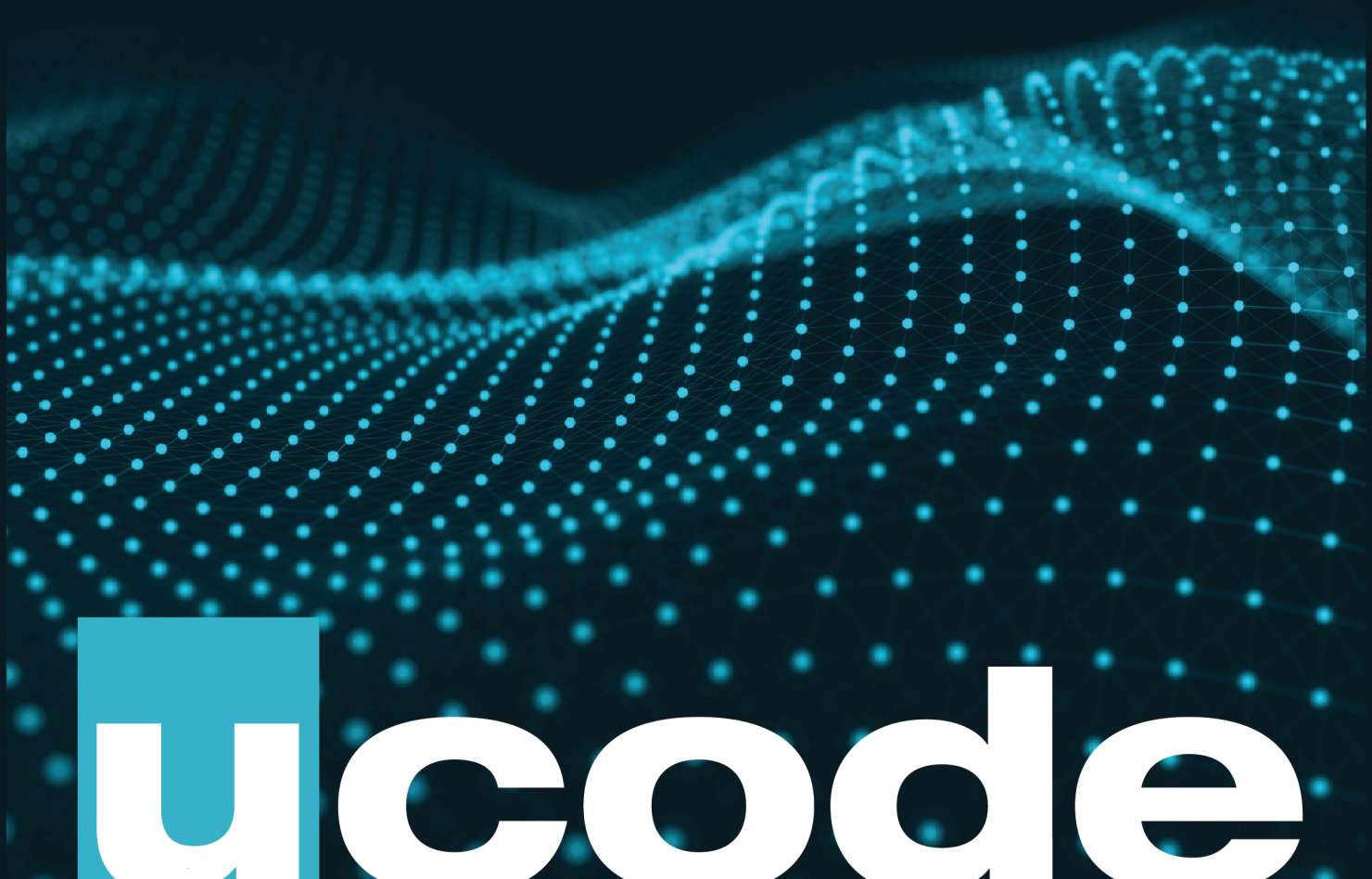
Sprint 08

Half Marathon C++

August 25, 2020



ucode

Contents

ucode

Engage

DESCRIPTION

Hello!

This is already Sprint08, which means that you have done a great job! You already know what OOP and STL are, also you can write your own templates and classes.

But C++ is very extensive. And more importantly, C++ is a "live" language. It constantly evolves, trying to make your life easier. And as a programmer, you are always responsible for keeping up-to-date with new developments.
Be on your guard at all times.

Each feature appears for a reason. Every update is aimed at giving you something useful. In this Sprint, you have an opportunity to explore cool features for manipulations with different kinds of data. They were created for language flexibility and your convenience.

Remember, your mind is the best weapon you have.
Good luck!

BIG IDEA

Advanced programming language features.

ESSENTIAL QUESTION

Which high-level opportunities are available in C++?

CHALLENGE

Replenish your arsenal with progressive skills.

Investigate

GUIDING QUESTIONS

We invite you to find answers to the following questions. By researching and answering them, you will gain the knowledge necessary to complete the challenge. To find answers, ask the students around you and search the internet. We encourage you to ask as many questions as possible. Note down your findings and discuss them with your peers.

- How do programming languages evolve?
- When to use `shared pointers`?
- When to use `unique pointers`?
- When to use `smart pointers`?
- What are the use cases of an `std::bind`?
- What is the difference between `std::bind` and `boost::bind`?
- What can be done with `std::initializer_list`?
- What is the purpose of the `parameter pack`?
- What kind of information can be obtained using the `typeid operator`?
- Why do we need type conversion?
- Which tools are available in C++ to do type conversion?

GUIDING ACTIVITIES

Complete the following activities. Don't forget that you have a limited time to overcome the challenge. Use it wisely. Distribute tasks correctly.

- Read the tasks carefully and try to find as much information as possible about them.
- Consider the algorithms found in the tasks.
- Allocate your resources and time.
- Arrange to brainstorm tasks with other students.
- Clone your git repository that is issued on the challenge page in the LMS.
- Push solutions to the repository.

ANALYSIS

Analyze your findings. What conclusions have you made after completing guiding questions and activities? In addition to your thoughts and conclusions, here are some more analysis results.

- Be attentive to all statements of the story.
- Analyze all information you have collected during the preparation stages. Try to define the order of your actions.
- Perform only those tasks that are given in this document.
- Submit your files using the layout described in the story. Only useful files allowed, garbage shall not pass!

- Tasks in `shell` must be executed with `zsh`.
- Compile files with commands `cmake . -Bbuild && cmake --build ./build` that will call `CMake` and build an app.
- Pay attention to what is allowed in a certain task. Use of forbidden stuff is considered a cheat and your tasks will be failed.
- Complete tasks according to the rules specified in the Google C++ Style Guide.
  But there are several exceptions for the guide listed below:
  - you can use `#pragma once` directive instead of `#ifndef ... #define`
  - variables can be written in `mixed case`
  - class data members must begin with `m_` prefix (m for member)
  - indent 4 spaces at a time
  - each line of text in your code must be at most 120 characters long
  - ignore the sections Inputs and Outputs and Legal Notice and Author Line in the style guide
- The solution will be checked and graded by students like you. Peer-to-Peer learning.
- You must use this project structure for each task:

```
>tree project_dir --dirsfirst --charset=ascii
project_dir
|-- app
|   |-- src
|   |   |-- CMakeLists.txt
|   |   `-- ...
|   |-- resources
|   |   `-- ...
|   |-- CMakeLists.txt
|   `-- main.cpp
|-- lib1
|   |-- CMakeLists.txt
|   `-- ...
...
|-- libN
|   |-- CMakeLists.txt
|   `-- ...
`-- CMakeLists.txt
```

- If you have any questions or don't understand something, ask other students or just Google it.
- In the name of Talos, use your brain!

Act: Task 00

NAME

Smart Pointers

DIRECTORY

`t00/`

BINARY

`smartPointers`

DESCRIPTION

Create a console program that imitates a game map with a player and obstacles in the console:

- `Map` class prints the game map to the standard output
- `Player` class represents the player of the game. The player walks on the map in 4 main directions: up, down, left and right
- the map knows the player's position when printing the map in the console, but the player knows nothing about the map
- do not worry, there is a `MoveManager` class that knows all the positions and moves the player around on the map.

The main points to consider when implementing a solution.

1. Describe all classes of `.h` files in `.cpp` files following this description:
   `Player` class:
   - contains X and Y positions of the player
   - member functions:
     - `movePlayer` changes the player's position according to the direction
     - `getIdentifier` returns a `P` character as the player identifier

   `Map` class:
   - contains the size of the map (width, height), a pointer to the `Player` and the map
   - member functions:
     - `width` and `height` return (respectively) the width and the height of the map
     - `outputMap` prints the map to the console
     - `generateMap` generates the map with trees `T` and shrubs `@` using the algorithm of your choice

   `MoveManager` class:
   - contains pointers to `Map` and `Player` classes

- has the enum class of directions
- member functions:
  - `processInputAndMove` validates input commands from the console and moves the player
  - `checkMove` checks whether the player can move to a certain direction on the map

2. Player can move only in 4 directions using commands: `u` (up), `d` (down), `r` (right), `l` (left)
3. `movePlayer` member function of `Player` class changes objects `m_posX` or `m_posY` according to the `Direction` enumerator
4. Additional command `e` (exit) for exit from the program must be implemented
5. Width and height of the map must be in a range of 5-20
6. Memory management must be implemented ONLY using smart pointers of classes

Error handling. The program prints:

- if the number of command-line arguments is not in the valid range, or width|height of the map is not a digit - `usage: ./smartPointers [width] [height]`
- if the width or height of the map is not in the range of min and max size of the map - `Invalid map size`
- if input commands are invalid - `Invalid input` and the last state of the map
- if the input command of direction is valid but the player can't move to this direction - `Invalid direction` and the last state of the map

SYNOPSIS

```
/* Player.h */

class Player final {
 public:
    Player() = default;
    ~Player() = default;

    void movePlayer(MoveManager::Direction dir);
    char getIdentifier() const;
    size_t posX() const;
    size_t posY() const;

 private:
    size_t m_posX{0};
    size_t m_posY{0};
};

/* Map.h */
```

```cpp
class Map final {
 public:
    Map(size_t width, size_t height, std::shared_ptr<Player>& player);
    ~Map() = default;

    void outputMap() const;
    size_t width() const;
    size_t height() const;

 private:
    void generateMap();

    const size_t m_width{0};
    const size_t m_height{0};
    std::shared_ptr<Player> m_player;
    std::unique_ptr<char[]> m_map;
};

/* MoveManager.h */

class MoveManager final {
 public:
    enum class Direction { Up, Down, Left, Right };

    MoveManager(std::shared_ptr<Player>& player, std::shared_ptr<Map>& map);
    ~MoveManager() = default;

    void processInputAndMove(const std::string& inputStr);

 private:
    bool checkMove(Direction dir) const;

    std::shared_ptr<Map> m_map;
    std::shared_ptr<Player> m_player;
};
```

CONSOLE OUTPUT

```
>./smartPointers | cat -e
usage: ./smartPointers [width] [height]
>./smartPointers 2 5 | cat -e
Invalid map size
>./smartPointers 5 5
P T . . @
@ . T . T
. . . @ @
. . T . T
T . T . T
:> 1
Invalid direction
P T . . @
@ . T . T
```

```
. . . @ @
. . T . T
T . T . T
:> u
Invalid direction
P T . . @
@ . T . T
. . . @ @
. . T . T
T . T . T
:> r
@ P . . @
@ . T . T
. . . @ @
. . T . T
T . T . T
:> d
@ T . . @
@ P T . T
. . . @ @
. . T . T
T . T . T
:> l
@ T . . @
P . T . T
. . . @ @
. . T . T
T . T . T
:> qwerty
Invalid input
@ T . . @
P . T . T
. . . @ @
. . T . T
T . T . T
:> e
>
```

SEE ALSO

Shared Pointer
Unique Pointer
Smart Pointers

Act: Task 01

NAME

Class With Initializer List

DIRECTORY

`t01/`

BINARY

`classWithInitializerList`

DESCRIPTION

Create a program that implements `ClassWithInitializerList` class and has an identical output with CONSOLE OUTPUT using a `main.cpp` from SYNOPSIS.

- `ClassWithInitializerList` class has three constructors that are implemented with three different types of arguments: `parameter pack`, `std::initializer_list` and `std::vector`
- All three constructors initialize `m_vecOfArs` with given arguments
- `outputVector()` member function prints `m_vecOfArs` content, every item is followed by a newline

SYNOPSIS

```
/* ClassWithInitializerList.h */

template <typename T>
class ClassWithInitializerList final {
 public:
    template <typename... Args>
    ClassWithInitializerList(Args&&... args);
    ClassWithInitializerList(const std::initializer_list<T> lst);
    ClassWithInitializerList(const std::vector<T>& vec);

    void outputVector() const;

 private:
    std::vector<T> m_vecOfArs;
};

/* main.cpp */

int main() {
    ClassWithInitializerList<int> c1{1, 2, 3, 4, 5};
    ClassWithInitializerList<char> c2('a', 'b', 'c', 'd', 'e');

    std::vector<std::string> vec{"one", "two", "three"};
    ClassWithInitializerList<std::string> c3(vec);
```

```
    c1.outputVector();

    std::cout << std::endl;

    c2.outputVector();

    std::cout << std::endl;

    c3.outputVector();

    return 0;
}
```

CONSOLE OUTPUT

```
>./classWithInitializerList | cat -e
1$
2$
3$
4$
5$
$
a$
b$
c$
d$
e$
$
one$
two$
three$
>
```

SEE ALSO

std::initializer_list
Parameter pack

Act: Task 02

NAME

Universal Reference Determinant

DIRECTORY

`t02/`

BINARY

`universalReferenceDeterminant`

DESCRIPTION

Create a program that prints to the standard output information about various types of references.
`determineReference` member function is implemented to print:

- the type of the given argument
- the type of reference: `r-value` or `l-value`

Check output format in the CONSOLE OUTPUT.

Use `main.cpp` from the SYNOPSIS.

SYNOPSIS

```cpp
/* UniversalReferenceDeterminant.h */

namespace UniversalReferenceDeterminant {

template <typename T>
void determineReference(T&& obj);

} // end namespace UniversalReferenceDeterminant

/* main.cpp */

int main() {
    UniversalReferenceDeterminant::determineReference(10);

    const auto intVal = 10;
    UniversalReferenceDeterminant::determineReference(intVal);

    UniversalReferenceDeterminant::determineReference('u');

    const auto charVal = 'u';
    UniversalReferenceDeterminant::determineReference(charVal);

    return 0;
}
```

CONSOLE OUTPUT

```
>./universalReferenceDeterminant | cat -e
int is r-value reference$
int is l-value reference$
char is r-value reference$
char is l-value reference$
>
```

SEE ALSO

typeid operator
Value category

Act: Task 03

NAME

Bind

DIRECTORY

`t03/`

BINARY

`bind`

DESCRIPTION

Create a program using `std::bind`:

- use `Bind.h` from the SYNOPSIS
- use `main.cpp` from the SYNOPSIS
- implement `std::bind` calls by replacing all `...` with valid source code in `Bind.h`. You are allowed to use only functions provided in `Bind.h` to make the program work correctly
- the output must be identical to the CONSOLE OUTPUT

SYNOPSIS

```
/* Bind.h */

namespace SpecializedFunctions {

namespace Math {

template <typename T>
auto pow2 = std::bind(...);

template <typename T>
T add(const T arg1, const T arg2) {
    return arg1 + arg2;
}

auto iDontWontToCalculate = std::bind(...);

}  // end namespace Math

namespace Output {

template <typename T>
void output3Arguments(const T& arg1, const T& arg2, const T& arg3) {
    std::cout << arg1 << " " << arg2 << " " << arg3 << std::endl;
}
```

```cpp
template <typename T>
void outputPrintWords(const T& arg1, const T& arg2, const T& arg3, const T& arg4) {
    std::cout << arg1 << " " << arg2 << " " << arg3 << " " << arg4 << std::endl;
}

template <typename T>
auto outputWeird3Arguments = std::bind(...);

auto outputFusRoDah = std::bind(...);

auto outputLovelyWords = std::bind(...);

}  // end namespace Output
}  // end namespace SpecializedFunctions

/* main.cpp */

#include "Bind.h"

int main() {
    std::cout << "===========" << std::endl;

    std::cout << SpecializedFunctions::Math::pow2<int>(10) << std::endl;
    std::cout << SpecializedFunctions::Math::pow2<double>(10.01) << std::endl;

    std::cout << "===========" << std::endl;

    std::cout << SpecializedFunctions::Math::iDontWontToCalculate() << std::endl;

    std::cout << "===========" << std::endl;

    SpecializedFunctions::Output::outputWeird3Arguments<std::string>(
        "Hey",
        "it's not",
        "right"
    );
    SpecializedFunctions::Output::outputWeird3Arguments<std::string>(
        "right",
        "This",
        "is"
    );

    SpecializedFunctions::Output::outputFusRoDah();

    SpecializedFunctions::Output::outputLovelyWords("Are", "you", "kidding", "?");

    std::cout << "===========" << std::endl;

    return 0;
}
```

CONSOLE OUTPUT

```
> ./bind | cat -e
==========$
100$
100.2$
==========$
4$
==========$
it's not right Hey$
This is right$
Fus Ro Dah$
I love you !$
==========$
>
```

SEE ALSO

std::bind

Act: Task 04

NAME

Serializer

DIRECTORY

`t04/`

BINARY

`serializer`

DESCRIPTION

Create a program that serializes and deserializes fundamental types and simple classes using `reinterpret_cast`:

- `Serializer` class serializes and deserializes fundamental types and simple classes
- `serialize` member function creates file with a given file name and writes object (the way it is written in the memory) to it
- `deserialize` member function reads the file with the given file name and returns a specified object. Use ONLY these names of file `int`, `double` or `anyFileName`
- `SomeClass` that is given below
- `output` member function prints all member variables to the standard output after deserializing to check if data are correct

The program usage:

- `./serializer [anyFileName] [intVal] [charVal] [floatVal]` where `anyFileName` is any file name where `SomeClass` object with `intVal, charVal, floatVal` values are serialized
- `./serializer [int] [intVal]` where `int` is fundamental type indicating that `intVal` is serialized as `int`
- `./serializer [float] [floatVal]` where `float` is fundamental type indicating that `floatVal` is serialized as `float`
- `./serializer [anyFileName]` where `anyFileName` can be `int` or `float` to deserialize fundamental types, or can be any file name to deserialize `SomeClass`

Error handling. The program prints to the standard error:

- if there are not two, three or four arguments - `usage: ./serializer [arg1] [arg2] [arg3] [arg4]`
- if the value is an invalid data type or an invalid filename - `error`

See the CONSOLE OUTPUT.
This is not a real world example, but a simple case to learn where `reinterpret_cast` might be used.

Do not serialize objects like this at home.
And once more, this won't work with complex classes, so there is no need to handle all the cases.

SYNOPSIS

```
/* Serializer.h */
class Serializer {
 public:
    template<class T>
    void serialize(const T* object, const std::string& fileName) {
        ...
    }

    template<class T>
    T deserialize(const std::string& fileName) {
        ...
    }
};

/* SomeClass.h */
class SomeClass {
 public:
    SomeClass() = default;
    SomeClass(int intVal, char charVal, float floatVal);

    void output() const;

 private:
    int m_intArg;
    char m_charArg;
    float m_floatArg;
};
```

CONSOLE OUTPUT

```
>./serializer | cat -e
usage: ./serializer [arg1] [arg2] [arg3] [arg4]
>./serializer class 10 @ 3.14159 | cat -e
SomeClass: 10 @ 3.14159 serialized$
>./serializer class | cat -e
Deserialized SomeClass: 10 @ 3.14159$
>./serializer class 10 @ yo | cat -e
error
>./serializer int 10 | cat -e
int: 10 serialized$
>./serializer int | cat -e
Deserialized int: 10$
>./serializer float 3.14 | cat -e
float: 3.14 serialized$
>./serializer float | cat -e
deserialized float: 3.14$
```

```
>./serializer 13 12 | cat -e
error
>./serializer a b 2.1478 | cat -e
usage: ./serializer [arg1] [arg2] [arg3] [arg4]
>./serializer qwe | cat -e
error
>
```

SEE ALSO

Type conversions

Share

PUBLISHING

Last but not least, the final stage of your work is to publish it. This allows you to share your challenges, solutions, and reflections with local and global audiences. During this stage, you will discover ways of getting external evaluation and feedback on your work. As a result, you will get the most out of the challenge, and get a better understanding of both your achievements and missteps.

To share your work, you can create:

- a text post, as a summary of your reflection
- charts, infographics or other ways to visualize your information
- a video, either of your work, or a reflection video
- an audio podcast. Record a story about your experience
- a photo report with a small post

Helpful tools:

- Canva - a good way to visualize your data
- QuickTime - an easy way to capture your screen, record video or audio

Examples of ways to share your experience:

- Facebook - create and share a post that will inspire your friends
- YouTube - upload an exciting video
- GitHub - share and describe your solution
- Telegraph - create a post that you can easily share on Telegram
- Instagram - share photos and stories from ucode. Don't forget to tag us :)

Share what you've learned and accomplished with your local community and the world. Use #ucode and #CBLWorld on social media.