C++20 Labs Part 2

Labs Introduction

This course introduces a number of new features. It is intended that these labs allow exploration of the features in isolation. Therefore, depending upon the development environment used create new projects or new online sessions if using an online compiler. When ‘project’s are created check that the appropriate compiler flags are set to support C++20.

Threading

jthread

26) Create a new project and within the main function declare a ‘jthread’ and initialize it with a lambda to output a sequence of numbers to the console.

Build and test.

Stop Token

27) Create a new project and within the main function declare a ‘jthread’ and initialize it with a lambda to output a sequence of numbers to the console. Add an initial parameter to the lambda taking a ‘std::stop_token’. Within the loop check the ‘stop_token’ for stop_requested. If stop is requested output a message and drop out of the loop.

Within main declare a ‘std::stop_callback’ and initialize with a the result of calling ‘get_stop_token’ on the thread and a lambda to output a message regarding stop.

Declare a variable for a stop source and call the ‘get_stop_source’ member function on the thread. Using the stop source call the ‘request_stop’ member function.

Build and test.

©Copyright Allotropical Ltd. 1998-2024 All rights reserved.

Barrier

28) Create a new project. Declared an array of string messages to be used for output from threads.

Declare a variable to hold a lambda for messages to indicate completion.

Within the main function define a lambda (capturing a std::barrier by reference) and taking a std::string as parameter, to execute the sequence:

1) output a message;
2) call ‘arrive_and_wait’ on the barrier;
3) output another message and
4) call ‘arrive_and_wait’ on the barrier;

Create a vector of thread and populate it with a number of jthread initialized with the lambda, passing a message from the array of string.

Build and test.

Atomic

29) Create a new project. Declare an atomic_int initialized to zero. Declare two jthreads called producer and consumer capturing the atomic by reference.

Within the lambda for the producer thread store a counted value within the atomic and notify all. Then sleep for a second.

Within the lambda for the consumer thread wait on the atomic and then load the value. Output the value.

Build and test.

©Copyright Allotropical Ltd. 1998-2024 All rights reserved.

Miscellaneous Features

Create a new project to explore the use of a number of miscellaneous features.

Using Enum

30) Define a Scoped Enum with a number of values. Within main declare a variable of the enum type and add a switch statement to select out on the enum value. Add output to each of the case statements.

Build and test.

30a) Add a using enum within the switch statement to simplify the selection.

Build and test.

Lambda Capture

31) Create a class with a member function. Within this member function create a lambda expression to use [=]. Confirm that in C++20 if 'this' is required it would need to be added to the lambda introducer separately.

Constexpr and consteval (Try within Compiler Explorer)

32) Define a factorial function. Within main call this function to initialize a variable passing a literal value. Output the result.

Build and test.

32a) Add constexpr in front of this function.

Try to build, what happens? If necessary, add constexpr before initializing the variable within main.

Build and test.

32b) Change constexpr to consteval.

Build and test.

Try passing a variable to the function. What happens?

©Copyright Allotropical Ltd. 1998-2024 All rights reserved.

Designated Initialisers

33) Create a simple class/struct with a number of public data members. Within the main function declare and try initializing an object using designated initializers.

Build and test.

Conditional Explicit

34) Create a template class with one template type parameter and a member variable of this type. Add a constructor taking a single value of the template parameter type. Add a type conversion operator to convert to the template parameter type. Add conditional explicit to the type conversion operator using a type trait.

Within the main function declare and initialize an object. Assign this object to a variable of the template parameter type.

Build and test.

Try a number of different types. What is the behaviour?

Likely and Unlikely Attributes - If Statement

35) This feature may be the most difficult to confirm working! Try writing an if statement but adding the use of [[unlikely]].

Build and test.

Likely and Unlikely Attributes - Switch

35b) This feature may be the most difficult to confirm working! Try writing a switch statement but adding the use of [[likely]] and [[unlikely]].

Build and test.

©Copyright Allotropical Ltd. 1998-2024 All rights reserved.

Spaceship Operator

This section is an opportunity to investigate the spaceship operator. Create a new project to explore the use of the spaceship operator.

Create a Simple class

36) Define a class with a single (simple) data member. Add a constructor to initialize this member variable. Specify that the spaceship operator should be given its default implementation.

Within main declare and initialise a number of objects and try comparing these using '<', '<=', '>' and '>='.

Build and test.

Code Insights Website (https://cppinsights.io/)

37) Use code insights to view the generated code.

37a) What happens when comparing for inequality?

37b) Investigate operator symmetry by using a number of comparisons where the left hand operand is a value and not an object of the class type. This requires that there is a converting constructor on the class.

©Copyright Allotropical Ltd. 1998-2024 All rights reserved.

Create a Class for Comparison

38) Define a new class with two simple data members (int or double). Add a constructor to initialize these member variables. Specify that the spaceship operator should be given its default implementation.

Write a program to populate many objects and output the result of comparing these objects using either of the operators '<' or '>'.

Build and test.

What conclusions can be taken from the results?

Custom Spaceship Operator

39) Define a new class with a simple data member (int or double). Add a constructor to initialize the member variables. Define a custom spaceship operator for this class to return an appropriate value dependent upon comparison of the data member.

Within main declare and initialise a number of objects and try comparing these using '<', '<=', '>' and '>='.

Build and test.

Use Spaceship Operator Directly

40) The previous examples have not required the direct use of the '<=>' operator. Try taking the result of using the '<=>' operator and use this to test for greater than or less than using the 'ordering' type.

Build and test.

Use Comparison Functions

41) Try taking the result of using the '<=>' operator and use this to test for greater than or less than using the ordering functions.

Build and test.

©Copyright Allotropical Ltd. 1998-2024 All rights reserved.

Bind and Bind Front

42) Define a function taking four parameters and returning some composite of these values. Call this function and output the value returned.

Build and test.

Use ‘std::bind’ to bind values to the first two parameters. Call the resulting ‘callable’ and output the result.

Build and test.

42a) Use ‘std::bind_front’ to achieve the same effect as the previous step..

Build and test. Try also using a member function with ‘std::bind_front’.

Source Location

43) Define a function to ‘log’ to the console taking a string for a message. Define a default argument to initialize a parameter with current location. Output the message and location from within this function.

Call this function from within main.

Build and test

Shared Pointer for Arrays

44) Define a class with a destructor containing output to indicate that the destructor was called. Store a dynamically allocated array of these objects within a shared_ptr.

Build and test.

Confirm that the items within the array have their destructors called.

Range-based For Initialisation

45) Define a function returning a vector of int. Within the main function use a range-based for, to iterate over the vector returned by the function. Initialize a local variable from the function within the range-based for and output the values to the console.

Build and test.

©Copyright Allotropical Ltd. 1998-2024 All rights reserved.

Removing Items from a Collection

46) Declare a vector of integer and initilise with a number of values. Use the remove algorithm to ‘remove’ some items (move the remaining items). Output the contents of the vector before and after the ‘remove’.

Build and test.

Use erase member function to remove the unwanted items at the end of the vector. Output the contents of the vector after this ‘erase’.

Build and test.

46a) Use the erase algorithm to remove items from the collection. Output the contents of the vector before and after the ‘erase’.

Build and test.

©Copyright Allotropical Ltd. 1998-2024 All rights reserved.

Format Library

This is one of the parts of the Standard Library which is most challenging to try out, as library support is not currently available with many compilers.

Create a new project to explore the use of the format library (if it is available).

Formatted Output

47) Declare a number of variables of different types and use ‘format’ to output these values in the default manner (#include <format> if available).

Build and test.

Formatted Output using String View

48) Declare a number of variables of different types and use ‘vformat’ to output these values in the default manner (#include <format> if available). Pass the parameters by using std::make_format_args.

Build and test.

Format Integral Types

49) Declare a number of integral variables and use ‘format’ to output these values using various bases (binary, octal, decimal and hexadecimal).

Build and test.

Format Floating Types

50) Declare a number of floating point variables and use ‘format’ to output these values using various widths and precisions.

Build and test.

©Copyright Allotropical Ltd. 1998-2024 All rights reserved.