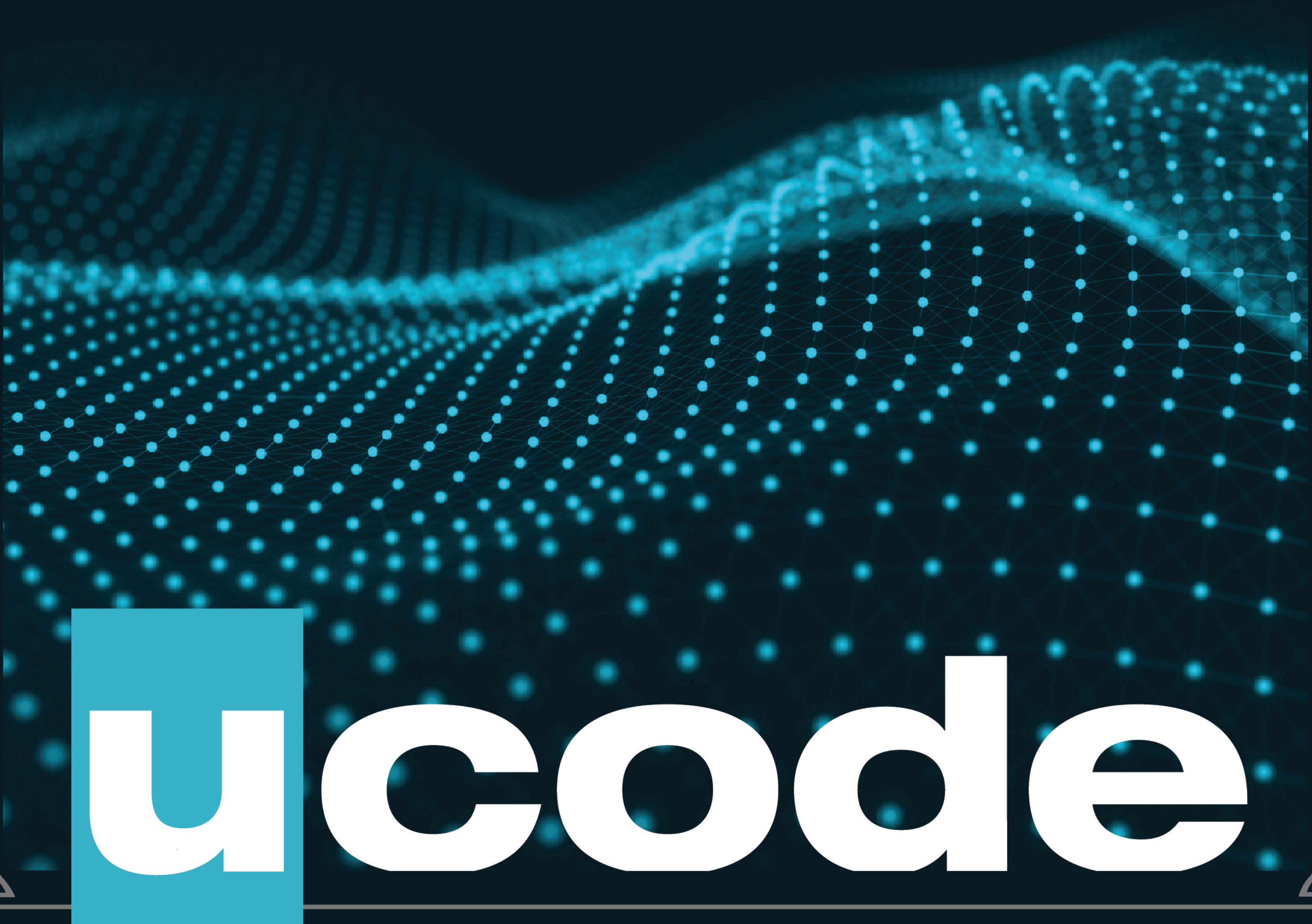
Sprint 04

Half Marathon C++

August 18, 2020



Contents

Engage

DESCRIPTION

- Hello, my friend! Do you want some Skooma?
- No? Well, okay.
- Let me offer you something more valuable.
- Do not forget to bring a bag of gold coins.

Simplifying a complex process is very rewarding.

Have you ever been developing a very complex and incomprehensible project?
When it takes a long time to search for a function or it is hard to understand a teammate's code because it's confusing and complicated.

When the project is too complicated, it's likely because its architecture and code are poor and not clear. One proven method of reducing complexity is to organize the program into modular parts.

A modular architecture makes development and maintenance of a program easier and more efficient. Remember, a program with a clear architecture is easier to expand and modify, as well as to test and debug.

To write a clean and understandable program, we advise you to first learn how to structure your project, work with folders and files.

Clear project architecture is impossible if you do not know how to write simple and modular code.

So, today you will find ways to make your architecture and code simple and understandable for everyone!

This is an important step towards success!

BIG IDEA

Software architecture.

ESSENTIAL QUESTION

How to reduce the complexity of software development?

CHALLENGE

Learning modular programming.

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.

- What is `modular programming`?
- Why does modular programming increase programmer productivity?
- Why keeping all files in one folder is a bad decision?
- Why is it worth creating more than one `.h` file?
- How can a clear, simple and understandable project structure help in development?
- How can C++ help you to divide the program into modules?
- How do `namespaces` help to divide the program into modules?
- What are classes and objects? What are your associations with it?
- What are class methods and attributes?
- Why do classes need constructors and destructors?
- What is the difference between a `class` and a `struct`?
- What is `encapsulation`?
- What are C++ access modifiers? How do they reduce quantity problems?
- What are getters and setters?
- Why is it not a good idea to do everything with `public` modifiers?
- What are the advantages of the `private` modifier?
- What is an `Exception`, and how to catch them all?
- What are the features of an `enum` in C++?
- What are `l-value` and `r-value` references?
- What are the use cases of `lookup tables`?
- What ideas do you have on how to make the architecture of the program understandable?

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.
- Use your research to carry out the tasks below.
- Discuss the tasks with students.

- Clone your git repository that is issued on the challenge page in the LMS.
- Try to implement your thoughts in code.
- Push the solution 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
- The solution will be checked and graded by students like you. Peer-to-Peer learning.
- Starting from this challenge, you must use the new project structure consisted of logical parts: the main app and own libraries. Each individual unit must know only about its resources and, if necessary, link other libraries to itself.

```
>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

Utils Library

DIRECTORY

`t00/`

SUBMIT

`utils/algorithmUtils.h, utils/CMakeLists.txt`

LEGEND

The Library tower is sure to please those who value literature. With space for all the bookshelves you could desire, you'll be able to enjoy your collection of books in style. Or perhaps take a book to the tower roof and enjoy the view of your steading?

DESCRIPTION

Create a static library `utils` for your future apps. Real-life projects are divided into components/modules. Reusing code is cool. This library will simplify some standard algorithms. Implement all the functions listed in the SYNOPSIS in the `utils/algorithmUtils.h`.

Your next projects must build and link the library only with few new lines of code in the `CMakeLists.txt` files.

- Add `add_subdirectory("utils")` into `./CMakeLists.txt` file in the project's root to build the library
- Add `target_link_libraries(<app_name> utils)` into `./app/CMakeLists.txt` to link the library

SYNOPSIS

```
// [from, to]
template <typename T>
bool IsInRange(const T& val, const T& from, const T& to);

template <typename T>
bool IsInRange(const T& val, const std::pair<const T&, const T&>& minmax);

// (from, to)
template <typename T>
bool IsInsideRange(const T& val, const T& from, const T& to);

template <typename T>
bool IsInsideRange(const T& val, const std::pair<const T&, const T&>& minmax);

// change the object if necessary
// and return the corresponding value of the operation success
template <class T, class U = T>
```

```
bool Modify(T& obj, U&& new_value);

template <class Collection, class T>
void RemoveAll(Collection& c, const T& value);

template <class Collection, class Pred>
void RemoveAllIf(Collection& c, Pred&& predicate);

template <class Collection, class T>
auto Find(Collection& c, const T& value);

template <class Collection, class Pred>
auto FindIf(Collection& c, Pred&& predicate);

template <class Collection, class T>
bool Contains(const Collection& c, const T& value);

template <class Collection, class Pred>
bool ContainsIf(const Collection& c, Pred&& predicate);

template <class Collection, class Pred>
int CountIf(const Collection& c, Pred&& predicate);

template <class T, class... Args>
const auto& Min(const T& arg, const Args&... args);

template <class T, class... Args>
const auto& Max(const T& arg, const Args&... args);

template <class Collection>
auto MaxElement(const Collection& c);

template <class Collection>
auto MinElement(const Collection& c);

template <class Collection, class Comp>
auto MaxElement(const Collection& c, Comp&& comparator);

template <class Collection, class Comp>
auto MinElement(const Collection& c, Comp&& comparator);

template <class Collection>
void Sort(Collection& c);

template <class Collection, class Comp>
void Sort(Collection& c, Comp&& comparator);

// remove all not unique elements in collection
template <class Collection>
void Unique(Collection& c);
```

```
template <class Collection, class Pred>
void Unique(Collection& c, Pred&& predicate);

template <class Collection, class Pred>
void ForEach(Collection& c, Pred&& predicate);

template <class Collection, class T>
int IndexOf(const Collection& c, const T& value);

template <class Collection, class Pred>
int IndexOfIf(const Collection& c, Pred&& predicate);
```

SEE ALSO

Value categories
<algorithm>

Act: Task 01

NAME

Imperial Vs Stormcloaks

DIRECTORY

`t01/`

SUBMIT

`CMakeLists.txt;`
`app/{main.cpp, CMakeLists.txt};`
`app/src/{ImperialSoldier.{cpp,h}, StormcloakSoldier.{cpp,h}, Sword.{cpp,h}, Axe.{cpp,h}}`

BINARY

`imperialVsStormcloak`

LEGEND

Classes are vocations, jobs or professions taken by persons. The protagonist of each game selects their class during character creation. Certain elements such as race impact class selection, due to the skill set defaulted by each class.
Each character class features a specialization, several major skills and an increase in attributes at level up.

DESCRIPTION

Create a program that simulates a battle between two soldiers. Implement four classes: `ImperialSoldier, StormcloakSoldier, Sword, Axe`. You can see prototypes of the classes in the SYNOPSIS.

- the `ImperialSoldier` has a `Sword` as the weapon
- the `StormcloakSoldier` has an `Axe`
- each soldier has `100` health points at creation
- weapon damage (in a range of `10-20` points) is initialized by values from command-line arguments at the weapon creation
- soldiers take turns dealing damage in the battle
- the ImperialSoldier attacks first
- the program prints out every step of simulation
- the program exits when the health of one of the soldiers drops to zero
- the pointer of the `<weapon class>` is stored inside the relevant `<soldier class>`
- `<soldier class>` cleans up the weapon and itself in `Destructor`

Error handling. The program prints to the standard error:

- in case of invalid number of arguments - `usage: ./imperialVsStormcloak [dmgOfSword] [dmgOfAxe]`
- if damage is not in the range of 10-20 - `Damage has to be in a range of 10-20 points.`

SYNOPSIS

```
/* <soldier class>.h */

class <soldier class> final {
public:
    <soldier class>();
    ~<soldier class>();

    void setWeapon(<weapon class>* sword);
    void attack(<enemy soldier class>& enemy);
    void consumeDamage(int amount);
    int getHealth() const;

private:
    <weapon class>* m_weapon;
    int m_health;
};

/* <weapon class>.h */

class <weapon class> final {
public:
    <weapon class>(int damage);

    int getDamage() const;

private:
    const int m_damage;
};
```

CONSOLE OUTPUT

```
>./imperialVsStormcloak | cat -e
usage: ./imperialVsStormcloak [dmgOfSword] [dmgOfAxe]
>./imperialVsStormcloak 10 3 | cat -e
Damage has to be in a range of 10-20 points.
>./imperialVsStormcloak 8 15 | cat -e
Damage has to be in a range of 10-20 points.
>./imperialVsStormcloak 20 18 | cat -e
Imperial soldier attacks and deals 20 damage$
Stormcloak soldier consumes 20 of damage$
Stormcloak soldier attacks and deals 18 damage$
Imperial soldier consumes 18 of damage$
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>$
Imperial soldier attacks and deals 20 damage$
Stormcloak soldier consumes 20 of damage$
Stormcloak soldier attacks and deals 18 damage$
Imperial soldier consumes 18 of damage$
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>$
Imperial soldier attacks and deals 20 damage$
Stormcloak soldier consumes 20 of damage$
Stormcloak soldier attacks and deals 18 damage$
```

```
Imperial soldier consumes 18 of damage$
<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>$
Imperial soldier attacks and deals 20 damage$
Stormcloak soldier consumes 20 of damage$
Stormcloak soldier attacks and deals 18 damage$
Imperial soldier consumes 18 of damage$
<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>$
Imperial soldier attacks and deals 20 damage$
Stormcloak soldier consumes 20 of damage$
Stormcloak soldier attacks and deals 18 damage$
Imperial soldier consumes 18 of damage$
<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>$
Imperial has won!$
>./imperialVsStormcloak 11 20 | cat -e
Imperial soldier attacks and deals 11 damage$
Stormcloak soldier consumes 11 of damage$
Stormcloak soldier attacks and deals 20 damage$
Imperial soldier consumes 20 of damage$
<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>$
Imperial soldier attacks and deals 11 damage$
Stormcloak soldier consumes 11 of damage$
Stormcloak soldier attacks and deals 20 damage$
Imperial soldier consumes 20 of damage$
<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>$
Imperial soldier attacks and deals 11 damage$
Stormcloak soldier consumes 11 of damage$
Stormcloak soldier attacks and deals 20 damage$
Imperial soldier consumes 20 of damage$
<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>$
Imperial soldier attacks and deals 11 damage$
Stormcloak soldier consumes 11 of damage$
Stormcloak soldier attacks and deals 20 damage$
Imperial soldier consumes 20 of damage$
<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>$
Imperial soldier attacks and deals 11 damage$
Stormcloak soldier consumes 11 of damage$
Stormcloak soldier attacks and deals 20 damage$
Imperial soldier consumes 20 of damage$
<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>$
Stormcloack has won!$
>
```

SEE ALSO

Classes
Constructor and destructor

Act: Task 02

NAME

Dragonborn

DIRECTORY

`t02/`

BINARY

`dragonborn`

LEGEND

When misrule takes its place at the eight corners of the world.
When the Brass Tower walks and Time is reshaped.
When the thrice-blessed fail and the Red Tower trembles.
When the Dragonborn Ruler loses his throne, and the White Tower falls.
When the Snow Tower lies sundered, kingless, bleeding.
The World-Eater wakes, and the Wheel turns upon the Last Dragonborn.

DESCRIPTION

Create a program that:

- uses a `Dragonborn` class. Its prototype listed in the SYNOPSIS
- takes one of three available actions `[shout, magic, weapon]` as a command-line argument
- converts a command to the appropriate `enum` value using a `lookup table` based on `std::map`
- calls the appropriate `Dragonborn` function using a `lookup table` based on `std::map`
- prints the respective message to the standard output. See in the CONSOLE OUTPUT for the exact output

It is forbidden to use `if` or `switch` statements with `enum class Actions`.

Starting from this task, follow the project structure in accordance with the ANALYSIS.

SYNOPSIS

```
class Dragonborn final {
public:
    enum class Actions {
        Shout,
        Magic,
        Weapon,
        Invalid
    };
    void executeAction(const Actions action);

private:
    void shoutThuum() const;
    void attackWithMagic() const;
    void attackWithWeapon() const;
};
```

CONSOLE OUTPUT

```
>./dragonborn | cat -e
usage: ./dragonborn [action]
>./dragonborn shout | cat -e
Yol Toor Shul$
>./dragonborn magic | cat -e
*attacks with magic spell*$
>./dragonborn weapon | cat -e
*attacks with weapon*$
>./dragonborn YOMAHAYOMASOUL | cat -e
Invalid action
>
```

SEE ALSO

Enumeration declaration

Act: Task 03

NAME

Dwemer Calculator

DIRECTORY

`t03/`

BINARY

`dwemerCalculator`

LEGEND

During existence, the Dwemer constructed many devices and machines. They were known to have created - and manufactured on a vast scale-thousands of mechanical apparatuses of varying complexity. Recreate the ancient `Dwemer Calculator`.

DESCRIPTION

Create a program that implements a simple integer calculator. The program:

- reads math expressions in realtime from standard input in format `operand1 operation operand2`
- prints `:>` before each expression
- handles 4 math operations between two operands:
  - `+` for addition operation
  - `-` for subtraction operation
  - `*` for multiplication operation
  - `/` for division operation
- has a simple cache feature in format `operand1 operation operand2 = variable`:
  - the program stores the result of the expression to a variable after the `=` character
  - the name of the variable is a non-zero length alphabetic string
  - the variable can be used in mentioned expressions as an operand
- prints the result of the expression on a new line
- catches exceptions while reading from standard input. Prints only 1 error message according to the priority listed below:
  - in case of division by zero - `division by zero`
  - in case of invalid input format - `invalid input`
  - in case of operand1 is out of integer range - `operand1 out of range`
  - in case of invalid operand1 - `invalid operand1`
  - in case of invalid operand2 - `invalid operand2`

  - in case of operand2 is out of integer range - `operand2 out of range`
- quits if the expression is `quit`

CONSOLE OUTPUT

```
>./dwemerCalculator
:>2+2
4
:>2+10=t
12
:>t - 1 = e
11
:>t     *     e =s
132
:>s/0
division by zero
:>z -  s
invalid operand1
:>s / Y0
invalid operand2
:>t e s
invalid input
:>2 +
invalid input
:>2 + 2 =
invalid input
:>2 + 2 = 4
invalid input
:>t * -2
-24
:>t+-2
10
:>t--2
14
:>321039721930217380 * 2
operand1 is out of range
:>123*9876543210
operand2 is out of range
:>U : 10
invalid operation
:>quit
>
```

SEE ALSO

Exceptions

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.