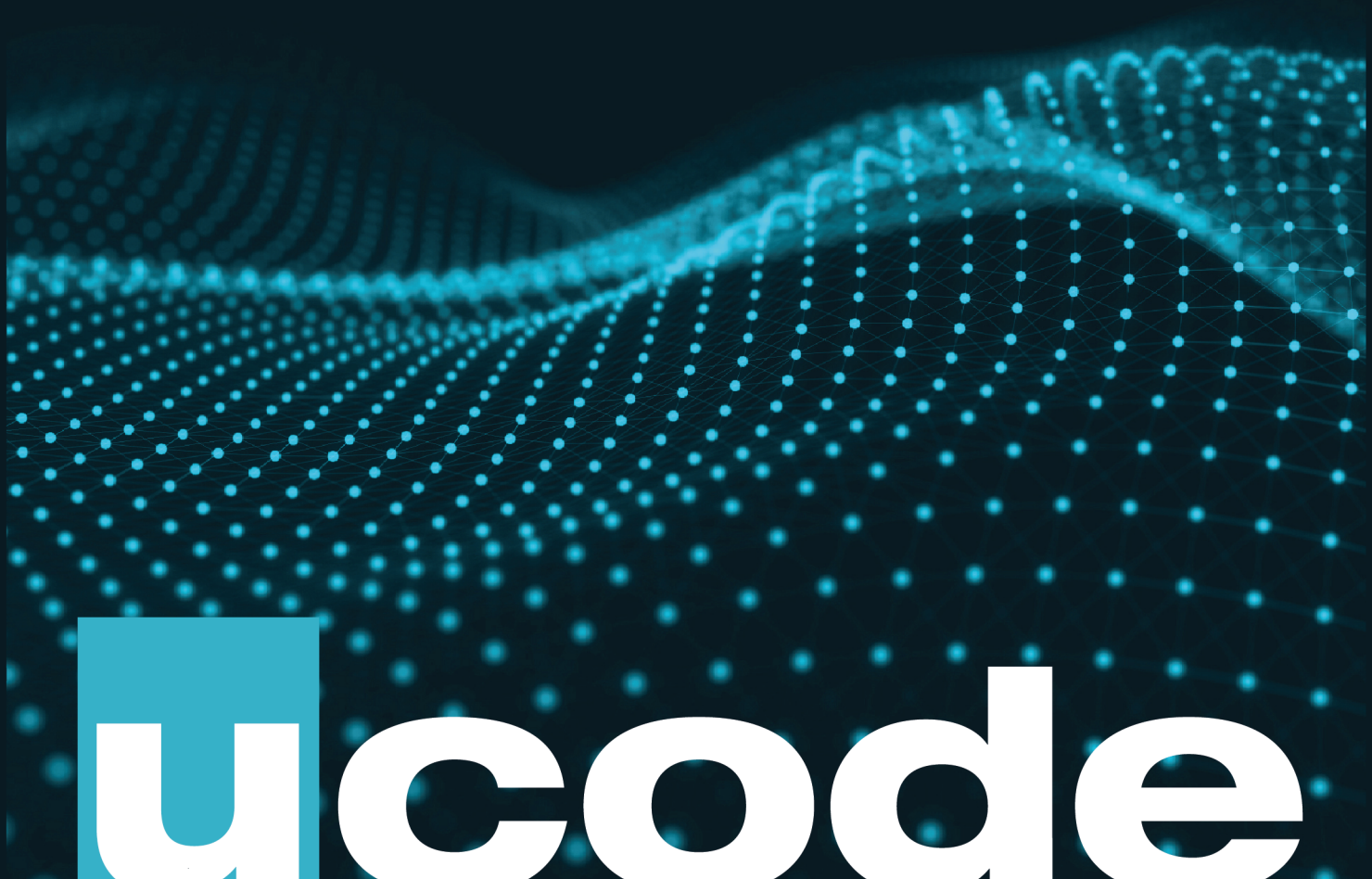
Sprint 05

Half Marathon C++

August 20, 2020



ucode

Contents

ucode

Engage

DESCRIPTION

Hail, Companion!

In order to develop a proper OOP program, it's important to understand the main OOP concepts and how they work together. The object-oriented paradigm is a completely different way of thinking in software development.

OOP was invented as an attempt to project real-world objects into the program code. It was thought that objects are easier to perceive and read for the developer because the world is easier to perceive as a set of interacting objects. For the same reason, it is possible to design a sufficiently clear software architecture. It will be easier for people to understand, expand, modify and also test such a project. It will save a lot of time in the future.

As to anything, there are some downsides to OOP. It does take a long time to design the architecture of a good OOP program. OOP architecture does not fit all projects, it depends on scale, desired functionality, etc. So, it's important to understand the advantages and disadvantages of OOP before using it in your project.

In this Sprint you will learn how OOP is achieved and what tools you have available. You'll find out how to work with C++ classes, create and delete them, inherit characteristics from each other, adapt to different data types, etc.

BIG IDEA

Object-oriented programming.

ESSENTIAL QUESTION

What are the benefits of using OOP in software architecture?

CHALLENGE

Learn the main features of OOP.

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 problems does OOP solve?
- What are the main features of OOP?
- How does OOP reduce the complexity of software architecture?
- What is `inheritance`?
- What is the difference between a parent class and a derived class?
- What is `composition`?
- What is the difference between inheritance and composition?
- In what cases is it better to use inheritance?
- What is multiple inheritance?
- What issues can arise when using multiple inheritance?
- What is `polymorphism`?
- How does polymorphism affect software architecture?
- What are constructors and destructors?
- What is a copy constructor?
- What is a delegating constructor?
- What is `encapsulation`?
- What are the access modifiers in OOP?
- What is the `member initializer list` and what is it used for?

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
  - each line of text in your code must be at most 120 characters long
- 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

Draugr v1

DIRECTORY

`t00/`

BINARY

`draugr`

LEGEND

Draugr are undead Nordic warriors of Skyrim. It is believed that draugr once served the Dragon Priests. All draugr are immune to poison and bleeding damage, with the additional attribute of possessing 50% resistance to frost.

DESCRIPTION

Create a program that:

- implements `Draugr` class which can be found in the SYNOPSIS
- uses `initializer list` to initialize data members of the class
- always initializes class members as follows: `m_health=100`, `m_frostResist=50`
- calls `shoutPhrase` to print phrase to the standard output that corresponds to the number from command-line argument. Find the list of phrases below

The list of phrases:

- `Qiilaan Us Dilon!`
- `Bolog Aaz, Mal Lir!`
- `Kren Sosaal!`
- `Dir Volaan!`
- `Aar Vin Ok!`
- `Unslaad Krosis!`
- `Faaz! Paak! Dinok!`
- `Aav Dilon!`
- `Sovngarde Saraan!`

Error handling. The program prints the corresponding messages to the standart error:

- if there are an invalid number of arguments - `usage: ./draugr [shoutNumber]`
- if there is invalid `shoutNumber` in the argument - `Invalid shoutNumber`

SYNOPSIS

```
class Draugr {
public:
    Draugr();

    void shoutPhrase(int shoutNumber) const;

private:
    double m_health;
    const int m_frostResist;
};
```

CONSOLE OUTPUT

```
>./draugr | cat -e
usage: ./draugr [shoutNumber]
>./draugr 0 | cat -e
Draugr (100 health, 50% frost resist) shouts:$
Qiilaan Us Dilon!$
>./draugr 4 | cat -e
Draugr (100 health, 50% frost resist) shouts:$
Aar Vin Ok!$
>./draugr 8 | cat -e
Draugr (100 health, 50% frost resist) shouts:$
Sovngarde Saraan!$
>./draugr 9 | cat -e
Invalid shoutNumber
>
```

SEE ALSO

Constructors and member initializer lists

Act: Task 01

NAME

Draugr v2

DIRECTORY

`t01/`

BINARY

`draugr`

DESCRIPTION

Let's extend functionality of the program in the TASK 00.

Improve the `Draugr` class from the previous task by adding two more constructors.
The constructors below must not allow implicit conversions of objects created using them.

Second constructor:

- takes one integer parameter called `frostResist` and initializes `m_frostResist` with the given value
- always sets `m_health` to a default value

Third constructor:

- takes a double parameter called `health` and initializes `m_health` with the given value
- takes an integer parameter called `frostResist` and initializes `m_frostResist` with the value if it is given, either sets to the default value, thus you can use this constructor like `two different constructors`
- prints `Draugr with N health and N% frost resist was born!` to the standard output

Default values are listed in TASK 00.

The program prints `Draugr with N health and N% frost resist was born!` on every Draugr creation but this message is printed `only in the third constructor`.

Now the program has to handle more arguments:

- in case of one argument - the program creates Draugr and makes him shout with the first constructor
- in case of two arguments - the program creates Draugr with the second constructor
- in case of three arguments - the program creates Draugr with the third constructor

Error handling. The program prints the corresponding messages to the standart error:

- if the command line argument number is out of range - `usage: ./draugr [shoutNumber] [health] [frostResist]`
- if there is invalid value of `shoutNumber` - `Invalid shoutNumber`

- if the program throws other exceptions - `Error`

CONSOLE OUTPUT

```
>./draugr | cat -e
usage: ./draugr [shoutNumber] [health] [frostResist]
>./draugr 1 22 22 22 | cat -e
usage: ./draugr [shoutNumber] [health] [frostResist]
>./draugr 7 | cat -e
Draugr with 100 health and 50% frost resist was born!$
Draugr (100 health, 50% frost resist) shouts:$
Aav Dilon!$
>./draugr 7 66 | cat -e
Draugr with 66 health and 50% frost resist was born!$
Draugr (66 health, 50% frost resist) shouts:$
Aav Dilon!$
Draugr with 100 health and 66% frost resist was born!$
Draugr (100 health, 66% frost resist) shouts:$
Aav Dilon!$
>./draugr 7 66 99 | cat -e
Draugr with 66 health and 99% frost resist was born!$
Draugr (66 health, 99% frost resist) shouts:$
Aav Dilon!$
>./draugr 10 66 99 | cat -e
Invalid shoutNumber
>./draugr 1 22 9876543210 | cat -e
Error
>
```

SEE ALSO

Constructors and member initializer lists

Act: Task 02

NAME

Draugr v3

DIRECTORY

`t02/`

BINARY

`draugr`

DESCRIPTION

Go even further and add some modifications into your program.

As for the `Draugr` class - you need to add:

- a string member variable called `m_name`
- a void method `setName` that has a constant `rvalue` reference called `name` as the only parameter and set the appropriate value of the parameter to the `m_name`
- a copy constructor that prints the message `Copy constructor was called` on call
- a move constructor that prints the message `Move constructor was called` on call
- the output of the name in Draugr shout message

The program has the next improvements:

- it works only if there are 3 or 4 command-line arguments
- the second argument contains two names delimited by `comma(,)`
- if only the `health` is initialized though the command-line argument - the program calls the second constructor and sets the `name1` to the created Draugr. Then, calls the copy constructor and sets `name2` to the newly created Draugr. Every Draugr shouts after creation
- if `frostResist` initialized though the command-line argument - the program calls the third constructor and sets the `name1` to the created Draugr. Then, calls the move constructor and sets `name2` to the newly created Draugr. Every Draugr shouts after creation

Error handling. The program prints the corresponding messages to the standart error:

- if the command line argument number is out of range - `usage: ./draugr [shoutNumber] [name1,name2] [health] [frostResist]`
- if there is invalid value of `shoutNumber` - `Invalid shoutNumber`
- if there is invalid name formatting in the second argument - `Invalid names`
- if the program throws other exceptions - `Error`

SYNOPSIS

```
...

        Draugr(Draugr& other);
        Draugr(Draugr&& other);

        void setName(const std::string&& name);

private:
        std::string m_name;

...
```

CONSOLE OUTPUT

```
>./draugr | cat -e
usage: ./draugr [shoutNumber] [name1,name2] [health] [frostResist]
>./draugr 1 "Lord,Overlord" 45 | cat -e
Draugr with 45 health and 50% frost resist was born$
Draugr Lord (45 health, 50% frost resist) shouts:$
Bolog Aaz, Mal Lir!$
Draugr with 45 health and 50% frost resist was born$
Copy constructor was called$
Draugr Overlord (45 health, 50% frost resist) shouts:$
Bolog Aaz, Mal Lir!$
>./draugr 1 "Lord,Overlord" 45 59 | cat -e
Draugr with 45 health and 59% frost resist was born$
Draugr Lord (45 health, 59% frost resist) shouts:$
Bolog Aaz, Mal Lir!$
Draugr with 45 health and 59% frost resist was born$
Move constructor was called$
Draugr Overlord (45 health, 59% frost resist) shouts:$
Bolog Aaz, Mal Lir!$
>./draugr 1 "Lord," 45 59 | cat -e
Invalid names
>./draugr 1 "Lord,Overlord" 45 9876543210 | cat -e
Error
>
```

SEE ALSO

Constructors and member initializer lists

Act: Task 03

NAME

Draugr v4

DIRECTORY

`t03/`

BINARY

`draugr`

DESCRIPTION

And at last, you need to change the `draugr` program in the following way:

- change the `Draugr` class from the previous task, so that it's not possible to copy or move `Draugr` objects
- use `main.cpp` from the SYNOPSIS in the program. The output of the program with this `main.cpp` must be exactly like in the CONSOLE OUTPUT

If you uncomment commented lines and try to compile the program, you will see the compilation error messages that refer to `Draugr` constructor declaration and class itself. It means that you've done all right.

SYNOPSIS

```
#include "Draugr.h"

int main() {
        Draugr d1;
        d1.setName("Death Overlord");
        d1.shoutPhrase(3);
//        Draugr d2 = d1;
        Draugr d3(10.0, 30);
        d3.setName("Thrall");
        d3.shoutPhrase(4);
//        Draugr d4 = std::move(d3);
        return 0;
}
```

CONSOLE OUTPUT

```
>./draugr | cat -e
Draugr with 100 health and 50% frost resist was born$
Draugr Death Overlord (100 health, 50% frost resist) shouts:$
Dir Volaan!$
Draugr with 10 health and 30% frost resist was born$
Draugr Thrall (10 health, 30% frost resist) shouts:$
Aar Vin Ok!$
>
```

Act: Task 04

NAME

Imperial Vs. Stormcloak

DIRECTORY

`t04/`

BINARY

`imperialVsStormcloak`

DESCRIPTION

Let's reimplement Sprint 04 Task 01 using inheritance.

Modify previous solution:

- create a base class `Soldier` for the `ImperialSoldier` and `StormcloakSoldier` classes
- create a base class `Weapon` for the `Axe` and `Sword` classes
- the ImperialSoldier's name is `Martin`
- the StormcloakSoldier's name is `Flynn`
- each soldier has `100` health points at creation
- the damage of the weapon has to be in the range of 10-20 points
- the soldiers take turns dealing damage to each other. The StormcloakSoldier attacks first
- the program prints out every step of the simulation. As well as on creation and deletion of each soldier
- the program exits when the health of one of the soldiers drops to zero

Error handling. The program prints the corresponding messages to the standart error:

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

Look at the CONSOLE OUTPUT and make sure that the program works exactly as it must.

SYNOPSIS

```
/* Soldier.h */

class Soldier {
public:
    Soldier(std::string&& name, int health);
    virtual ~Soldier();

    void attack(Soldier& other);
    void setWeapon(Weapon* weapon);
```

```cpp
    int getHealth() const;

private:
    Weapon* m_weapon;
    std::string m_name;
    int m_health;
};

/* Weapon.h */

class Weapon {
public:
    explicit Weapon(int damage);
    virtual ~Weapon() = default;

    int getDamage() const;

private:
    const int m_damage;
};
```

CONSOLE OUTPUT

```
>./imperialVsStormcloak  | cat -e
usage: ./imperialVsStormcloak [dmgOfSword] [dmgOfAxe]
>./imperialVsStormcloak 12 4 | cat -e
Damage has to be in a range of 10-20 points.
>./imperialVsStormcloak 3 17 | cat -e
Damage has to be in a range of 10-20 points.
>./imperialVsStormcloak 15 18 | cat -e
Soldier Martin was created$
Imperial soldier was created$
Soldier Flynn was created$
Stormcloak soldier was created$
$
***The battle has begun!***$
$
Flynn attacks Martin and deals 18 damage$
Martin attacks Flynn and deals 15 damage$
<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>$
Flynn attacks Martin and deals 18 damage$
Martin attacks Flynn and deals 15 damage$
<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>$
Flynn attacks Martin and deals 18 damage$
Martin attacks Flynn and deals 15 damage$
<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>$
Flynn attacks Martin and deals 18 damage$
Martin attacks Flynn and deals 15 damage$
<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>$
Flynn attacks Martin and deals 18 damage$
Martin attacks Flynn and deals 15 damage$
<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>$
Flynn attacks Martin and deals 18 damage$
```

```
Martin attacks Flynn and deals 15 damage$
<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>$
Stormcloack has won!$
$
Stormcloak soldier was deleted$
Soldier Flynn was deleted$
Imperial soldier was deleted$
Soldier Martin was deleted$
>./imperialVsStormcloak 20 10 | cat -e
Soldier Martin was created$
Imperial soldier was created$
Soldier Flynn was created$
Stormcloak soldier was created$
$
***The battle has begun!***$
$
Flynn attacks Martin and deals 10 damage$
Martin attacks Flynn and deals 20 damage$
<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>$
Flynn attacks Martin and deals 10 damage$
Martin attacks Flynn and deals 20 damage$
<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>$
Flynn attacks Martin and deals 10 damage$
Martin attacks Flynn and deals 20 damage$
<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>$
Flynn attacks Martin and deals 10 damage$
Martin attacks Flynn and deals 20 damage$
<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>$
Flynn attacks Martin and deals 10 damage$
Martin attacks Flynn and deals 20 damage$
<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>$
Imperial has won!$
$
Stormcloak soldier was deleted$
Soldier Flynn was deleted$
Imperial soldier was deleted$
Soldier Martin was deleted$
>
```

SEE ALSO

Inheritance
Polymorphism
Derived class

Act: Task 05

NAME

Werewolf

DIRECTORY

`t05/`

BINARY

`werewolf`

LEGEND

A Werewolf is someone who has Lycanthropy and has the ability to turn into a wolf or wolf-like creature. Werewolves were originally created by Hircine. Werewolves are the most common type of Lycanthrope and can be found in nearly all areas of Tamriel.

DESCRIPTION

Implement the following in the program:

- create a base class `Creature`
- `Wolf` and `Human` classes are derived from `Creature` class
- `Werewolf` class is derived from both `Wolf` and `Human` classes
- use `main.cpp` from SYNOPSIS
- print `Creature was created` and `Creature was deleted` on `Creature` constructor/destructor calls
- print `Werewolf was created` and `Werewolf was deleted` on `Werewolf` constructor/destructor calls
- `Wolf` and `Human` classes have `default` constructors and destructors

Investigate in which way you can create only one instance of base class `Creature` while `Werewolf` is created.

SYNOPSIS

```
#include "Werewolf.h"

int main(int argc, char** argv) {
    Werewolf wolf;

    return 0;
}
```

CONSOLE OUTPUT

```
>./werewolf | cat -e
Creature was created$
Werewolf was created$
Werewolf was deleted$
Creature was deleted$
>
```

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.