Sprint 02

Half Marathon C++

August 18, 2020

ucode

Contents

ucode

Engage

DESCRIPTION

Hello!

Hopefully, the last Sprint helped you learn how to work with containers and iterators. However, there's more to learn in `STL`. This time, you will go deeper and get acquainted with associative containers. In associative containers, data is accessed not sequentially, but by using a key. Keys are simply line numbers, they are usually used to organize stored items in a specific order and are automatically modified by containers. An example is a regular spelling dictionary where words are alphabetically placed. `STL` is a broad mechanism of functions, especially when used in conjunction with generic programming.

You have already learned how to use sequence containers. It will help you in the next Sprints. Let's delve into associative containers and discover new C++ capabilities.

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.

- How do you understand the concept of generic programming?
- What is the main difference between sequence and associative containers?
- What are the use cases of `std::set`?
- What are the use cases of `std::multiset`?
- What are the use cases of `std::map`?
- What are the use cases of `std::multimap`?
- What are the use cases of `<regex>`?
- What is the main difference between `std::set` and `std::multiset`?
- What is the main difference between `std::map` and `std::multimap`?
- What is a regular expression and in which cases it can be applied?
- What regex operations does C++ support?

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.
- Try to implement your thoughts in code.
- 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
- 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

Unique Words

DIRECTORY

`t00/`

SUBMIT

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

BINARY

`uniqueWords`

LEGEND

ACT IV, SCENE III, CONTINUED

Lifts-Her-Tail: Certainly not, kind sir! I am here but to clean your chambers.
Crantius Colto: Is that all you have come here for, little one? My chambers?
...
Crantius Colto: Plenty of time, my sweet. Plenty of time.

END OF ACT IV, SCENE III

DESCRIPTION

Create a program that saves only unique words from the given file in a new file with the suffix `_mod`. The program takes an initial text file as a command-line argument.
The unique words are case sensitive. A word is a sequence of alphanumeric and `'`, `-` characters. There must be one word per line in a new file.

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

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

Use `std::set`.

CONSOLE OUTPUT

```
>cat -e example.txt
Lifts-Her-Tail$
Certainly not, kind sir! I'm here but to clean your chambers.$
$
Crantius Colto$
Is that all you have come here for, little one? My chambers?$
$
Lifts-Her-Tail$
I have no idea what it is you imply, master.$
I am but a poor Argonian maid.$
$
```

```
Crantius Colto$
So you are, my dumpling. And a good one at that.$
Such strong legs and shapely tail.$
>./uniqueWords example.txt
>cat -e example_mod.txt
And$
Argonian$
Certainly$
Colto$
Crantius$
I$
I'm$
Is$
Lifts-Her-Tail$
My$
So$
Such$
a$
all$
am$
and$
are$
at$
but$
chambers$
clean$
come$
dumpling$
for$
good$
have$
here$
idea$
imply$
is$
it$
kind$
legs$
little$
maid$
master$
my$
no$
not$
one$
poor$
shapely$
sir$
strong$
tail$
that$
to$
what$
you$
your$
```

```
>./uniqueWords example.txt yo | cat -e
usage: ./uniqueWords [file_name]
>./uniqueWords invalid_file | cat -e
error
>
```

SEE ALSO

std::set

Act: Task 01

NAME

Count Unique Words

DIRECTORY

`t01/`

SUBMIT

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

BINARY

`countUniqueWords`

LEGEND

"And as for you, my little mortal minion... feel free to keep the Wabbajack. As a symbol of my... Oh, just take the damn thing."

- Sheogorath

DESCRIPTION

Create a program that counts unique words from the given file and saves them in a new file with the suffix `_mod`. The program takes initial text file as a command-line argument. The unique words are case sensitive.
A word is a sequence of alphanumeric and `'`, `-` characters. There must be one word with its count per line in a new file. The output format of each line is `word: <count>`.

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

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

Use `std::multiset`.

CONSOLE OUTPUT

```
> cat -e example.txt
The Wabbajack is the Daedric artifact of Sheogorath, a mysterious staff that casts random spells$
ranging in effect from the complete disintegration, to the transformation, or empowerment of$
the target.$
These abilities are able to completely alter the course of battle in one of any number of ways.$
$
The Wabbajack is one of 5 Daedric artifacts made by the Daedric Prince of Madness, Sheogorath.$
Its rogue nature is a reflection upon Sheogorath's own chaos, choosing at random to either help$
or hinder those he encounters.$
>./countUniqueWords example.txt
>cat -e example_mod.txt
5: 1$
Daedric: 3$
Its: 1$
```

```
Madness: 1$
Prince: 1$
Sheogorath: 2$
Sheogorath's: 1$
The: 2$
These: 1$
Wabbajack: 2$
a: 2$
abilities: 1$
able: 1$
alter: 1$
any: 1$
are: 1$
artifact: 1$
artifacts: 1$
at: 1$
battle: 1$
by: 1$
casts: 1$
chaos: 1$
choosing: 1$
complete: 1$
completely: 1$
course: 1$
disintegration: 1$
effect: 1$
either: 1$
empowerment: 1$
encounters: 1$
from: 1$
he: 1$
help: 1$
hinder: 1$
in: 2$
is: 3$
made: 1$
mysterious: 1$
nature: 1$
number: 1$
of: 7$
one: 2$
or: 2$
own: 1$
random: 2$
ranging: 1$
reflection: 1$
rogue: 1$
spells: 1$
staff: 1$
target: 1$
that: 1$
the: 6$
those: 1$
to: 3$
transformation: 1$
```

```
upon: 1$
ways: 1$
>./countUniqueWords example.txt yo | cat -e
usage: ./countUniqueWords [file_name]
>./countUniqueWords invalid_file | cat -e
error
>
```

SEE ALSO

std::multiset

Act: Task 02

NAME

Book Saver

DIRECTORY

`t02/`

SUBMIT

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

BINARY

`bookSaver`

LEGEND

Library in Vivec City is accessible from the Hall of Wisdom beneath the High Fane in Vivec, Temple Canton. It has a very large collection of books. Mehra Milo can be found here.

DESCRIPTION

Create a program that manages the library. Implement simple library manager using standard input stream `stdin`.
The program handles 5 commands:

- `add <bookName>` - adds a book including its content to the library
- `delete <bookName>` - removes a book including its content from the library
- `read <bookName>` - prints content of the book to the standard output
- `list` - prints the list of all available books in the library to the standard output
- `quit` - quits the program

`<bookName>` is a name of the file which is stored in the `./library` directory.

The program prints:

- `enter command:>` - before each command
- `<bookName> added` - if a book exists in the `./library/<bookName>` and was successfully added to the library
- `<bookName> deleted` - in case of successful deletion of the book from the library
- `invalid book` - if the book does not exist in the `./library/<bookName>`, or it has not been added to the library previously
- `invalid command` - in case of invalid commands
- `bye` - in case of quit

Use `std::map`.

CONSOLE OUTPUT

```
>ls ./library
The_Oblivion_Crisis.txt
>./bookSaver
enter command:> yo
invalid command
enter command:> add
invalid command
enter command:> add not_existing_book.txt
invalid book
enter command:> add not_existing_book.txt yo
invalid command
enter command:> add The_Oblivion_Crisis.txt
The_Oblivion_Crisis.txt added
enter command:> add The_Oblivion_Crisis.txt
The_Oblivion_Crisis.txt added
enter command:> list
The_Oblivion_Crisis.txt
enter command:> read The_Oblivion_Crisis.txt
At the turning of the Fourth Age, in the year 3E 433, the Emperor Uriel Septim VII was assassinated
...
# book contents
...
Emperor dead, the barrier to Oblivion is sealed forever.
enter command:> delete yo
invalid book
enter command:> delete The_Oblivion_Crisis.txt
The_Oblivion_Crisis.txt deleted
enter command:> list
enter command:> quit
bye
>
```

SEE ALSO

`std::map`

Act: Task 03

NAME

TES Library

DIRECTORY

`t03/`

SUBMIT

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

BINARY

`tesLibrary`

LEGEND

The various books that appear throughout Skyrim can grant quests, increase certain skills, or record locations on the world map. Books vary from simple stories, to letters, recipes, notes, and journals that assist in quests and provide snippets of lore that help players become more familiar with the culture, people, and history of Tamriel.

DESCRIPTION

Create a program that catalogs authors' books from the given file. The program takes an initial text file containing list of authors and books as a command-line argument.
Every valid file meets requirements listed below:

- every line contains an author name and a book title
- both author name and book title are surrounded by quotes `"`
- the author name and the book title are separated by a single colon `:`
- only whitespaces are allowed between `:` and `"`
- see a detailed example of possible input file formatting in the CONSOLE OUTPUT

Use `std::regex` to parse input file.

The program prints the list of unique author names with the grouped list of its book titles. See an output format of authors and their books in the CONSOLE OUTPUT.

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

- in case of invalid number of arguments - `usage: ./tesLibrary [file_name]`
- for the first invalid line of the input file if it does not match the rules listed above - `Line <number> is invalid`
- in case of any other errors - `error`

Use `std::multimap`.

CONSOLE OUTPUT

```
>cat -e books.txt
"Carlovac Townway" : "2920, vol 06 - Mid Year" $
        "Waughin Jarth" :"A Dance in Fire, Book VII" $
$
 "Carlovac Townway":"2920, vol 03 - First Seed"$
"Reven" : "Beggar"$
$
 "Aldetuile":           "Cats of Skyrim"        $
"Warrior" : "Reven"$
        "Torhal Bjorik" : "There Be Dragons"$
"Waughin Jarth": "The Argonian Account, Book IV"        $
$
>./tesLibrary books.txt | cat -e
Aldetuile:$
 1: Cats of Skyrim$
Carlovac Townway:$
 1: 2920, vol 06 - Mid Year$
 2: 2920, vol 03 - First Seed$
Reven:$
 1: Beggar$
Torhal Bjorik:$
 1: There Be Dragons$
Warrior:$
 1: Reven$
Waughin Jarth:$
 1: A Dance in Fire, Book VII$
 2: The Argonian Account, Book IV$
>./tesLibrary example.txt yo | cat -e
usage: ./tesLibrary [file_name]
>./tesLibrary invalid_file | cat -e
error
>cat -e invalid.txt
"Carlovac Townway" : 2920, vol 06 - Mid Year"$
>./tesLibrary invalid.txt | cat -e
Line 1 is invalid
>
```

SEE ALSO

std::multimap
std::regex

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.