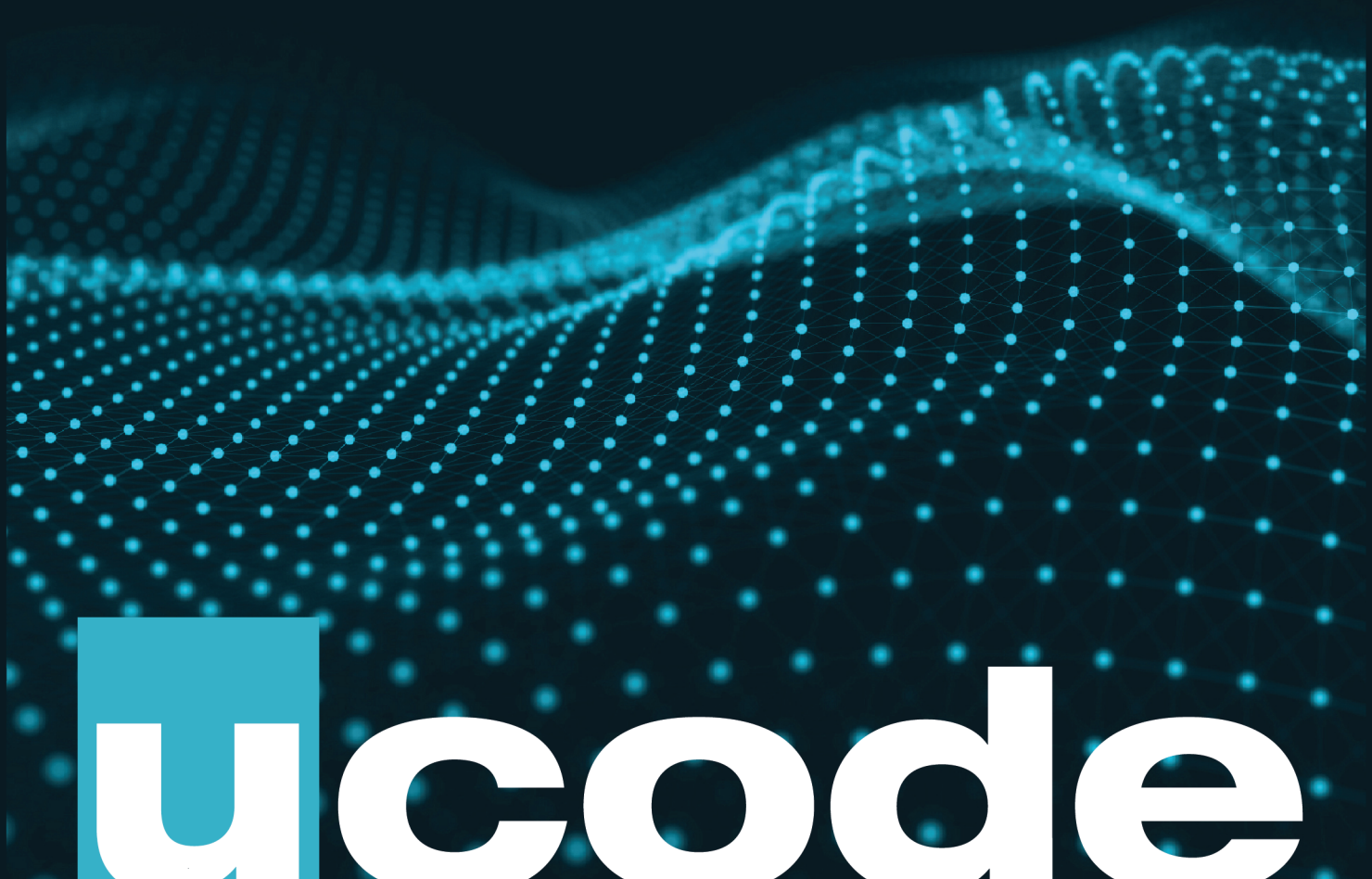
Sprint 07

Half Marathon C++

August 26, 2020



ucode

Contents

ucode

Engage

DESCRIPTION

You have a class,
You have a template,
Ugh! Class template!

Welcome, friend!
Your knowledge is becoming deeper and deeper every day. You've already written your classes and templates. Let's combine them into something useful!
Finding solutions, generating ideas and writing code independently are a few of the most important skills of a good programmer. Avoid copy-pasting someone's code from StackOverflow, GitHub, etc.

In this Sprint we offer you to develop two complex systems, based on your knowledge, for storing and managing data. It is a great chance to look under the hood of something you always use. So how about building these systems from scratch now?

BIG IDEA

Class templates.

ESSENTIAL QUESTION

How do the best tools for storing data in C++ work?

CHALLENGE

Recode standard containers.

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 can we use `class templates`?
- What is the difference between a `class template` and a `function template`?
- How can containers be classified?
- Why do we need so many containers?
- Which methods do you use in container classes?
- What methods are needed in container classes?
- What is the difference between an `std::string` and an `std::basic_string`?
- What are the pros and cons of using a `std::vector` compared to a simple array?
- What are the disadvantages of the standard `std::vector`?
- When should we not use `class templates`?

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.
- Open the story and read.
- Arrange to brainstorm tasks with other students.
- Clone your git repository that is issued on the challenge page in the LMS.
- Start to develop tasks. Test your code.

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

Vector

DIRECTORY

`t00/`

BINARY

`vector`

DESCRIPTION

Create your own simplified implementation of the `Vector` class:

- implement ONLY the features that are specified in the SYNOPSIS
- the class must be able to store objects of any type
- provide `main.cpp` with enough test coverage to prove that the class works correctly. The purpose of this task is to learn how to create quality tests. It must be written in a way that even an assessor without experience can understand it and check the correctness of your program with the minimum amount of time and effort. Bring your `main.cpp` when assessing your peers on this challenge

SYNOPSIS

```
namespace CBL {

template <class T>
class Vector {
 public:
    using iterator = T*;

    Vector();
    explicit Vector(size_t size);
    Vector(size_t size, const T& value);
    Vector(iterator first, iterator last);
    Vector(const std::initializer_list<T>& lst);
    Vector(const Vector<T>& other);
    ~Vector();

    Vector<T>& operator=(const Vector<T>& other);
    bool operator==(const Vector<T>& other) const;
    bool operator!=(const Vector<T>& other) const;
    bool operator<(const Vector<T>& other) const;
    bool operator>(const Vector<T>& other) const;
    bool operator<=(const Vector<T>& other) const;
    bool operator>=(const Vector<T>& other) const;

    // iterators
    iterator begin() const;
```

```
    iterator end() const;

    // capacity
    size_t size() const;
    size_t capacity() const;
    bool isEmpty() const;
    void resize(size_t size, const T& value = T());
    void reserve(size_t size);

    // element access
    T& operator[](size_t index) const;
    T& at(size_t index) const;
    T* data() const;

    // modifiers
    void push_back(const T& value);
    void pop_back();
    iterator insert(iterator pos, const T& value);
    iterator erase(iterator pos);
    iterator erase(iterator first, iterator last);
    void clear();

 private:
    size_t m_size{0};
    size_t m_capacity{0};
    T* m_buffer{nullptr};
};

} // end namespace CBL
```

SEE ALSO

std::vector

Act: Task 01

NAME

Basic String

DIRECTORY

`t01/`

BINARY

`basicString`

DESCRIPTION

In this task you must:

- implement `BasicString` class
- create two specialized `String` and `WString` classes for `char` and `wchar_t` types respectively
- `String` and `WString` classes must have all the features of `BasicString` and a little bit more. See the SYNOPSIS for details
- provide `main.cpp` with enough test coverage to prove that class works correctly. The purpose of this task is to learn how to create quality tests. It must be written in a way that even an assessor without experience can understand it and check the correctness of your program with the minimum amount of time and effort. Bring your `main.cpp` when assessing your peers on this challenge

SYNOPSIS

```
namespace CBL {

template <class T>
class BasicString {
 public:
    using iterator = T*;
    static const size_t npos = -1ul;

    BasicString();
    BasicString(const BasicString<T>& str);
    BasicString(const BasicString<T>& str, size_t pos, size_t len = npos);
    explicit BasicString(const T* str);
    BasicString(const T* str, size_t n);
    BasicString(size_t n, T c);
    BasicString(iterator first, iterator last);
    virtual ~BasicString();

    // iterators
    iterator begin() const;
    iterator end() const;
```

```
    // capacity
    size_t length() const;

    // element access
    T& operator[](size_t index) const;
    T& at(size_t index) const;
    T& back() const;
    T& front() const;

    // modifiers
    BasicString<T>& operator=(const BasicString<T>& str);
    BasicString<T>& operator=(const T* str);
    BasicString<T>& operator=(const T c);

    BasicString<T>& operator+=(const BasicString<T>& str);
    BasicString<T>& operator+=(const T* str);
    BasicString<T>& operator+=(const T c);

    void append(const BasicString<T>& str);
    void append(const T* str);
    void append(const T* str, size_t n);
    void append(size_t n, T c);
    void append(iterator first, iterator last);

    void swap(BasicString<T>& str);

    // string operations
    const T* c_str() const;

    virtual int compare(const BasicString<T>& str) const;
    virtual int compare(const T* str) const;

 protected:
    size_t m_size{0};
    T* m_buffer{nullptr};
};

} // end namespace CBL

template <class T>
CBL::BasicString<T> operator+(
    const CBL::BasicString<T>& lhs,
    const CBL::BasicString<T>& rhs
);

template <class T>
CBL::BasicString<T> operator+(const T* lhs, const CBL::BasicString<T>& rhs);

template <class T>
CBL::BasicString<T> operator+(const CBL::BasicString<T>& lhs, const T* rhs);
```

```cpp
template <class T>
CBL::BasicString<T> operator+(const T lhs, const CBL::BasicString<T>& rhs);

template <class T>
CBL::BasicString<T> operator+(const CBL::BasicString<T>& lhs, const T rhs);

template <class T>
bool operator==(const CBL::BasicString<T>& lhs, const CBL::BasicString<T>& rhs);

template <class T>
bool operator==(const T* lhs, const CBL::BasicString<T>& rhs);

template <class T>
bool operator==(const CBL::BasicString<T>& lhs, const T* rhs);

template <class T>
bool operator!=(const CBL::BasicString<T>& lhs, const CBL::BasicString<T>& rhs);

template <class T>
bool operator!=(const T* lhs, const CBL::BasicString<T>& rhs);

template <class T>
bool operator!=(const CBL::BasicString<T>& lhs, const T* rhs);

template <class T>
bool operator<(const CBL::BasicString<T>& lhs, const CBL::BasicString<T>& rhs);

template <class T>
bool operator<(const T* lhs, const CBL::BasicString<T>& rhs);

template <class T>
bool operator<(const CBL::BasicString<T>& lhs, const T* rhs);

template <class T>
bool operator<=(const CBL::BasicString<T>& lhs, const CBL::BasicString<T>& rhs);

template <class T>
bool operator<=(const T* lhs, const CBL::BasicString<T>& rhs);

template <class T>
bool operator<=(const CBL::BasicString<T>& lhs, const T* rhs);

template <class T>
bool operator>(const CBL::BasicString<T>& lhs, const CBL::BasicString<T>& rhs);

template <class T>
bool operator>(const T* lhs, const CBL::BasicString<T>& rhs);

template <class T>
bool operator>(const CBL::BasicString<T>& lhs, const T* rhs);
```

```
template <class T>
bool operator>=(const CBL::BasicString<T>& lhs, const CBL::BasicString<T>& rhs);

template <class T>
bool operator>=(const T* lhs, const CBL::BasicString<T>& rhs);

template <class T>
bool operator>=(const CBL::BasicString<T>& lhs, const T* rhs);

/* Derived class additional member functions */

size_t find(const <string class>& str, size_t pos = 0) const;
size_t find(const <char type>* str, size_t pos = 0) const;
size_t find(<char type> c, size_t pos = 0) const;

size_t rfind(const <string class>& str, size_t pos = 0) const;
size_t rfind(const <char type>* str, size_t pos = 0) const;
size_t rfind(<char type> c, size_t pos = 0) const;
```

See also

std::basic_string

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.