University of Wollongong in Dubai

UNIVERSITY
OF WOLLONGONG
IN DUBAI

CSCI251 Advanced Programming – Dr HC Lim/Dr Abdellatif Tchantchane

Coverage

- Review C++ Essentials
- Note:
  - C++ is governed by the ISO C++ standard. There are multiple ISO C++ standards listed here in chronological order: C++03, C++11, C++14, C++17, C++ 20, C++ 23 and the upcoming C++26 standard. Every C++ standard starting with the C++11 onwards is referred to as “Modern C++.”

review

- Classes and Objects
  - A class is a block of code that defines a data type. A class has a name that is the name for the type. An item of data of a class type is referred to as an object. You use the class type name when you create variables that can store objects of your data type. Being able to define your own data types enables you to specify a solution to a problem in terms of the problem.
  - If you were writing a program processing information about students, for example, you could define a Student type. Your Student type could incorporate all the characteristic of a student—such as age, gender, or school record—that was required by the program.

review

- Templates
  - You sometimes need several similar classes or functions in a program where the code differs only in the kind of data that is processed. A template is a recipe that you create to be used by the compiler to generate code automatically for a class or function customized for a particular type or types.
  - The compiler uses a class template to generate one or more of a family of classes. It uses a function template to generate functions.
  - Each template has a name that you use when you want the compiler to create an instance of it. The Standard Library uses templates extensively.

review

- Procedural and Object-Oriented Programming
  - Historically, procedural programming is the way almost all programs were written. To create a procedural programming solution to a problem, you focus on the process that your program must implement to solve the problem. Here is a rough outline of what you do, once the requirements have been defined precisely:
    - You create a clear, high-level definition of the overall process that your program will implement.
    - You segment the overall process into workable units of computation that are, as much as possible, self-contained. These will usually correspond to functions.
    - You code the functions in terms of processing basic types of data: numerical data, single characters, and character strings.

C++ programming

- Programming Task:
  - Your body mass index (BMI) is your weight, w, in kilograms divided by the square of your height, h, in meters (w/(h*h)). Write a program to calculate the BMI from a weight entered in pounds and a height entered in feet and inches. a kilogram is 2.2 pounds, and a foot is 0.3048 meters.
- Understand the Problem
  - The first step in program design is to understand the problem. We begin by reading the requirements statement carefully. When we fully understand it, we review our understanding with the user. Often this involves asking questions to confirm our understanding. For example, after reading our simple requirements statement, we should ask several clarifying questions
  - What is in this problem?

C++ programming

- Develop the Solution
  - Once we fully understand the problem and have clarified any questions we may have, we develop a solution in the form of an algorithm. An algorithm is a set of logical steps necessary to solve a problem. Algorithms have two important characteristics: first, they are independent of the computer system. This means that they can be used to implement a manual system in an office as well as a program in a computer. Second, an algorithm accepts data as input and processes the data into an output.
  - To write the algorithm for our problem, we use an intuitive approach, calling not only on the problem statement but also our knowledge and experience.
- What is required in the given programming task? Using procedural programming and using Object-oriented programming?

C++ programming

- Procedural Programming:
  - Get inputs (weight and height)
  - Convert from Imperial to metrics scales
  - Compute BMI
  - Display results
- Object-Oriented Programming
  - Identify class: data? Methods? Use class diagram. What can be a possible class for this programming task? BMI class?

C++ programming

- Class: BMI
- Data:
  - int weight, float height
- Method:
  - Ctor;
  - convertImperials()
  - computeBMI();

C++ programming

- Procedural BMI

```
31  void ProceduralBMI()
32  {
33      float w{ 90.0 }; float h{ 6.2 };
34      float& wRef = w;
35      float& hRef = h;
36      GetInputs(wRef, hRef);
37      int wImperial = w;
38      float hImperial = h;
39      ConvertImperial(wRef, hRef);
40      cout << "Your BMI with weight - " << wImperial << " pounds and height - " << hImperial;
41      cout << " feet is : " << ComputeBMI(wRef, hRef) << std::endl;
42
43      // Test data: 90 pounds, height = 6.2 feet
44
45      /*output:
46      90
47      6.2
48      Your BMI with weight - 90 pounds and height - 6.2 feet is : 11.2007
49      */
50
51  }
52
```

C++ programming

- Procedural BMI

```
13  void GetInputs(float& wt, float& ht)
14  {
15      wt = 90.0;
16      ht = 6.2;
17      //std::cin >> wt >> ht;
18  }
19
20  void ConvertImperial(float& wt, float& ht)
21  {
22      wt = (wt / 2.2);
23      ht = (ht * 0.3048);
24  }
25
26  float ComputeBMI(const float& wt, const float& ht)
27  {
28      return (float)(wt / (ht * ht));
29  }
```

C++ programming

- BMI.h

```
1   #pragma once
2
3   #ifndef BMI_H
4   #define BMI_H
5
6   class BMI
7   {
8   private:
9       int weight;
10      float height;
11
12  public:
13      BMI();
14      BMI(int wt, float ht);
15      BMI(const BMI& bmi);
16      void setWt(int w);
17      void setHt(float h);
18      int getWt();
19      float getHt();
20      void convertImperial(int w, float h);
21      int computeBMI();
22  };
23
24  #endif
```

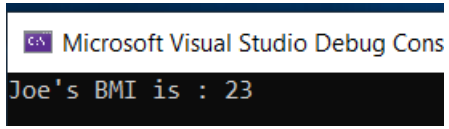
C++ programming

- BMI.cpp

```cpp
#include "BMI.h"

BMI::BMI() : weight(0), height(0.00) { }
BMI::BMI(int wt, float ht) : weight(wt), height(ht) { }
BMI::BMI(const BMI& bmi) : weight(bmi.weight), height(bmi.height) { }

void BMI::setWt(int w)
{
    weight = w;
}
void BMI::setHt(float h)
{
    height = h;
}
int BMI::getWt()
{
    return weight;
}
float BMI::getHt()
{
    return height;
}
void BMI::convertImperial(int w, float h)
{
    setWt(w / 2.2);
    setHt(h * 0.3048);
}
int BMI::computeBMI()
{
    return (weight / (height * height));
}

```

C++ programming

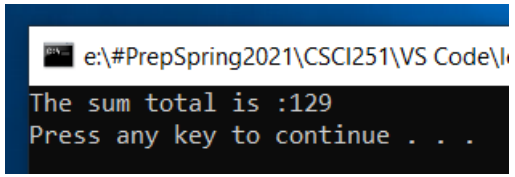
- Application file (main.cpp)




```cpp
/*****************************************************************
* The application file to test BMI class                         *
*****************************************************************/
#include "BMI.h"
#include <iostream>

using namespace std;

int main()
{
    BMI joe;
    int w = 170;
    float h = 6.0;
    joe.convertImperial(w, h);

    cout << "Joe's BMI is : " << joe.computeBMI() << endl;
    return 0;
}

// Test data: 170 pounds and 6 feet, the BMI is 23

```

C++ programming

- Range-based for loop
  - The range-based for loop iterates over all the values in a range of values.
  - Syntax:
    - for (range_declaration : range_expression)
    - loop statement or block;




```cpp
/*************************************************************
* Program to demonstrate a range-based for loop              *
*************************************************************/
#include <iostream>
using namespace std;

int main()
{
    int values [] {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
    int total {};

    // Range-based for loop
    for (int x : values)
    total += x;
    cout << "The sum total is :" << total << endl;
    return 0;
}
```

C++ programming

- Alternative way to code a range-based for loop:
- Using the auto keyword causes the compiler to deduce the correct type for x. The auto keyword is used often with the range-based for loop. This is a nice way of iterating over all the elements in an array or other kind of range. You don't need to be aware of the number of elements. The loop mechanism takes care of that.
- Note that the values from the range are assigned to the range variable, x. This means you cannot modify the elements of values by modifying the value of x. For example, this doesn't change the elements in the values array:

```
e:\#PrepSpring2021\CSCI251\VS Code\lect
The sum total is :129
The sum total2 is :129
Press any key to continue . . .
```

```cpp
/*******************************************************
* Program to demonstrate a range-based for loop        *
*******************************************************/
#include <iostream>
using namespace std;

int main()
{
    int values [] {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
    int total {};

    // Range-based for loop
    for (int x : values)
    total += x;
    cout << "The sum total is :" << total << endl;

    // Another compact way to write
    // a range-based for loop
    int total2 {};
    for (auto x : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29})
    total2 += x;
    cout << "The sum total2 is :" << total2 << endl;
    return 0;
}
```

C++ programming

- Basic Function Objects
  - A function object or functor is simply an object that can be called as if it were a function. The key in constructing one is to overload the function call operator().
  - To see how this is done, we will define a class of function objects that encapsulate this simple function:
    - bool less(int one, int other) { return one < other; }

C++ programming

- Here is how you define a class with an overloaded function call operator:

```
1   #pragma once
2
3   // Less.h - A basic class of functor objects
4   #ifndef LESS_H
5   #define LESS_H
6   class Less
7   {
8   public:
9       bool operator()(int a, int b) const;
10  };
11  #endif // LESS_H
12
13  // Less.cpp - definition of a basic function call operator
14  bool Less::operator()(int a, int b) const
15  {
16      return a < b;
17  }
18
```

C++ programming

- Sample code to use the functor object

```
#include "less.h"
#include <iostream>

int main()
{
    Less less;                          // Create a 'less than' functor...
    const bool is_less = less(5, 6); // ... and 'call' it
    std::cout << (is_less ? "5 is less than 6" : "Huh?") << std::endl;
    return 0;
}

```

C++ programming

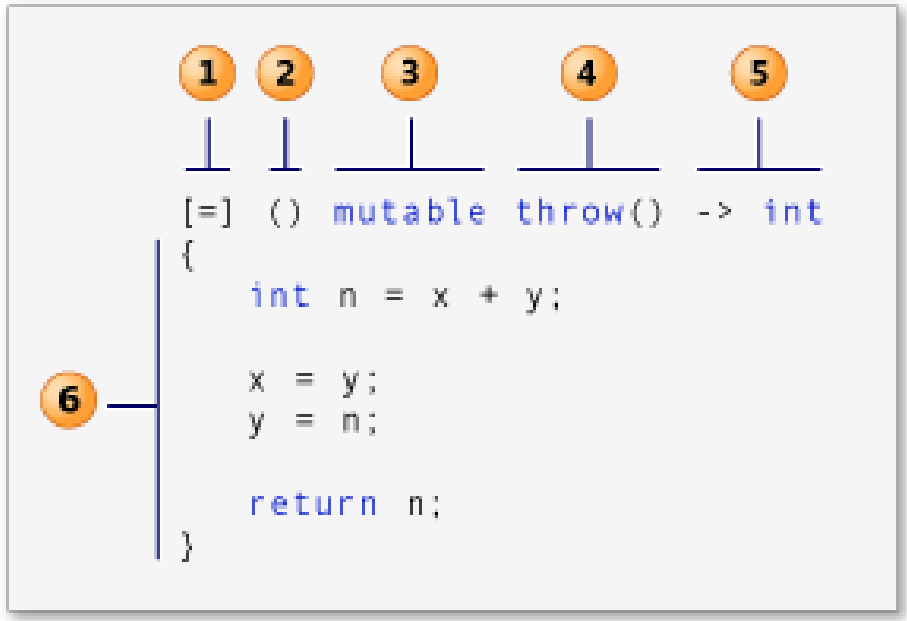
- About Lambda programming (see https://docs.microsoft.com/en-us/cpp/cpp/lambda-expressions-in-cpp?view=msvc-170)
- In C++11 and later, a lambda expression—often called a *lambda*—is a convenient way of defining an anonymous function object (a *closure*) right at the location where it is invoked or passed as an argument to a function.
- Typically, lambdas are used to encapsulate a few lines of code that are passed to algorithms or asynchronous methods.
- We will briefly cover what lambdas are, compares them to other programming techniques, describes their advantages, and provides a basic example.

C++ programming

- Lambda Expressions
  - A lambda expression is a compact way of creating a function object without having to write a class declaration. It is an expression that contains only the logic of the object's operator() member function. When the compiler encounters a lambda expression, it automatically generates a function object in memory, using the code that you provide in the lambda expression for the operator() member function.
  - Typically, a lambda expression takes the following general format:
    - [ ] (parameter list) { function body }

C++ programming

- Essential Syntax of a lambda:
- 1. capture clause (Also known as the lambda-introducer in the C++ specification.)
- 2. parameter list Optional. (Also known as the lambda declarator)
- 3. mutable specification Optional.
- 4. exception-specification Optional.
- 5. trailing-return-type Optional.
- 6. lambda body.



C++ programming

- Lambdas are lightweight, nameless functions that can be defined just-in-place where they are used.
- Lambdas didn't come to C++ as a specific-purpose feature (i.e., functional programming):
- The existing STL components, like the for_each algorithm, and its new ones like <thread> or <future>, allow you to specify function arguments as lambdas beside C-like functions and functors.
- For more information, see: https://docs.microsoft.com/en-us/cpp/cpp/examples-of-lambda-expressions?view=msvc-160

C++ programming

- Typically, a lambda expression takes the following general format:
  - [ ] (parameter list) { function body }

- The [ ] at the beginning of the expression is known as the lambda introducer. It marks the beginning of a lambda expression. The parameter list is a list of parameter declarations for the function object's operator() member function, and the function body is the code that should be the body of the object's operator() member function.
- For example, the lambda expression for a function object that computes the sum of two integers is:
  - [ ](int a, int b) { return a + b; }

C++ programming

- And, the lambda expression for a function object that determines whether an integer is even is:
  - [ ] (int x) { return x % 2 == 0;}
- As another example, the lambda expression that takes an integer as input and prints the square of that integer is written like this:
  - [ ] (int a) { std::cout << a * a << " "; }
- Note:
  - Optionally, a capture list can appear inside the brackets [ ] of a lambda expression. A capture list is a list of variables in the surrounding scope of the lambda expression that can be accessed from the lambda's function body

C++ programming

- When you call a lambda expression, you write a list of arguments, enclosed in parentheses, right after the expression. For example, the following code snippet displays 7, which is the sum of the variables x and y.

```
1   // Sample code to show the use of lambda expression
2   // to add two variables, x and y
3
4   #include <iostream>
5
6   int main()
7   {
8       int x = 2;
9       int y = 5;
10      std::cout << [](int a, int b) {return a + b;}(x, y) << std::endl;
11
12  }
```

```
i:\##000LectPrepSpring2021\01 CSCI251\02 LectPrep\wk09\LectPrep09Code\LectPrep09.exe
7
Press any key to continue . . .
```

C++ programming

- The following code segment counts the even numbers in a vector, (code here shows firstly program without using lambda expression and a program using lambda expression, ie it uses lambda expression in place of the IsEven() function object as the third argument to the count_if() function.
- We first show the IsEven.h

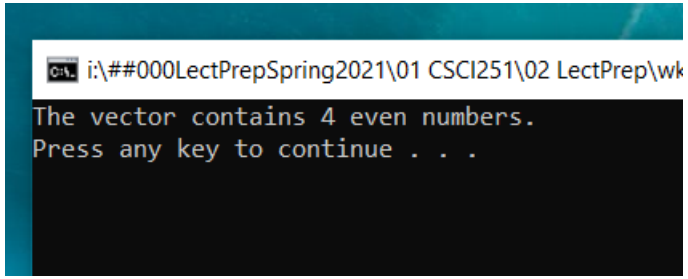
```
1   //IsEven.h - the function object or functor
2
3   #ifndef IS_EVEN_H
4   #define IS_EVEN_H
5
6   class IsEven
7   {
8       public:
9       bool operator()(int x)
10      { return x % 2 ==0; }
11  };
12  #endif
13
14
```

C++ programming

- This example, using the count_if() function. The count_if() function iterates over a range of elements, passing each element as an argument to a function that you provide. The function that you provide must accept one argument, and return either true or false.
- You provide the function as either a function pointer, or a function object. The count_if function returns the number of elements for which function returns true.
- We will use a function object along with the count_if() function to count the number of even numbers that appear in a vector.
- Our function object will be an instance of the IsEven class, shown in the previous slide (IsEven.h).
- This program shows codes without using lambda expressions

C++ programming

- Code without lambda expressions.




```cpp
// Program showing codes without using lambda expressions
#include <iostream>
#include <vector>
#include <algorithm>
#include "IsEven.h"

//using namespace std;


int main()
{
    // Create a vector of ints.
    std::vector<int> v = { 1, 2, 3, 4, 5, 6, 7, 8 };

    // Create an instance of the IsEven class.
    IsEven isNumberEven;

    // Get the number of elements that even.
    int evenNums = std::count_if(v.begin(), v.end(), isNumberEven);

    // Display the results.
    std::cout << "The vector contains " << evenNums << " even numbers." << std::endl;
    return 0;
}

```

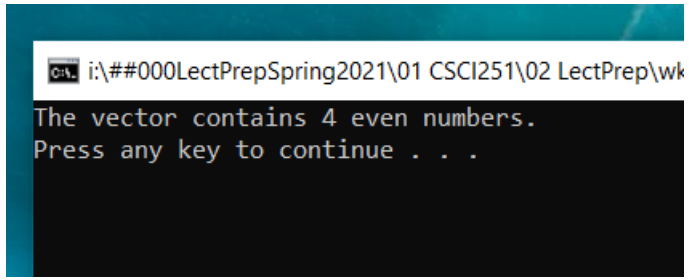
C++ programming

- Note:
  - This code can be improved using anonymous function object. That is, objects that are created and used without being given a name are said to be anonymous. In this code snippet, IsEven() is the anonymous function object, and IsEven() (2) is the expression that calls the function object and passes it the argument 2 as in:

```
if (IsEven()(2))
{
        cout << "The number is even.\n";
}
else {
        cout << "The number is not even.\n";
}
```

C++ programming

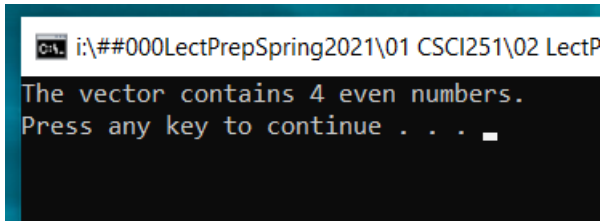
- Code without lambda expressions but using anonymous function object, IsEven().




```cpp
// Program showing codes without using lambda expressions
// using anonymous function object, IsEven().
#include <iostream>
#include <vector>
#include <algorithm>
#include "IsEven.h"

//using namespace std;


int main()
{
    // Create a vector of ints.
    std::vector<int> v = { 1, 2, 3, 4, 5, 6, 7, 8 };

    // Create an instance of the IsEven class.
    IsEven isNumberEven;

    // Get the number of elements that even.
    int evenNums = std::count_if(v.begin(), v.end(), IsEven());

    // Display the results.
    std::cout << "The vector contains " << evenNums << " even numbers." << std::endl;
    return 0;
}
```

C++ programming

- Code show the use of a lambda expression in place of the IsEven() function object as the third argument to the count_if() function



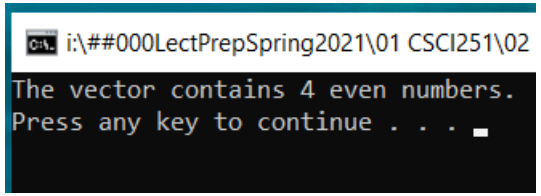

```cpp
// Code using lambda expressions

#include <iostream>
#include <vector>
#include <algorithm>

int main()
{
    std::vector<int> v = { 1, 2, 3, 4, 5, 6, 7, 8 };

    // Get the number of elements that are even.
    int evenNums = std::count_if(v.begin(), v.end(), [](int x) {return x % 2 == 0;});

    // Display the results.
    std::cout << "The vector contains " << evenNums << " even numbers." << std::endl;

    return 0;
}

```

C++ programming

- Because lambda expressions generate function objects, you can assign a lambda expression to a variable of a suitable type and then call it through the variable's name. For example, the lambda expression shown in the following code gives us a function object that adds two integers. The function object is assigned to a variable named sum:
- auto sum = [ ](int a, int b) {return a + b;};
- Once the statement has executed, we can use the sum variable to call the function object, as shown here:
  - int x = 2;
  - int y = 5;
  - int z = sum(x, y);
- After this code has executed, the variable z will be assigned 7

C++ programming

- Program shows an example where a lambda expression (that determines whether a value is even) is assigned to a variable named isEven, and that variable is passed as the third argument to the count_if() function.




```cpp
// Code using lambda expressions

#include <iostream>
#include <vector>
#include <algorithm>

int main()
{
    // Create a vector of ints.
    std::vector<int> v = { 1, 2, 3, 4, 5, 6, 7, 8 };

    // Use a lambda expression to create a function object.
    auto isEven = [](int x) {return x % 2 == 0; };

    // Get the number of elements that are even.
    int evenNums = count_if(v.begin(), v.end(), isEven);

    // Display the results.
    std::cout << "The vector contains " << evenNums << " even numbers.\n";

    return 0;
}
```

summary

- We have covered some C++ essentials
- We have briefly introduced lambda expressions that is built upon functions and anonymous function objects
- Function objects are so useful that the C++ library defines a number of classes that you can instantiate to create function objects in your program. To use these classes, you must #include the <functional> header file. For more information, see https://docs.microsoft.com/en-us/cpp/standard-library/functional-functions?view=msvc-160