Sprint 01

Half Marathon C++

August 4, 2020

ucode

Contents

ucode

Engage

DESCRIPTION

Hello!

Hope you enjoyed the introduction to the world of C++. C++ is a great way to implement your ideas into reality. Generic programming simplifies a lot of processes in programming. It enables the programmer to write a general algorithm which will work with all data types using C++ class templates and functions. Generics can be used for any style of programming, making it relevant for future data types. Today you will learn a vital part of C++ - how to use STL.

The Standard Template Library provides common data structures and programming features, such as lists, stacks, arrays, etc. It is a library of container classes, algorithms and iterators for generic programming, which is a fundamental part in most programs.
STL helps you develop better, more functional and scalable programs. Let's get started.

BIG IDEA

Generic programming.

ESSENTIAL QUESTION

How to use generic algorithms and data structures in C++ effectively?

CHALLENGE

Learn STL.

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 `STL` in C++?
- What are the main components of STL?
- What are iterators?
- What generic containers exist in STL?
- What are the properties of sequence containers?
- What are the use cases of `<algorithm>`?
- What are the use cases of `std::deque`?
- What are the use cases of `std::array`?
- What are the use cases of `std::vector`?
- What is the main difference between `std::forward_list` and `std::list`, and in which cases can they be applied?

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.
- Try to use one of the sequence containers.
- 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.
- Try to implement your thoughts in 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)
- The solution will be checked and graded by students like you. Peer-to-Peer learning.
- 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

Simple Sort

DIRECTORY

`t00/`

SUBMIT

`main.cpp, src/*.[cpp,h], CMakeLists.txt`

BINARY

`simpleSort`

DESCRIPTION

Create a program that sorts command-line arguments in ASCII order. The program works only with exactly 5 arguments.
The program prints `usage: ./simpleSort arg1 arg2 arg3 arg4 arg5` to the standard error in case of invalid number of arguments.

Use `std::array` and `standard library function` for sorting.

CONSOLE OUTPUT

```
>./simpleSort Oblivion Skyrim Online Legends Arena | cat -e
Arena Legends Oblivion Online Skyrim$
>./simpleSort 5 4 3 2 1 | cat -e
1 2 3 4 5$
>./simpleSort tes | cat -e
usage: ./simpleSort arg1 arg2 arg3 arg4 arg5
>./simpleSort omae wa mou shindeiru nani | cat -e
mou nani omae shindeiru wa$
>
```

SEE ALSO

std::array
Standard Library

Act: Task 01

NAME

Move Along

DIRECTORY

`t01/`

SUBMIT

`main.cpp, src/*.[cpp,h], CMakeLists.txt`

BINARY

`moveAlong`

LEGEND

Today you have the honor of working with Ganciele Douar! Help him to make sure that thieves will not go through the post. Names of the thieves should be familiar to you.

DESCRIPTION

Create a program that prints `<name>, move along!` to the standard output if a command-line argument contains one of these names: `mercer, emer, jim` (case insensitive).

The program prints `usage: ./moveAlong [args]` to the standard error if there is no command-line arguments.

Use `std::vector`.

CONSOLE OUTPUT

```
>./moveAlong ohhiMErCER James jiMemer :o | cat -e
ohhiMErCER, move along!$
jiMemer, move along!$
>./moveAlong | cat -e
usage: ./moveAlong [args]
>
```

SEE ALSO

`std::vector`

Act: Task 02

NAME

Visit All

DIRECTORY

`t02/`

SUBMIT

`main.cpp, src/*.[cpp,h], CMakeLists.txt`

BINARY

`visitAll`

LEGEND

New quest, traveller! Stock up with stamina potions, the road won't be short.

DESCRIPTION

Create a program that calculates the best town to start the journey. The program prints the route of the journey to the standard output. See a detailed example in the CONSOLE OUTPUT.
Towns are arranged in series and form an enclosed circle, where the first town is next to the last one. Initial sequence of towns is given by order of command-line arguments.
Each town is defined by `name,stamina,distance` command-line argument where:

- `name` - the name of the town
- `stamina` - the amount of stamina potions available in the town
- `distance` - the distance from the current town to the next one

Assuming that with 1 stamina potion, a traveller can go 1 unit of distance.
If there are more than one `best town to start the journey`, the program prints the route for the one that goes first in queue.

Error handling. The program prints errors to the standart error:

- if there is no command-line arguments - `usage: ./visitAll [[name,stamina,distance] ...]`
- in case of invalid arguments - `Argument <argument> is not valid`
- if it is impossible to finish the journey with given stamina and distance - `Mission: Impossible`

Use `std::deque`.

CONSOLE OUTPUT

```
>./visitAll | cat -e
usage: ./visitAll [[name,stamina,distance] ...]
>./visitAll "Riften,4,6" "Morthal,6,5" "Dawnstar,7,5" "Markarth,4,5" | cat -e
```

```
1. Morthal$
2. Dawnstar$
3. Markarth$
0. Riften$
>./visitAll "London,0,2" "Kharkiv,1,3" | cat -e
Mission: Impossible
>./visitAll ",0,2" "Kharkiv,1,3"  | cat -e
Argument ,0,2 is not valid
>./visitAll "London,0z,2" "Kharkiv,1,3"  | cat -e
Argument London,0z,2 is not valid
>
```

SEE ALSO

std::deque

Act: Task 03

NAME

Vampire Rabbits

SUBMIT

`main.cpp, src/*.[cpp,h], CMakeLists.txt`

BINARY

`vampireRabbits`

LEGEND

This time, traveller, you will work on rabbit ranch.
But beaware of those vampire rabbits!

DESCRIPTION

Create a program that simulates a rabbit ranch:

- the program starts with 10 rabbits
- every rabbit is created with a random gender. 50% chance of male or female gender creation
- there is a 1% random chance of vampire rabbit creation
- each turn lasts for 1 second
- each turn the rabbits' age increments for 1 year
- vampire rabbits do not take part in reproduction
- each turn one male and one female rabbit produce one new rabbit (i.e. if there are 7 males and 3 females, they produce 3 new bunnies).
- if the vampire rabbit was born then each turn it will change exactly one non-vampire rabbit into a vampire
- after each turn the program prints to the standard output how many rabbits of each type/gender there are. See CONSOLE OUTPUT
- if the rabbit becomes older than 3 years old, it dies
- if a vampire rabbit becomes older than 8 years old, it dies
- when all the rabbits have died - the program terminates
- if the overall population exceeds 1000 - the program terminates

Use `std::list`.

SYNOPSIS

```
enum class Gender {
    Male,
    Female
};

struct Rabbit {
    Gender gender;
    bool isVampire;
    int age;
};
```

CONSOLE OUTPUT

```
>./vampireRabbits | cat -e
Males: 7$
Females: 3$
Vampires: 0$
$
Males: 9$
Females: 4$
Vampires: 0$
$
...    # hidden output of the program
$
Males: 357$
Females: 368$
Vampires: 26$
$
Males: 484$
Females: 483$
Vampires: 55$
>
```

SEE ALSO

std::list

Act: Task 04

NAME

Standard Algorithms V1

DIRECTORY

`t04/`

SUBMIT

`main.cpp, src/*.[cpp,h], CMakeLists.txt`

BINARY

`stdAlgoV1`

LEGEND

Most of the NPCs in Skyrim are named characters. Quest-givers, trainers, merchants, as well as most NPCs in cities and towns are named characters.

DESCRIPTION

Create a program that analyzes names. The program takes file as a command-line argument with the list of names, one per line.
Using `standard library algorithms` and `std::forward_list`, get and print to the standard output the following information:

- `size: <amount>` - number of names listed in the file
- `contains 'rich': <true || false>` - `true` if any of names contain substring 'rich' and `false` otherwise
- `none of lengths is 15: <true || false>` - `true` if none of names' length is equal to 15 and `false` otherwise
- `all end with 'vel': <true || false>` - `true` if all names end with 'vel' and `false` otherwise
- `not contains 'mel': <amount>` - number of names that don't contain the substring 'mel'

Error handling. The program prints errors to the standart error:

- in case of invalid number of arguments - `usage: ./stdAlgoV1 [file_name]`
- in case of any other errors (invalid file, invalid file formatting, etc.) - `error`

CONSOLE OUTPUT

```
>cat names.txt
Amelie
Anne
Anne-Corinne
Anne-Marie
Aveline
```

```
Enrichetta
Helen-Elizabeth
>./stdAlgoV1 names.txt | cat -e
size: 7$
contains 'rich': true$
none of lengths is 15: false$
all end with 'vel': false$
not contains 'mel': 6$
>./stdAlgoV1 names.txt yo | cat -e
usage: ./stdAlgoV1 [file_name]
>./stdAlgoV1 directory_name | cat -e
error
>
```

SEE ALSO

<algorithm>
std::forward_list

Act: Task 05

NAME

Standard Algorithms V2

DIRECTORY

`t05/`

SUBMIT

`main.cpp, src/*.[cpp,h], CMakeLists.txt`

BINARY

`stdAlgoV2`

LEGEND

"Names have power, lad. You should know by now. Just say "Velehk Sain, I release you.", and "Poof!", I'm on my way."

DESCRIPTION

Create a program that modifies a given list of names. The program takes a file as a command-line argument with the list of names, one per line. Just like in the previous task.
The program modifies the list if it meets the following conditions using `standard library algorithms` and `std::forward_list`, and saves it in the new file with the suffix `_mod`:

- `remove` name if it contains at least one of `'c', 'b', 'l'` characters
- `replace` name with `Long one` string if it is longer than 10 characters
- `replace` name with `Short one` string if it is shorter than 4 characters
- make sure that the resulting list contains only unique values
- `reverse` resulting list

Error handling. The program prints errors to the standart error:

- in case of an invalid number of arguments - `usage: ./stdAlgoV2 [file_name]`
- in case of any other errors (invalid file, invalid file formatting, etc.) - `error`

CONSOLE OUTPUT

```
>cat names.txt
Amelie
Anne
Anne-Corinne
Anne-Marie
Aveline
Enrichetta
Helen-Elizabeth
```

```
Joe
>./stdAlgoV2 names.txt | cat -e
>cat names_mod.txt
Short one
Long one
Anne-Marie
Anne
>./stdAlgoV2 names.txt yo | cat -e
usage: ./stdAlgoV2 [file_name]
>./stdAlgoV2 directory_name | cat -e
error
>
```

SEE ALSO

<algorithm>
std::forward_list

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.