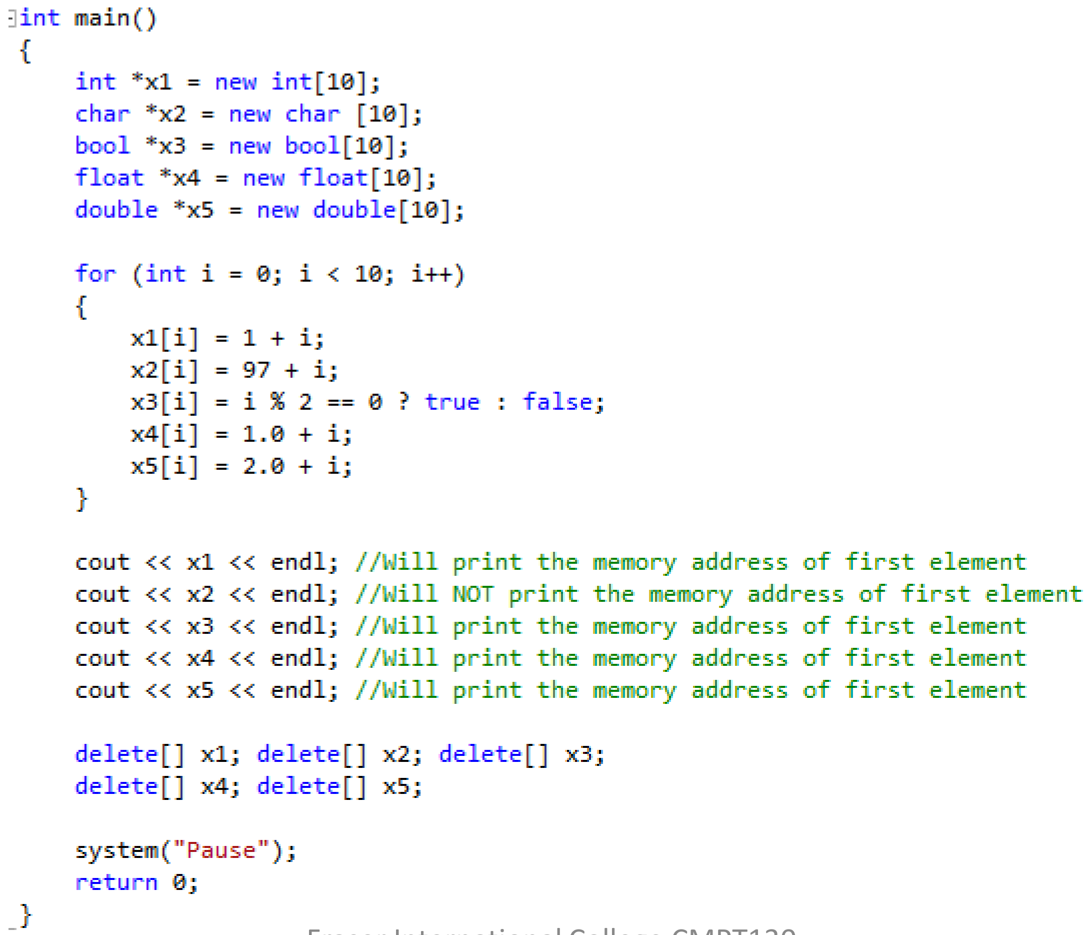
C-strings

In this Week

- char arrays and overloaded << operator
- What is C-string?
- Working with C-strings
- C-string library functions (**cstdlib** library)
  - strlen, strcmp, strcpy,…
- char arrays and overloaded >> operator
- Dynamic Arrays of C-strings

char arrays and overloaded << operator

- Consider the following program and determine its output



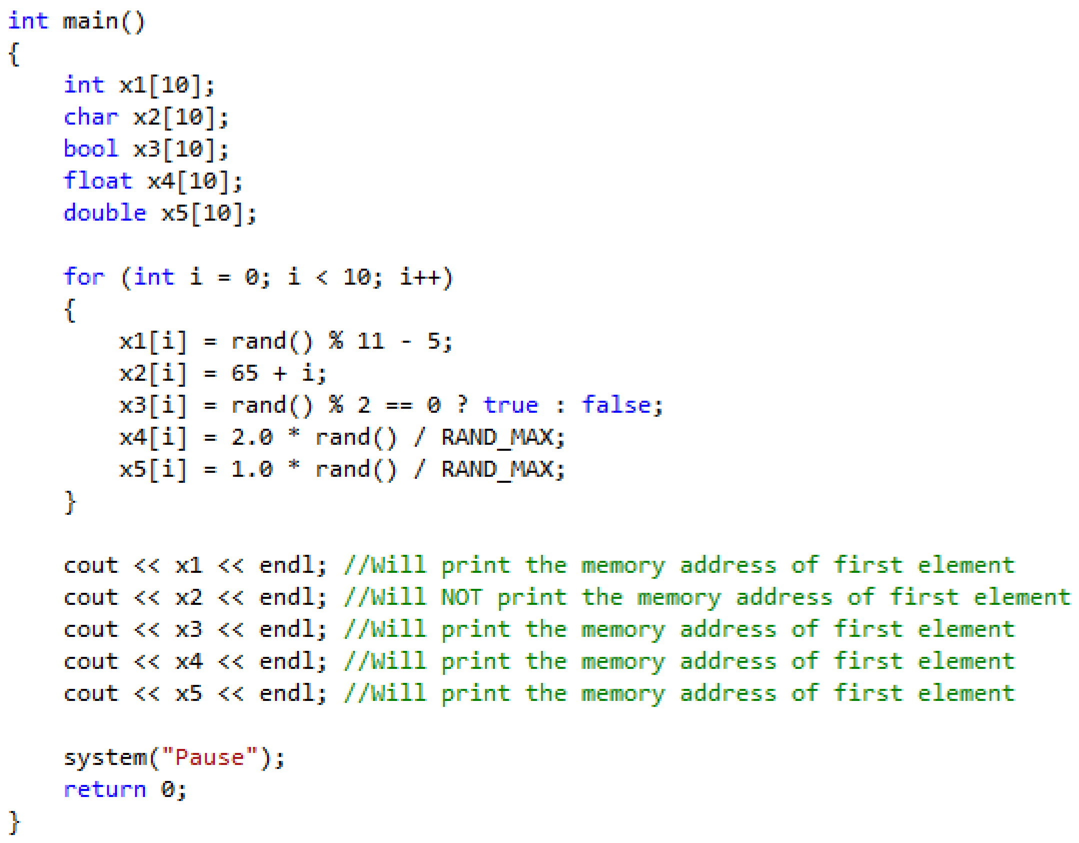
```
int main()
{
    int *x1 = new int[10];
    char *x2 = new char [10];
    bool *x3 = new bool[10];
    float *x4 = new float[10];
    double *x5 = new double[10];

    for (int i = 0; i < 10; i++)
    {
        x1[i] = 1 + i;
        x2[i] = 97 + i;
        x3[i] = i % 2 == 0 ? true : false;
        x4[i] = 1.0 + i;
        x5[i] = 2.0 + i;
    }

    cout << x1 << endl; //Will print the memory address of first element
    cout << x2 << endl; //Will NOT print the memory address of first element
    cout << x3 << endl; //Will print the memory address of first element
    cout << x4 << endl; //Will print the memory address of first element
    cout << x5 << endl; //Will print the memory address of first element

    delete[] x1; delete[] x2; delete[] x3;
    delete[] x4; delete[] x5;

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

char arrays and overloaded << operator

- As expected printing the values of the pointers **x1, x3, x4 and x5** print the memory addresses of the first elements of these dynamic arrays
- But printing the value of **x2** does **NOT** print the memory address of the first element of **x2**
- Instead it prints all the elements of **x2** and keeps on printing garbage characters which may result to a run time error
- We say the dynamic array of characters (**which is a pointer to char data type**) is treated differently by **cout <<** compared to any other data types
- **Conclusion:- The output stream operator << is overloaded for pointers of char data type**

char arrays and overloaded << operator

- Next, consider the following program and determine its output. This time using static arrays



```cpp
int main()
{
    int x1[10];
    char x2[10];
    bool x3[10];
    float x4[10];
    double x5[10];

    for (int i = 0; i < 10; i++)
    {
        x1[i] = rand() % 11 - 5;
        x2[i] = 65 + i;
        x3[i] = rand() % 2 == 0 ? true : false;
        x4[i] = 2.0 * rand() / RAND_MAX;
        x5[i] = 1.0 * rand() / RAND_MAX;
    }

    cout << x1 << endl; //Will print the memory address of first element
    cout << x2 << endl; //Will NOT print the memory address of first element
    cout << x3 << endl; //Will print the memory address of first element
    cout << x4 << endl; //Will print the memory address of first element
    cout << x5 << endl; //Will print the memory address of first element

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

char arrays and overloaded << operator

- Once again as expected printing the values of the static array variables **x1, x3, x4 and x5** print the memory addresses of the first elements of these static arrays
- But printing the value of **x2** does **NOT** print the memory address of the first element of **x2**
- Instead it prints all the elements of **x2** and keeps on printing garbage characters which may result to a run time error
- We say the static array of characters is treated differently by **cout <<** compared to any other data types
- **Conclusion:- Recalling the fact that static arrays are also pointers variables , we conclude once again that the output stream operator << is overloaded for pointers of char data type**

char arrays and overloaded << operator

- So we may ask the question: when does the **cout** statement stop printing?
- **Answer**:- Given a dynamic or static array of characters, the **cout** statement will keep on printing the characters in the array (**and may be past the elements of the array**) until it finds a special character known as the **null character** (written as **'\0'**) which instructs the **cout** statement to stop printing
- This means unless our character array has **'\0'** as its last element, our program will access memory spaces that do not belong to our program which may crash our program
- **Remark:- The '\0' character has ascii code of 0.**

C-strings

- **A C-string is a C++ static or dynamic array of characters whose last element is the null character ('\0')**
- An example of a C-string using a static array is

  ```
  char A[] = {'a', '6', '@', 'g', '\0'};
  ```
- The same example of C-string using a dynamic array is

  ```
  char *A = new char[5];
  A[0] = 'a'; A[1] = '6'; A[2] = '@'; A[3] = 'g'; A[4] = '\0';
  ```
- **When using a dynamic array, the heap space must first be reserved with the new operator**

C-strings

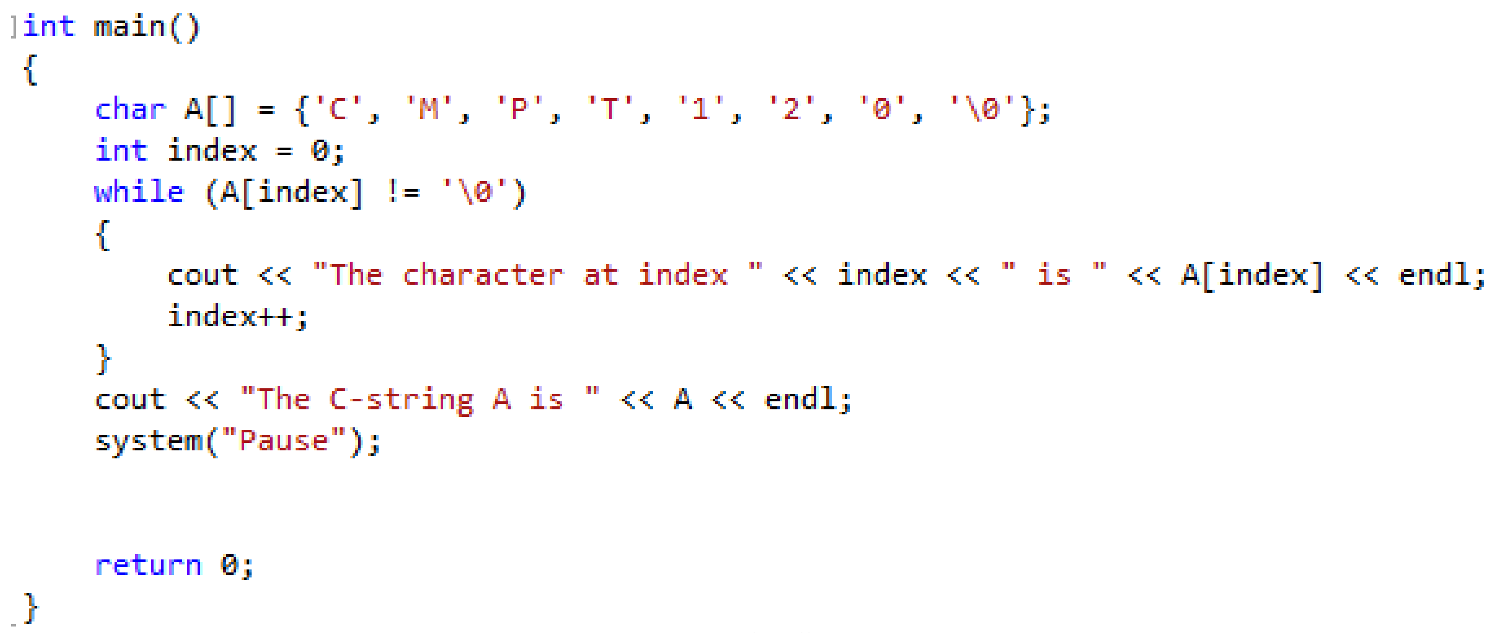
- The last character i.e. **the null character** indicates **the end of the C-string**
- The **cout** statement can print a C-string and is designed to recognize the end of the C-string thanks to the null character at the end
- Therefore the following code

```
char A[] = {'a', '6', '@', 'g', '\0'};
cout << A << endl;
```

  Will print **a6@g**
- The **'\0'** is not printed!
- We say the C-string **A** contains
  - The **printable characters 'a', '6', '@',** and **'g'**; and
  - The **non-printable character '\0'**

C-strings

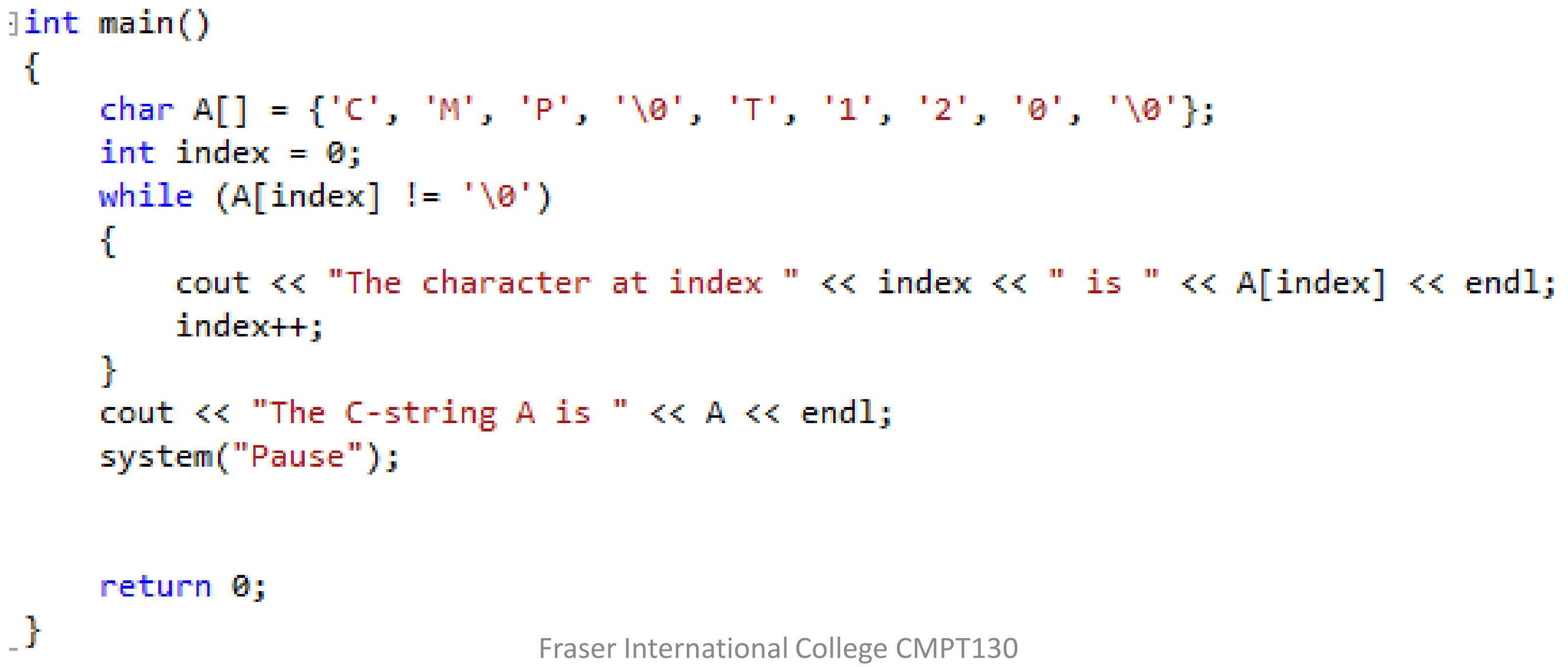
- When we process C-strings, we always use the null character as indicator of the end of the C-string
- In the following example, we print the characters of a C-string one by one and use the null character as a stopping criteria



```
int main()
{
    char A[] = {'C', 'M', 'P', 'T', '1', '2', '0', '\0'};
    int index = 0;
    while (A[index] != '\0')
    {
        cout << "The character at index " << index << " is " << A[index] << endl;
        index++;
    }
    cout << "The C-string A is " << A << endl;
    system("Pause");


    return 0;
}
```

C-strings

- When a null character is found inside a C-string then it will act as the end marker of the C-string and the characters after it will not be printed by cout <<
- What will be the output of the following program?



```
int main()
{
    char A[] = {'C', 'M', 'P', '\0', 'T', '1', '2', '0', '\0'};
    int index = 0;
    while (A[index] != '\0')
    {
        cout << "The character at index " << index << " is " << A[index] << endl;
        index++;
    }
    cout << "The C-string A is " << A << endl;
    system("Pause");


    return 0;
}
```

C-strings: Initialization

- C-strings can be initialized as character arrays as we showed before:
- Example: **char C[ ] = {'C','M','P','T',' ','1','2','0','\0'};**
- However because C-strings are treated a little differently than arrays, they can also be initialized as follows:

```
char C[ ] = "CMPT 120";
```

- This is the same as writing

```
char C[ ] = {'C','M','P','T',' ','1','2','0','\0'};
```

Length of C-strings

- The length of a C-string is the number of the printable characters in it
- The **null character** is **not counted for length**!
- Thus in order to create a C-string of length n, we need to create an array of size n+1
- For example, to create a C-string containing "CMPT" then we need to create an array of 5 characters and then initialize the elements at index 0,1,2, and 3 with the characters 'C', 'M', 'P' and 'T' respectively and initialize the element at index 4 with a null character

Length of C-strings

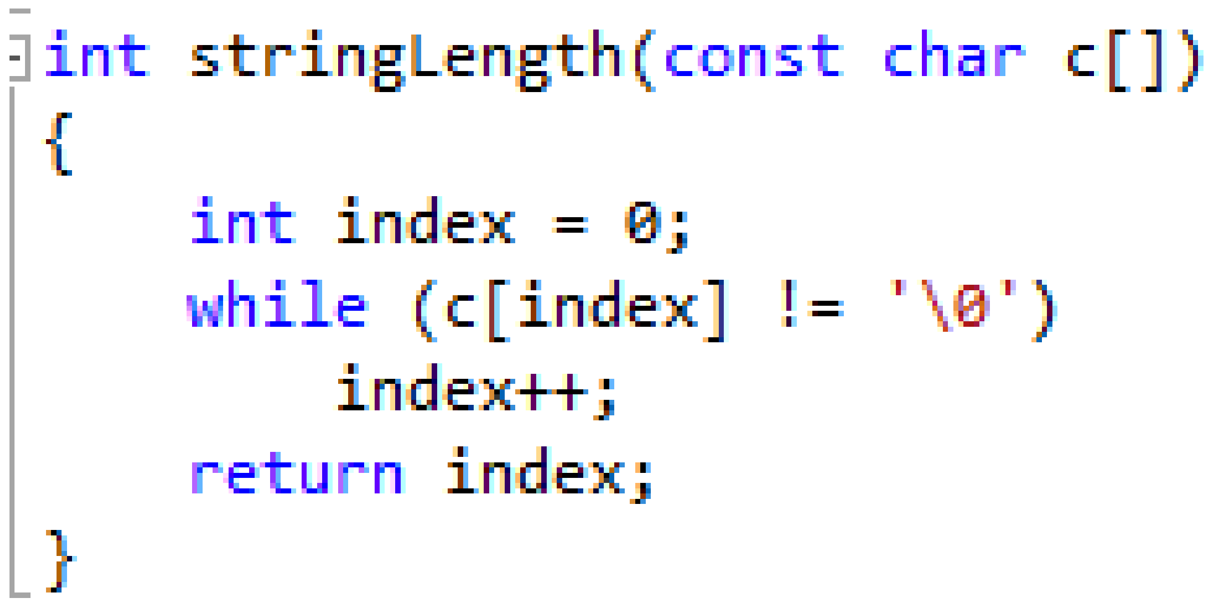
- Consider the C-string

    char A[] = {'a', '6', '@', 'g', '\0'};

- The length of this C-string is **4** (**NOT 5**)
- Of course as an array of characters **A** contains 5 characters, i.e., including the null character
- But when we look at **A** as a C-string, we say it has length 4

Length of C-strings

- Write a C++ function named **stringLength** that takes a C-string argument and returns its length
- Here all we need to remember is not to count the null character in the length computation



```cpp
int stringLength(const char c[])
{
    int index = 0;
    while (c[index] != '\0')
        index++;
    return index;
}
```

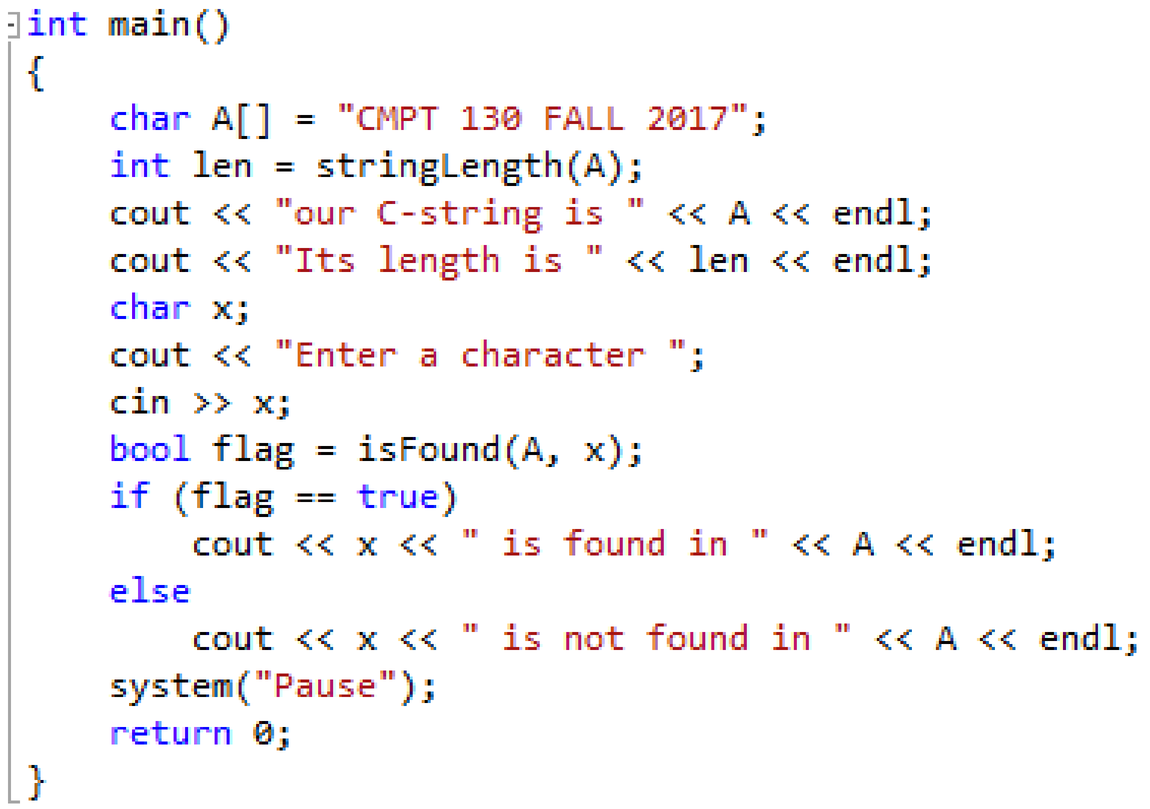
Empty C-string

- A C-string is said to be empty if it has zero printable characters, i.e., it's length is **0**
- Therefore the array must have size 1; that is, one memory space for the null character
- For example

  char A[] = {'\0'}; OR **char A[] = "";**
- OR **char *A = new char('\0');**
- Now A is an empty C-string

Processing C-strings

- Consider the following program and make it work by implementing the required functions



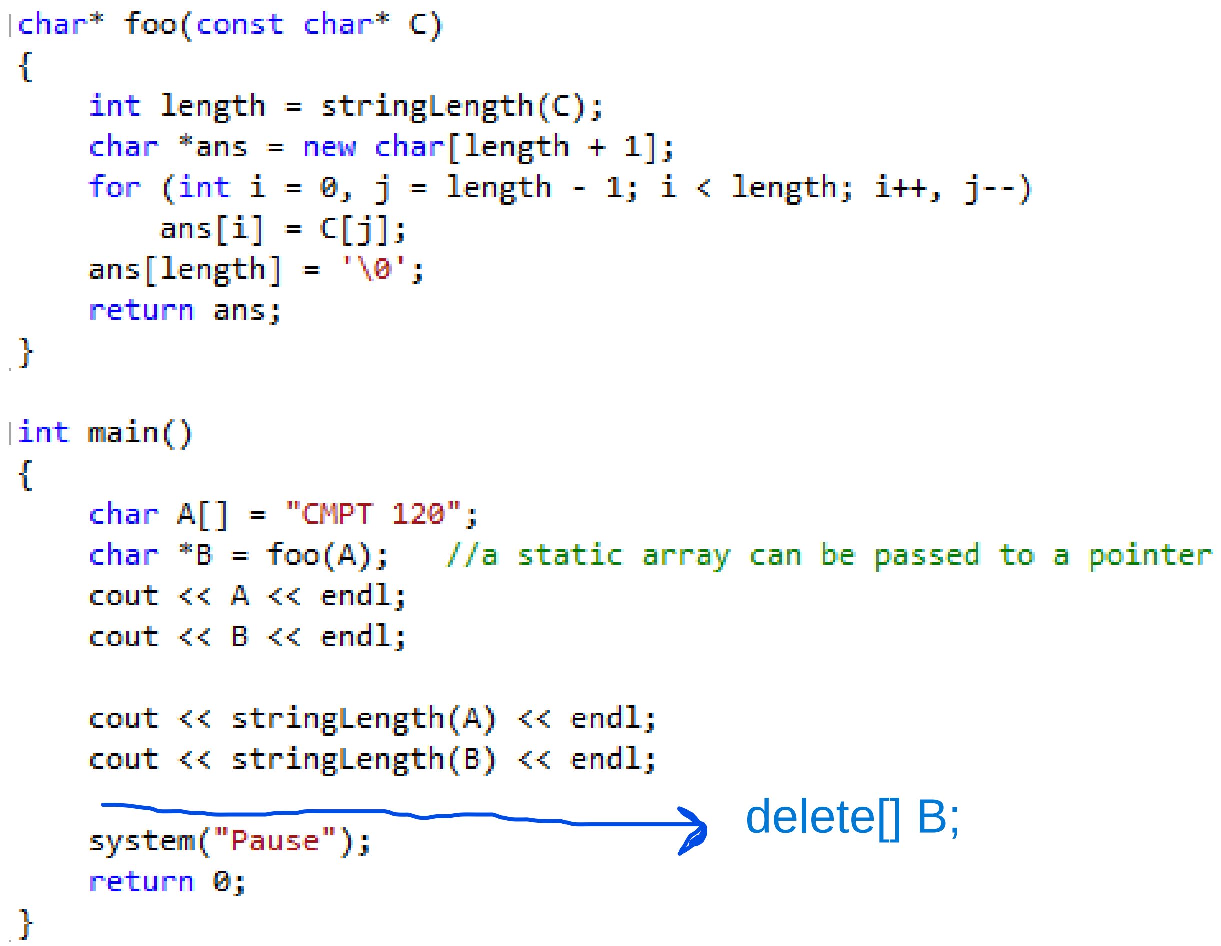
```
int main()
{
    char A[] = "CMPT 130 FALL 2017";
    int len = stringLength(A);
    cout << "our C-string is " << A << endl;
    cout << "Its length is " << len << endl;
    char x;
    cout << "Enter a character ";
    cin >> x;
    bool flag = isFound(A, x);
    if (flag == true)
        cout << x << " is found in " << A << endl;
    else
        cout << x << " is not found in " << A << endl;
    system("Pause");
    return 0;
}
```

Processing C-strings

- When working with C-strings, it is advisable we first compute the length of the C-string, say by calling the **stringLength** function we wrote earlier, and then use the length in loop structures
- In the subsequent examples, we deploy the **stringLength** function to process C-strings
- For more examples…. See lab work questions

Processing C-strings

- Analyze the following code and determine its output



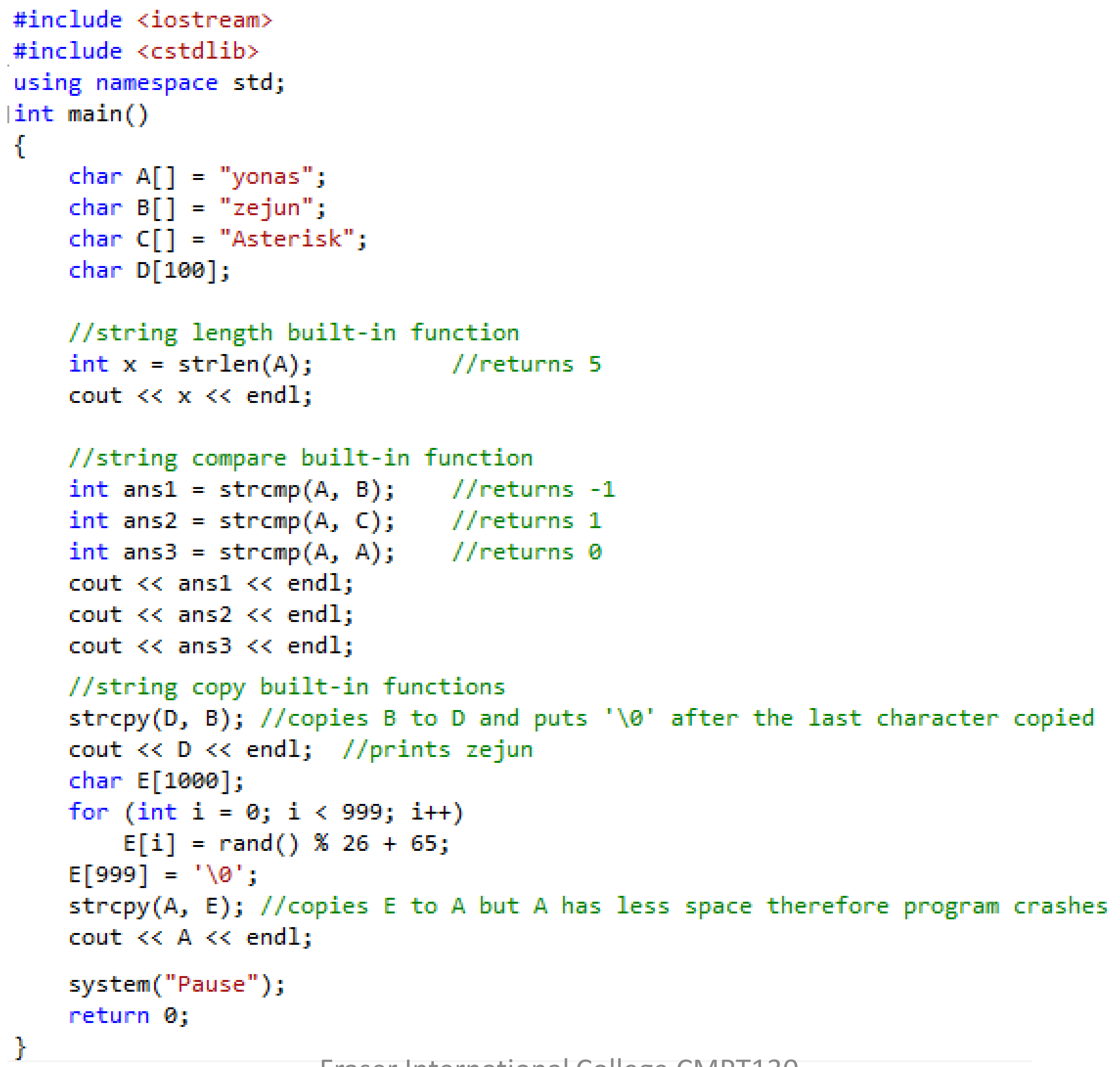
```
char* foo(const char* C)
{
    int length = stringLength(C);
    char *ans = new char[length + 1];
    for (int i = 0, j = length - 1; i < length; i++, j--)
        ans[i] = C[j];
    ans[length] = '\0';
    return ans;
}

int main()
{
    char A[] = "CMPT 120";
    char *B = foo(A);    //a static array can be passed to a pointer
    cout << A << endl;
    cout << B << endl;

    cout << stringLength(A) << endl;
    cout << stringLength(B) << endl;

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

delete[] B;

C-string Library Functions

- The **cstdlib** library comes with built-in functions for C-strings
- Few of these functions are:
  - **strlen(char* s)**:- returns the number of printable characters in s
  - **strcmp(char* s1, char* s2)**:- compares two C-strings s1 and s2. Returns 1 if s1 is greater than s2, returns -1 if s1 is less than s2, and returns 0 if they are equal
  - **strcpy(char* s1, char* s2)**:- copies all the characters of s2 including the null character to s1. Here we should first make sure that s1 has enough space to copy all characters of s2 including the '\0'
- Analyze the following program and determine its output

C-string Library Functions



```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
    char A[] = "yonas";
    char B[] = "zejun";
    char C[] = "Asterisk";
    char D[100];

    //string length built-in function
    int x = strlen(A);          //returns 5
    cout << x << endl;

    //string compare built-in function
    int ans1 = strcmp(A, B);    //returns -1
    int ans2 = strcmp(A, C);    //returns 1
    int ans3 = strcmp(A, A);    //returns 0
    cout << ans1 << endl;
    cout << ans2 << endl;
    cout << ans3 << endl;
    //string copy built-in functions
    strcpy(D, B); //copies B to D and puts '\0' after the last character copied
    cout << D << endl;  //prints zejun
    char E[1000];
    for (int i = 0; i < 999; i++)
        E[i] = rand() % 26 + 65;
    E[999] = '\0';
    strcpy(A, E); //copies E to A but A has less space therefore program crashes
    cout << A << endl;

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

char arrays and overloaded >> operator

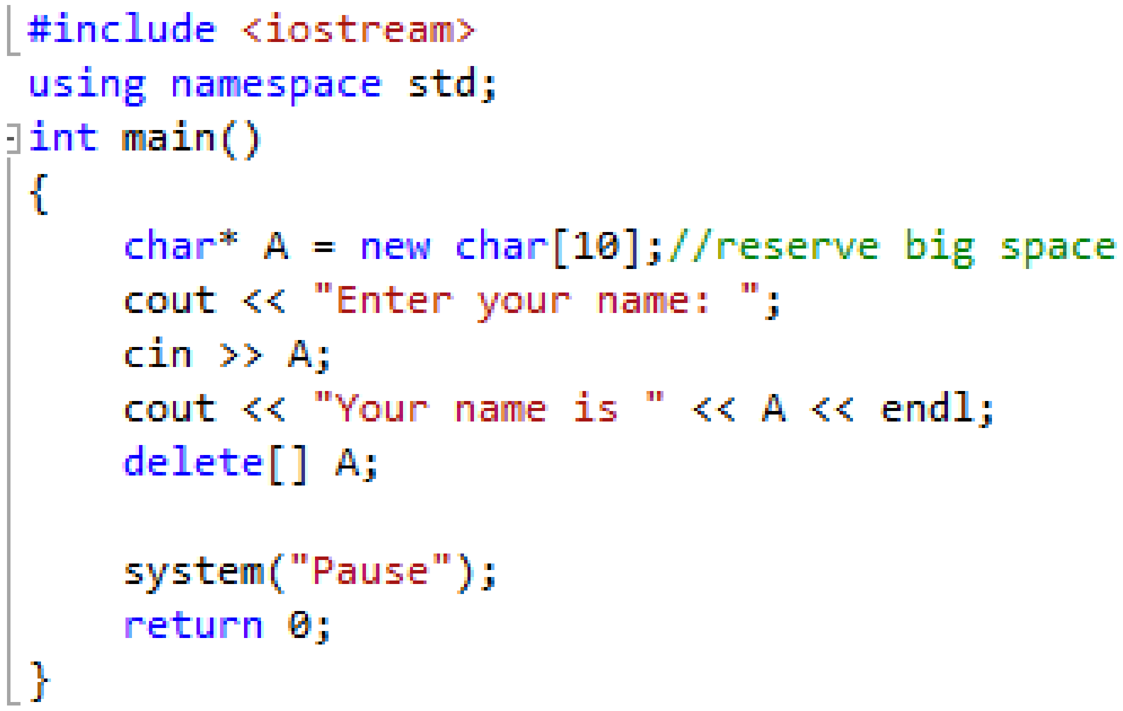
- Given a character array such as

  char *A = new char[100]; OR **char A[100];**
- We can read characters from the keyboard and put them in char A using

  cin >> A;
- Here **cin >>** will insert each character we type into the array and it will stop when either a **spacebar** key or **enter** key is pressed on the keyboard. We say the >> operator is overloaded.
- Moreover the **cin >>** will also **insert '\0' after the last character** inserted into the array making the character array a correct C-string
- Of course we have to make sure that the character array has enough space to store all the characters entered from the keyboard and the '\0' appended at the end by **cin >>**.
- Otherwise the **cin** will access memory spaces that do not belong to our program; thus crashing our program

char arrays and overloaded >> operator

- Run the program below several times and give it name inputs of different lengths including less than 10 characters and more than 10 characters and see how the program behaves



```
#include <iostream>
using namespace std;
int main()
{
	char* A = new char[10];//reserve big space
	cout << "Enter your name: ";
	cin >> A;
	cout << "Your name is " << A << endl;
	delete[] A;

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

C-string versus C++ string Data Type

- It is important to note that a **C-string** is **NOT** a data type on its own
- A C-string is simply a static or dynamic array of characters whose last element is the null character
- The C++ **string** data type however is a data type on its own
- Thus the **C++ string** and **C-string** are different things and should not be mixed up

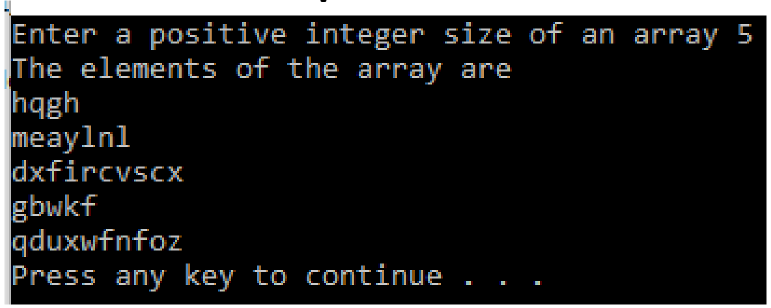
Dynamic Arrays of C-strings

- We can also create an array of C-Strings
- In this case, every element of the array will be a C-string; which is a pointer to char data type
- Therefore an array of C-strings will be a pointer to pointers to char data type
- In the following example, we will create a user defined size array of C-strings and then assign each element of the array a random C-string
- **Practice:-** Define the function **random_C_String** that takes no argument and generates and returns a C-string of random length in the range [3, 12] and filled with random lower case English alphabets

Dynamic Arrays of C-strings

```cpp
int main()
{
    int size;
    do
    {
        cout << "Enter a positive integer size of an array ";
        cin >> size;
    }while (size <= 0);

    //Declare an array of C-strings and allocate memory for the pointer elements
    char ** arr = new char*[size];

    //Assign each element a random C-string
    for (int i = 0; i < size; i++)
        arr[i] = random_C_String();

    //Print the elements of the array
    cout << "The elements of the array are" << endl;
    for (int i = 0; i < size; i++)
        cout << arr[i] << endl;

    //De-allocate (i.e. delete) the memory spaces allocated
    for (int i = 0; i < size; i++)
        delete[] arr[i];
    delete[] arr;

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

Sample Run



```
Enter a positive integer size of an array 5
The elements of the array are
hqgh
meaylnl
dxfircvscx
gbwkf
qduxwfnfoz
Press any key to continue . . .
```