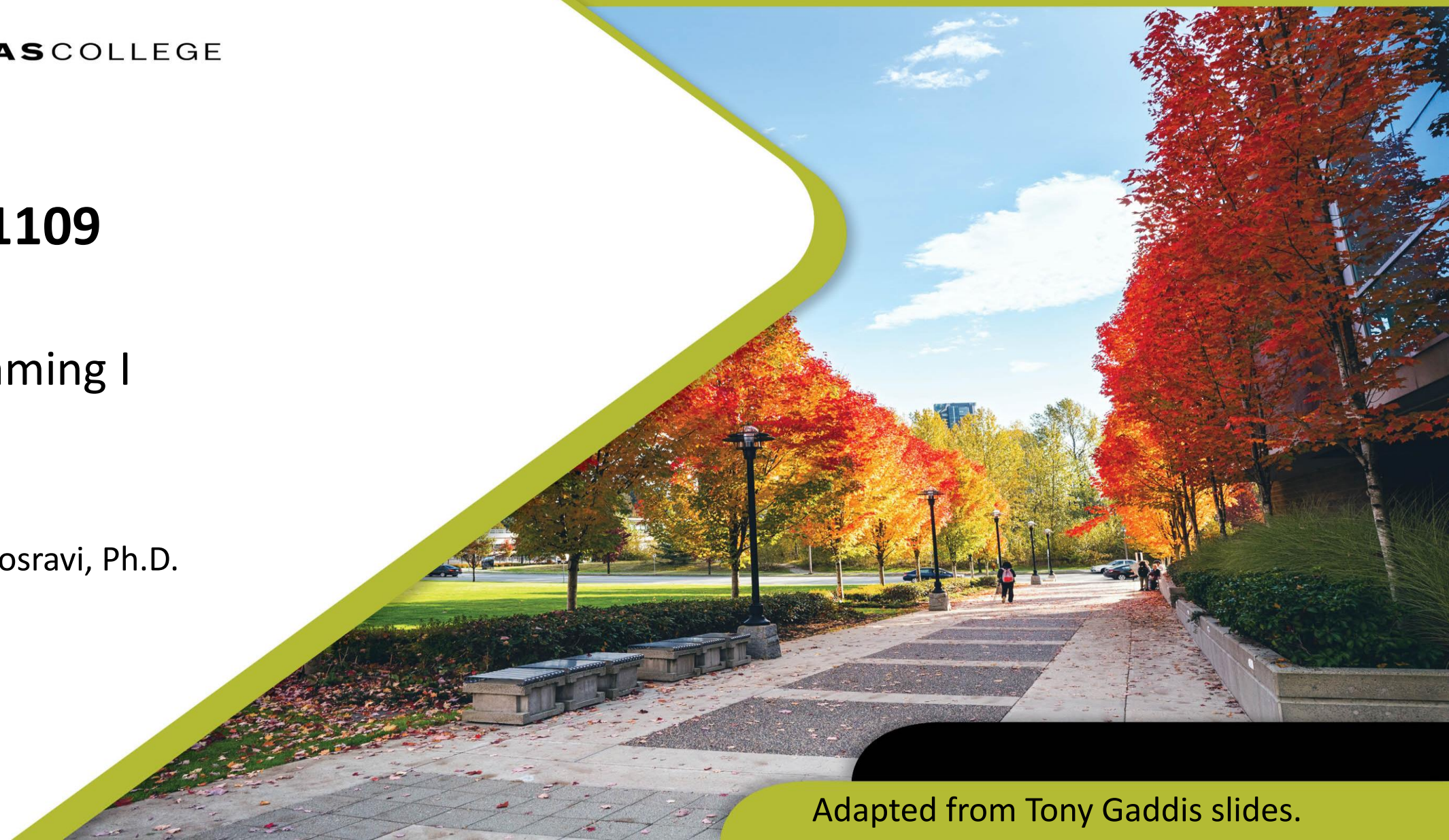
DOUGLAS COLLEGE

CMPT 1109

Programming I

Shahriar Khosravi, Ph.D.

Lecture 9



Adapted from Tony Gaddis slides.

Plan for Today

- Character Testing
- Character Case Conversion
- C-Strings
- Library Functions for Working with C-Strings
- String/Numeric Conversion Functions
- More about the C++ string Class

DOUGLAS COLLEGE

Character Testing

Character Testing

- The C++ library provides several functions that allow us to test the value of a character.
- These functions test a single **`char`** argument and return either **`true`** or **`false`**.
- To use these functions, we must **`#include`** the **`<cctype>`** header file.
- For example, this program uses the **`isupper()`** function to determine whether the character passed as an argument is an uppercase letter.

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

int main()
{
    char letter = 'a';
    if (isupper(letter))
        cout << "Letter is uppercase.\n";
    else
        cout << "Letter is lowercase.\n";

    return 0;
}
```

Character Testing Functions

Character Function	Description
isalpha	Returns **true** (a nonzero number) if the argument is a letter of the alphabet. Returns **0** if the argument is not a letter.
isalnum	Returns **true** (a nonzero number) if the argument is a letter of the alphabet or a digit. Otherwise, it returns **0**.
isdigit	Returns **true** (a nonzero number) if the argument is a digit from **0** through **9**. Otherwise, it returns **0**.
islower	Returns **true** (a nonzero number) if the argument is a lowercase letter. Otherwise, it returns **0**.
isprint	Returns **true** (a nonzero number) if the argument is a printable character (including a space). Returns **0** otherwise.
ispunct	Returns **true** (a nonzero number) if the argument is a printable character other than a digit, letter, or space. Returns **0** otherwise.
isupper	Returns **true** (a nonzero number) if the argument is an uppercase letter. Otherwise, it returns **0**.
isspace	Returns **true** (a nonzero number) if the argument is a whitespace character. Whitespace characters are any of the following: space vertical tab **'\v'** newline **'\n'** tab **'\t'** Otherwise, it returns **0**.

DOUGLAS COLLEGE

Case Conversion

Case Conversion

- The C++ library provides two functions, **`toupper()`** and **`tolower()`**, for converting the case of a character.

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

int main()
{
    const double PI = 3.14159;  // Constant for Pi
    double radius;              // The circle's radius
    char goAgain;               // To hold Y or N

    cout << "This program calculates the area of a circle.\n";
    cout << fixed << setprecision(2);

    do
    {
        // Get the radius and display the area.
        cout << "Enter the circle's radius: ";
        cin >> radius;
        cout << "The area is " << (PI * radius * radius);
        cout << endl;

        // Does the user want to do this again?
        cout << "Calculate another? (Y or N) ";
        cin >> goAgain;

        // Validate the input.
        while (toupper(goAgain) != 'Y' && toupper(goAgain) != 'N')
        {
            cout << "Please enter Y or N: ";
            cin >> goAgain;
        }

    } while (toupper(goAgain) == 'Y');
    return 0;
}
```

DOUGLAS COLLEGE

C-Strings

C-Strings

- **String** is a generic term that describes any consecutive sequence of characters.
- In the C++ language, there are two primary ways that strings are stored in memory: as **`string`** objects or as **C-strings**.
- We have already seen the **`string`** class, and by now, we have written several programs that use **`string`** objects.
- A **C-string** is a string whose characters are stored in consecutive memory locations and are followed by a **null character**, or **null terminator**.
  - A a **`null character`** or **`null terminator`** is a byte holding the ASCII code **`0`**.
- Strings that are stored this way are called **C-strings** **because this is the way strings are handled in the C programming language**.
- Recall that a **string literal** (or string constant) is the literal representation of a string in a program. String literals are enclosed in double quotation marks.

C-Strings

- The diagram below illustrates how the string literal **"Hi there!"** is stored in memory, as a C-string.

```
int main()
{
    cout << "Hi there!";
    return 0;
}
```

H	i		t	h	e	r	e	!	\0

- Remember that **\0** ("slash zero") is the escape sequence representing the **null terminator**. It stands for the ASCII code **0**.
- The purpose of the null terminator is to mark the end of the C-string.
- **Without the null terminator, there would be no way for a program to know the length of a C-string**.

C-Strings

- It is important to understand that a string literal has its own storage location, just like a variable or an array.
- **When a string literal appears in a statement, C++ uses its memory address!**

```
cout << "Hi there!";
```

- In the above statement, the memory address of the string literal **"Hi there!"** is passed to the **cout** object.
- The **cout** object then displays the consecutive characters found at this address.
- It stops displaying the characters when a null terminator is encountered. This is why in C++, every C-string ends with the null terminator.

Why Care about C-Strings?

- The C programming language does **not** provide a string class like that which C++ provides.
- In the C language, all strings are treated as C-strings.
- When a C programmer wants to store a string in memory, they must create a **char** array that is large enough to hold the string**, plus one extra element for the null character**.
- We need to know about C-strings for the following reasons:
  - The string class has not always existed in the C++ language. Several years ago, C++ stored strings as C-strings. As professional programmers, we might encounter older legacy C++ code that uses C-strings.
  - Some of the C++ library functions work only with C-strings.
  - In the industry, it is not unusual for C++ programmers to work with specialized libraries that are written in C. Any strings with which C libraries work will be C-strings.

C-String Initialization

- If we want to store a C-string in memory, we have to define a **char** array that is large enough to hold the string, plus one extra element for the null character.
- This code defines a **char** array that has **21** elements, so it is big enough to hold a C-string that is no more that **20** characters long.

```
const int SIZE = 21;
char name[SIZE] = "Jasmine";
```

- The **name** array will be created with **21** elements. The first eight elements will be initialized with the characters **'J'**, **'a'**, **'s'**, **'m'**, **'i'**, **'n'**, **'e'**, and **'\0'**.
- The null character is automatically added as the last character.
- We can implicitly size a char array by initializing it with a string literal:

```
char name[] = "Doja Cat";
```

C-String Input

- C-string input can be performed by the **cin** object.

```
const int SIZE = 21;
char name[SIZE];
cin >> name;
```

- **Recall that the name of an array (with no brackets and no subscript) is an alias for the address of the zeroth element of the array**.
- In the above statement, **name** indicates the address in memory where the string is to be stored.
- The **cin** object has no way of knowing that name has 21 elements.
- If the user enters a string of **30** characters, cin will write past the end of the array!

C-String Input

- C-string input can be performed by the **cin** object.

```
const int SIZE = 21;
char name[SIZE];
cin >> name;
```

- **Recall that the name of an array (with no brackets and no subscript) is an alias for the address of the zeroth element of the array**.
- In the above statement, **name** indicates the address in memory where the string is to be stored.
- The **cin** object has no way of knowing that name has 21 elements.
- If the user enters a string of **30** characters, cin will write past the end of the array!

C-String Input

- This issue can be prevented by using **`cin`**'s **`getline()`** member function.

```
const int SIZE = 80;
char line[SIZE];
cin.getline(line, SIZE);
```

- The first argument tells **`getline()`** where to store the string input.
- The above **`getline()`** statement indicates the starting address of the **`line`** array as the storage location for the string.
- The second argument indicates the maximum length of the string, **including the null terminator**.
- In this example, the **`SIZE`** constant is equal to **`80`**, so **`cin`** will read **`79`** characters, or until the user presses the **Enter** key, whichever comes first.
- The **`cin`** object will automatically append the null terminator to the end of the string.

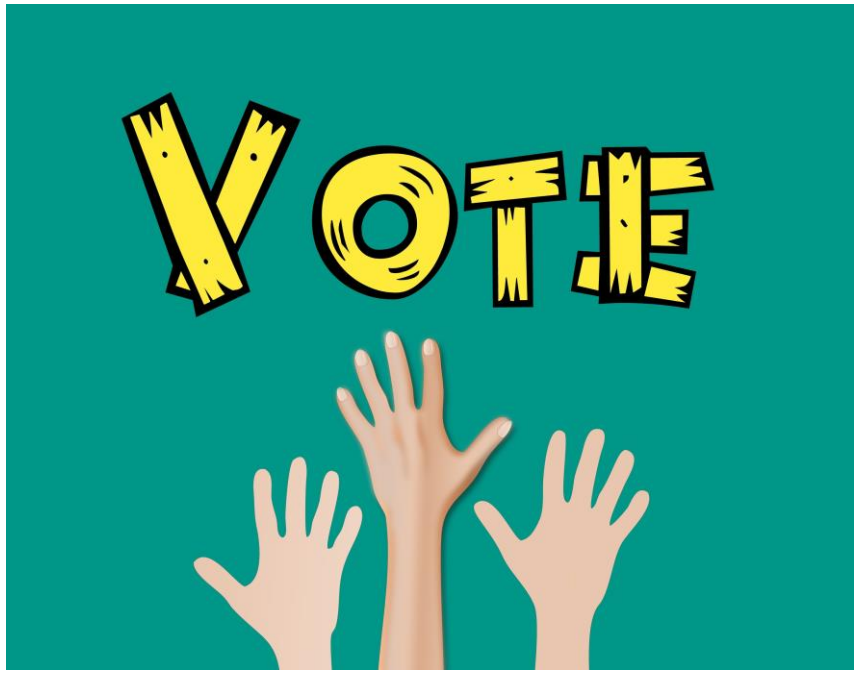
Poll 1 (Extra Credit)

The output of the following program on the console screen is "NOT the same!".

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

int main()
{
    char str1[] = "Bieber";
    char str2[] = "Bieber";

    if (str1 != str2)
        cout << "NOT the same!" << endl;

    return 0;
}
```

a) Yay!

b) Nay!



Please use the "Poll" window to participate for extra credit! One answer only please!

Example

```
// This program displays a string stored in a char array.
#include <iostream>
using namespace std;

int main()
{
    const int SIZE = 80;  // Array size
    char line[SIZE];      // To hold a line of input
    int count = 0;        // Loop counter variable

    // Get a line of input.
    cout << "Enter a sentence of no more than "
         << (SIZE - 1) << " characters:\n";
    cin.getline(line, SIZE);

    // Display the input one character at a time.
    cout << "The sentence you entered is:\n";
    while (line[count] != '\0')
    {
        cout << line[count];
        count++;
    }
    return 0;
}
```

DOUGLAS COLLEGE

Library Functions for Working with C-Strings

The `strlen()` Function

- Since C-strings are stored in arrays, working with them is quite different from working with string objects.
- The C++ library has numerous functions for handling C-strings. These functions perform various tests and manipulations.
- These functions all require the **`<cstring>`** header file to be included.
- For instance, the following code segment uses the **`strlen()`** function to determine the length of the string stored in the **`name`** array.

```
char name[] = "Nikola Tesla";
int length;
length = strlen(name);
```

- The **`strlen()`** function accepts a pointer to a C-string as its argument.
- It returns the length of the string, which is the number of characters up to, **but not including, the null terminator**.

The `strlen()` Function

- When using a C-string-handling function, we must pass one or more C-strings as arguments.
- This means passing the address of the C-string, which may be accomplished by using any of the following as arguments:
  - The name of the array holding the C-string
  - A pointer variable that holds the address of the C-string
  - A literal string
- **Anytime a literal string is used as an argument to a function, the address of the literal string is passed,** not the literal itself!

```
length = strlen("Nikola Tesla");
```

The strcat() Function

- The **strcat()** function accepts two pointers to C-strings as its arguments.
- The function concatenates, or appends one string to another.

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
    const int SIZE = 20;
    char city[SIZE] = "New Westminster, ";
    char province[3] = "BC";
    strcat(city, province);
    // city now has "New Westminster, BC"

    return 0;
}
```

It is the programmer's responsibility to make sure the array holding str1 is large enough to hold str1 plus str2 plus a null terminator. The strcat() function performs no bounds checking, so str1 can potentially be overflown if we are not careful!!!

The strcat() Function

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
    const int SIZE = 19;
    char city[SIZE] = "New Westminster, ";
    char province[3] = "BC";
    if (sizeof(city) >= (strlen(city) + strlen(province) + 1))
        strcat(city, province);
    else
        cout << "Huston, we have a size problem!" << endl;
    return 0;
}
```

Poll 2 (Extra Credit)

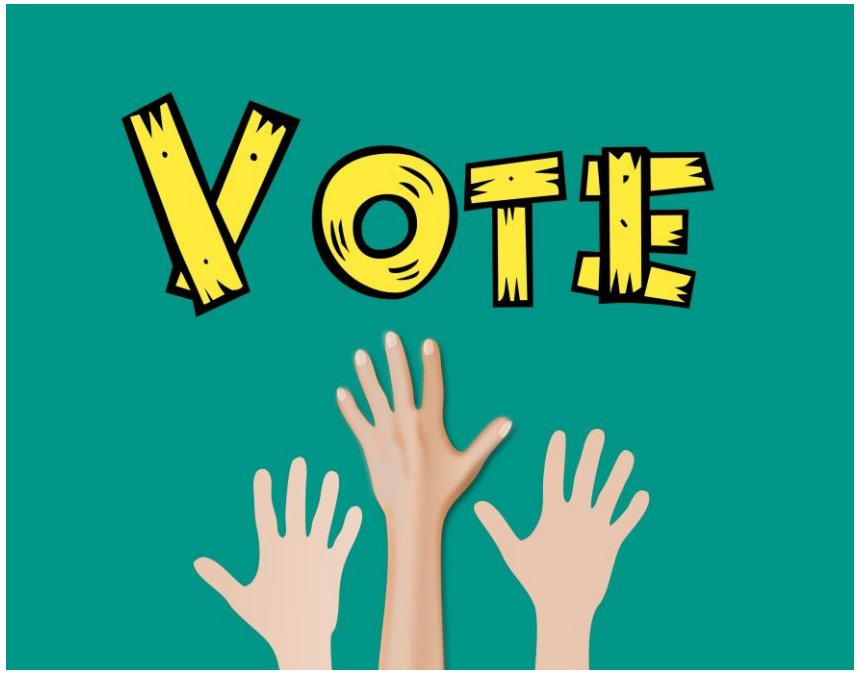
The following code is syntactically legal in C++.

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

int main()
{
    int arr1[] = {1, 2, 3};
    int arr2[] = {4, 5, 6};
    arr2 = arr1;

    return 0;
}
```

a) Yay!

b) Nay!

Please use the "Poll" window to participate for extra credit! One answer only please!



The `strcpy()` Function

- Recall that one array **cannot** be assigned to another with the = operator.
- Each individual element must be assigned, usually inside a loop.
- The **`strcpy()`** function can be used to copy one string to another without a loop.
- The **`strcpy()`** function's two arguments are C-string addresses.
- The contents of the second argument are copied to the memory location specified by the first argument, **including the null terminator**.
- If anything is already stored in the location referenced by the first argument, it is overwritten.

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
    const int SIZE = 13;
    char name[SIZE];
    strcpy(name, "Albert Einstein");

    return 0;
}
```

The strcpy() function performs no bounds checking, so str1 can potentially be overflown if we are not careful!!!

The `strncat()` and `strncpy()` Functions

- Since the **`strcat()`** and **`strcpy()`** functions can potentially overwrite the bounds of an array, they make it possible to write unsafe code.
- As an alternative, we should use **`strncat()`** and **`strncpy()`** whenever possible.
- The **`strncat()`** functions works like **`strcat()`**, except it takes a third argument **`n`** specifying the maximum number of characters from the second string to append to the first.
- The **`strncat()`** function will append no more than **`n`** characters from **`string2`** to **`string1`**.

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
    int maxChars;
    const int SIZE_1 = 17;
    const int SIZE_2 = 18;
    char string1[SIZE_1] = "Welcome ";
    char string2[SIZE_2] = "to North Carolina";

    cout << string1 << endl;
    cout << string2 << endl;
    maxChars = sizeof(string1) - (strlen(string1) + 1);
    strncat(string1, string2, maxChars);
    cout << string1 << endl;

    return 0;
}
```

The `strncat()` and `strncpy()` Functions

- The **`strncpy()`** function allows you to copy a specified number of characters from a string to a destination.
- Calling **`strncpy()`** is similar to calling **`strcpy()`**, except we pass a third argument specifying the maximum number of characters from the second string to copy to the first.
- The **`strncat()`** function will append no more than **`n`** characters from string2 to string1.
- If the specified number of characters is less than or equal to the length of **`string2`**, a null terminator is **not** appended to **`string1`**.
- If the specified number of characters is greater than the length of **`string2`**, then **`string1`** is padded with null terminators, up to the specified number of characters.

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
    int maxChars;
    const int SIZE = 11;
    char string1[SIZE];
    char string2[] = "I love C++ programming!";

    maxChars = sizeof(string1) - 1;
    strncpy(string1, string2, maxChars);
    // Put the null terminator at the end.
    string1[maxChars] = '\0';
    cout << string1 << endl;

    return 0;
}
```

The `strstr()` Function

- The **`strstr()`** function searches for a string inside of a string.
- The function's first argument is the string to be searched, and the second argument is the string for which to look.
- If the function finds the second string inside the first, it returns the address of the occurrence of the second string within the first string. Otherwise, it returns **`nullptr`**.
- The **`strstr()`** function can be useful in any program that must locate data inside one or more strings.

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
    char arr[] = "Four score and seven years ago";
    char* strPtr = nullptr;
    cout << arr << endl;
    strPtr = strstr(arr, "seven");
    if (strPtr != nullptr)
        cout << strPtr << endl;
    return 0;
}
```

Example

```
// This program uses the strstr function to search an array.
#include <iostream>
#include <cstring>      // For strstr
using namespace std;

int main()
{
    // Constants for array lengths
    const int NUM_PRODS = 5;    // Number of products
    const int LENGTH = 27;      // String length

    // Array of products
    char products[NUM_PRODS][LENGTH] =
    { "TV327 31 inch Television",
      "CD257 CD Player",
      "TA677 Answering Machine",
      "CS109 Car Stereo",
      "PC955 Personal Computer" };

    char lookUp[LENGTH];// To hold user's input
    char* strPtr = nullptr;  // To point to the found product
    int index;// Loop counter

    // Prompt the user for a product number.
    cout << "\tProduct Database\n\n";
    cout << "Enter a product number to search for: ";
    cin.getline(lookUp, LENGTH);

    // Search the array for a matching substring
    for (index = 0; index < NUM_PRODS; index++)
    {
        strPtr = strstr(products[index], lookUp);
        if (strPtr != nullptr)
            break;
    }

    // If a matching substring was found, display the product info.
    if (strPtr != nullptr)
        cout << products[index] << endl;
    else
        cout << "No matching product was found.\n";

    return 0;
}
```

The `strcmp()` Function

- Since C-strings are stored in **`char`** arrays, we **cannot** use the relational operators to compare two C-strings.
- To compare C-strings, we should use the library function **`strcmp()`**.
- This function takes two C-strings as arguments and returns an integer that indicates how the two strings compare to each other.

```
int strcmp(char* string1, char* string2);
```

- The function takes two pointer-to-**`char`** parameters and returns an integer result:
  - The result is **zero** if the two strings are equal on a character-by-character basis.
  - The result is **negative** if **`string1`** comes before **`string2`** in alphabetical order.
  - The result is **positive** if **`string1`** comes after **`string2`** in alphabetical order.

Example

```
// This program uses strcmp to compare the string entered
// by the user with the valid MP3 player part numbers.
#include <iostream>
#include <cstring>
#include <iomanip>
using namespace std;

int main()
{
    // Price of parts.
    const double A_PRICE = 99.0,
        B_PRICE = 199.0;

    // Character array for part number.
    const int PART_LENGTH = 9;
    char partNum[PART_LENGTH];

    // Instruct the user to enter a part number.
    cout << "The MP3 player part numbers are:\n"
        << "\t16 Gigabyte, part number S147-29A\n"
        << "\t32 Gigabyte, part number S147-29B\n"
        << "Enter the part number of the MP3 player you\n"
        << "wish to purchase: ";

    // Read a part number of at most 8 characters.
    cin >> partNum;

    // Determine what user entered using strcmp
    // and print its price.
    cout << showpoint << fixed << setprecision(2);
    if (strcmp(partNum, "S147-29A") == 0)
        cout << "The price is $" << A_PRICE << endl;
    else if (strcmp(partNum, "S147-29B") == 0)
        cout << "The price is $" << B_PRICE << endl;
    else
        cout << partNum << " is not a valid part number.\n";
    return 0;
}
```

Poll 3 (Extra Credit)

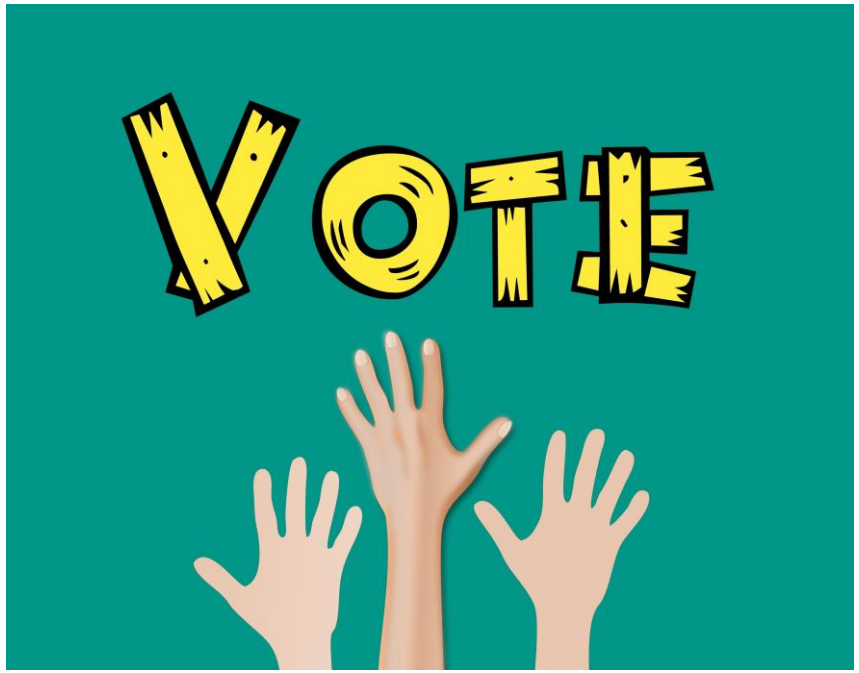
The following if statements perform the same operation. Both conditions return true if the two strings are identical.

```
if (strcmp(firstString, secondString) == 0)
if (!strcmp(firstString, secondString))
```

a) Yay!

b) Nay!

Please use the "Poll" window to participate for extra credit! One answer only please!



Summary of Useful <cstring> Functions

Function	Description
strlen	Accepts a C-string or a pointer to a C-string as an argument. Returns the length of the C-string (not including the null terminator.) *Example Usage:* len = strlen(name);
strcat	Accepts two C-strings or pointers to two C-strings as arguments. The function appends the contents of the second string to the first C-string. (The first string is altered, the second string is left unchanged.) *Example Usage:* strcat(string1, string2);
strcpy	Accepts two C-strings or pointers to two C-strings as arguments. The function copies the second C-string to the first C-string. The second C-string is left unchanged. *Example Usage:* strcpy(string1, string2);
strncat	Accepts two C-strings or pointers to two C-strings, and an integer argument. The third argument, an integer, indicates the maximum number of characters to copy from the second C-string to the first C-string. *Example Usage:* strncat(string1, string2, n);
strncpy	Accepts two C-strings or pointers to two C-strings, and an integer argument. The third argument, an integer, indicates the maximum number of characters to copy from the second C-string to the first C-string. If n is less than the length of string2, the null terminator is not automatically appended to string1. If n is greater than the length of string2, string1 is padded with '\0' characters. *Example Usage:* strncpy(string1, string2, n);
strcmp	Accepts two C-strings or pointers to two C-strings arguments. If string1 and string2 are the same, this function returns 0. If string2 is alphabetically greater than string1, it returns a negative number. If string2 is alphabetically less than string1, it returns a positive number. *Example Usage:* if (strcmp(string1, string2))
strstr	Accepts two C-strings or pointers to two C-strings as arguments. Searches for the first occurrence of string2 in string1. If an occurrence of string2 is found, the function returns a pointer to it. Otherwise, it returns nullptr (address 0). *Example Usage:* cout << strstr(string1, string2);

In-Class Exercise #1

Complete the following program skeleton.

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

int main()
{
    char place[] = "The Windy City";
    // Complete the program. It should search the array place
    // for the string "Windy" and display the message "Windy
    // found" if it finds the string. Otherwise, it should
    // display the message "Windy not found."
    return 0;
}
```

In-Class Exercise #1

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

int main()
{
    char place[] = "The Windy City";

    char* strPtr = strstr(place, "Windy");

    if (strPtr != nullptr)
        cout << "Windy found" << endl;
    else
        cout << "Windy not found" << endl;

    return 0;
}
```

In-Class Exercise #2

What will be the output of this program?

Modify it such that all array operations use the array offset [ ] operator.

```
#include <iostream>
using namespace std;
// Function Prototype
void mess(char*);
int main()
{
    char stuff[] = "Tom Talbert Tried Trains";
    cout << stuff << endl;
    mess(stuff);
    cout << stuff << endl;
    return 0;
}
// Definition of function mess
void mess(char* str)
{
    while (*str != '\0')
    {
        if (*str == 'T')
            *str = 'D';
        str++;
    }
}
```

In-Class Exercise #2

What will be the output of this program?

Modify it such that all array operations use the array offset [ ] operator.

```
#include <iostream>
using namespace std;
// Function Prototype
void mess(char[]);
int main()
{
    char stuff[] = "Tom Talbert Tried Trains";
    cout << stuff << endl;
    mess(stuff);
    cout << stuff << endl;
    return 0;
}
// Definition of function mess
void mess(char str[])
{
    int step = 0;
    while (str[step] != '\0')
    {
        if (str[step] == 'T')
            str[step] = 'D';
        step++;
    }
}
```

DOUGLAS COLLEGE

String/Numeric Conversion Functions

String/Numeric Conversion Functions

- There is a great difference between a number that is stored as a string, and one stored as a numeric value.
- The string **"26792"** is not a number in memory, but a series of ASCII codes representing the individual digits of the number.
- It uses 6 bytes of memory (including the null terminator).
- Since it is not a number in memory, we cannot perform mathematical operations with it, unless it is first converted to a numeric value.
- Several functions exist in the C++ library for converting **C-string** representations of numbers into numeric values.

Function	Description
atoi	Accepts a C-string as an argument. The function converts the C-string to an integer and returns that value. *Example Usage:* `int num = atoi("4569");`
atol	Accepts a C-string as an argument. The function converts the C-string to a long integer and returns that value. *Example Usage:* `long lnum = atol("500000");`
atof	Accepts a C-string as an argument. The function converts the C-string to a double and returns that value. *Example Usage:* `double fnum = atof("3.14159");`

DOUGLAS COLLEGE

More about the C++ `string` Class

The C++ `string` Class

- The **`string`** class is an **abstract data type**.
  - It is **not** a built-in, primitive data type like **`int`** or **`char`**.
  - It is a **programmer-defined** data type that accompanies the C++ language.
- It provides many capabilities that make storing and working with strings easy and intuitive.
- To define objects of the **`string`** class, we must **`#include`** the **`string`** class.

```
#include <string>
```

- The following statement defines two **`string`** objects.

```
string firstName, lastName;
```

- We can assign values to **`string`** objects using the assignment operator.

```
firstName = "Shahriar";
lastName = "Khosravi";
```

- The contents of string objects can be displayed onto the console screen using **`cout`**.

```
cout << "My favorite professor is " << firstName << " " << lastName << endl;
```

Input into a `string` Object

- We can use **`cin`** to read an item into a **`string`**:

```
cin >> firstName;
```

- If we want to **read a line of input with spaces** into a **`string`** object, we should use the **`getline()`** function.
- The **`getline()`** function's first argument is the name of a stream object from which we wish to read the input.
- The function call passes the **`cin`** object to **`getline()`**, so the function reads a line of input from the keyboard.
- The second argument is the name of a **`string`** object. This is where **`getline()`** stores the input that it reads.

```
string name;
cout << "What is your name? ";
getline(cin, name);
```

`string` Comparison

- There is no need to use a function (such as **`strcmp()`**) to compare **`string`** objects.
- W can use the **`<`**, **`>`**, **`<=`**, **`>=`**, **`==`**, and **`!=`** relational operators for **`string`** object comparisons.
- In the following example, **`set1`** is considered less than **`set2`** because the characters **`"ABC"`** alphabetically precede the characters **`"XYZ"`** .

```
string set1 = "ABC";
string set2 = "XYZ";

if (set1 < set2)
    cout << "set1 is less than set2.\n";
```

- Relational operators perform comparisons on **`string`** objects in a fashion similar to the way the **`strcmp()`** function compares C-strings, i.e. character-by-character.
- This allows us to sort **`string`** objects easily using relational operators.

Other Ways to Define `string` Objects

Definition	Description
`string address;`	Defines an empty **`string`** object named **`address`**.
`string name("William Smith");`	Defines a **`string`** object named **`name`**, initialized with "William Smith."
`string person1(person2);`	Defines a **`string`** object named **`person1`**, which is a copy of **`person2`**. **`person2`** may be either a **`string`** object or **`char`** array.
`string str1(str2, 5);`	Defines a **`string`** object named **`str1`**, which is initialized to the first five characters in the character array `str2`.
`string lineFull('z', 10);`	Defines a **`string`** object named **`lineFull`** initialized with 10 'z' characters.
`string firstName(fullName, 0, 7);`	Defines a **`string`** object named **`firstName`**, initialized with a substring of the string **`fullName`**. The substring is seven characters long, beginning at position 0.

Using `string` Class Member Functions

- The **`string`** class also has many useful member functions.
- For example, the **`length()`** member function returns the length of the **`string`**. The value is returned as an **`unsigned int`**.

Member Function Example	Description
`mystring.append(n, 'z')`	Appends n copies of 'z' to mystring.
`mystring.append(str)`	Appends str to mystring. str can be a string object or character array.
`mystring.append(str, n)`	The first n characters of the character array str are appended to mystring.
`mystring.append(str, x, n)`	n number of characters from str, starting at position x, are appended to mystring. If mystring is too small, the function will copy as many characters as possible.
`mystring.assign(n, 'z')`	Assigns n copies of 'z' to mystring.
`mystring.assign(str)`	Assigns str to mystring. str can be a string object or character array.
`mystring.assign(str, n)`	The first n characters of the character array str are assigned to mystring.
`mystring.assign(str, x, n)`	n number of characters from str, starting at position x, are assigned to mystring. If mystring is too small, the function will copy as many characters as possible.
`mystring.at(x)`	Returns the character at position x in the string.
`mystring.back()`	Returns the last character in the string. (This member function was introduced in C++ 11.)
`mystring.begin()`	Returns an iterator pointing to the first character in the string.
`mystring.c_str()`	Converts the contents of mystring to a C-string, and returns a pointer to the C-string.

Using `string` Class Member Functions

Definition	Description
`mystring.capacity()`	Returns the size of the storage allocated for the string.
`mystring.clear()`	Clears the string by deleting all the characters stored in it.
`mystring.compare(str)`	Performs a comparison like the strcmp function, with the same return values. str can be a string object or a character array.
`mystring.compare(x, n, str)`	Compares mystring and str, starting at position x, and continuing for n characters. The return value is like strcmp. str can be a string object or character array.
`mystring.copy(str, x, n)`	Copies the character array str to mystring, beginning at position x, for n characters. If mystring is too small, the function will copy as many characters as possible.
`mystring.empty()`	Returns true if mystring is empty.
`mystring.end()`	Returns an iterator pointing to the last character of the string in mystring.
`mystring.erase(x, n)`	Erases n characters from mystring, beginning at position x.
`mystring.find(str, x)`	Returns the first position at or beyond position x where the string str is found in mystring. str may be either a string object or a character array.
`mystring.find('z', x)`	Returns the first position at or beyond position x where 'z' is found in mystring. If 'z' is not found, the function returns the special value string::npos.

Using `string` Class Member Functions

Definition	Description
`mystring.front()`	Returns the first character in the string. (This member function was introduced in C++ 11.)
`mystring.insert(x, n, 'z')`	Inserts 'z' n times into mystring at position x.
`mystring.insert(x, str)`	Inserts a copy of str into mystring, beginning at position x. str may be either a string object or a character array.
`mystring.length()`	Returns the length of the string in mystring.
`mystring.replace(x, n, str)`	Replaces the n characters in mystring beginning at position x with the characters in string object str.
`mystring.resize(n, 'z')`	Changes the size of the allocation in mystring to n. If n is less than the current size of the string, the string is truncated to n characters. If n is greater, the string is expanded and 'z' is appended at the end enough times to fill the new spaces.
`mystring.size()`	Returns the length of the string in mystring.
`mystring.substr(x, n)`	Returns a copy of a substring. The substring is n characters long and begins at position x of mystring.
`mystring.swap(str)`	Swaps the contents of mystring with str.

Example

```
// This program lets the user enter a number. The
// dollarFormat function formats the number as
// a dollar amount.
#include <iostream>
#include <string>
using namespace std;

// Function prototype
void dollarFormat(string&);

int main()
{
    string input;

    // Get the dollar amount from the user.
    cout << "Enter a dollar amount in the form nnnnn.nn : ";
    cin >> input;
    dollarFormat(input);
    cout << "Here is the amount formatted:\n";
    cout << input << endl;
    return 0;
}

void dollarFormat(string& currency)
{
    int dp;

    dp = currency.find('.');    // Find decimal point
    if (dp > 3)                 // Insert commas
    {
        for (int x = dp - 3; x > 0; x -= 3)
            currency.insert(x, ",");
    }
    currency.insert(0, "$");    // Insert dollar sign
}
```

Thank you.

DOUGLAS COLLEGE

DOUGLAS COLLEGE