Sprint 00

Half Marathon C++

August 17, 2020

ucode

Contents

Engage

DESCRIPTION

Hi, friend!

Congratulations, you've already taken the first step into the world of programming. It's time to jump into a new technology. We want to introduce you to C++, a direct descendant of the C language. There is a big difference between these languages that we propose you to explore in practice.

Forget about C, you are in the C++ world now!

C++ increases capabilities and has a much wider range of programming styles. In addition, it supports object-oriented programming that improves modularity of the created software. Welcome to the new perspective direction - C++.
Personal growth involves accumulation of gained knowledge and the stimulus to learn new ones. Make it a habit to learn something new at least every day.

BIG IDEA

Self-evolution in learning.

ESSENTIAL QUESTION

How to build new knowledge on top of prior experience?

CHALLENGE

Dive into C++.

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 to create an efficient work atmosphere?
- What components does a productive programming environment consist of?
- What is the difference between C and C++?
- What standard C++ libraries do you know?
- What is the difference between C and C++ strings?
- What are the features of dynamic memory allocation in C++?
- What are the features of type casting in C++?
- How to declare data structures in C++?
- What is the true advantage of the usage of the keyword `auto`?
- What are the differences between references and pointers?
- What standard containers do you know and how 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.

- Customize your work environment and focus on motivating yourself to achieve results.
- 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.
- Think through alternative solutions to help you develop a product mindset.
- Choose a convenient syntax highlighting code editor.
- Get acquainted with the features of working with `CMake`.
- Create a simple console app `Hello world` using C++.
- 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)
- 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

Hello Tamriel

DIRECTORY

`t00/`

SUBMIT

`helloTamriel.cpp, CMakeLists.txt`

BINARY

helloTamriel

LEGEND

"Why do you think your world has always been contested ground, the arena of powers and immortals? It is Tamriel, the Realm of Change, brother to Madness, sister to Deceit."

- Mankar Camoran

DESCRIPTION

Create a program that outputs the text below to the standard output followed by a newline.

CONSOLE OUTPUT

```
>./helloTamriel | cat -e
Hello Tamriel$
>
```

SEE ALSO

iostream
ostream

Act: Task 01

NAME

Print Dialog

DIRECTORY

`t01/`

SUBMIT

`printDialog.h, printDialog.cpp`

LEGEND

"M'aiq's father was also called M'aiq.
As was M'aiq's father's father.
At least, that is what his father said.
But then again, you can never trust a liar."

- M'aiq the Liar

DESCRIPTION

Create a function that:

- takes two C++ style strings: `name` and `sentence`
- prints the quote in the format `<name> says: "<sentence>"` with a new line

See CONSOLE OUTPUT for an example of a program, which uses such function.

SYNOPSIS

```
void printDialog(const std::string& name, const std::string& sentence);
```

CONSOLE OUTPUT

```
>./printDialog | cat -e
Guard says: "I used to be an adventurer like you. Then I took an arrow in the knee..."$
>
```

SEE ALSO

std::string

Act: Task 02

NAME

Wallet Manager

DIRECTORY

`t02/`

SUBMIT

`walletManager.h, main.cpp, walletManager.cpp, CMakeLists.txt`

BINARY

walletManager

LEGEND

The Septim is the main currency in all of Tamriel. It is named after the dynasty that rules the Septim Empire founded by Tiber Septim. It is officially called a Septim due to the engraving of Tiber Septim on the obverse side.

We need some money, to survive in this world. Hmm... I think we can create them!

DESCRIPTION

Create a program that creates and destroys wallets:

- compile the program with our `main.cpp` which is listed in the SYNOPSIS
- use a struct `Wallet` in `walletManager.h` which is listed in the SYNOPSIS
- create a function that creates a single wallet
- create a function that destroys a single wallet
- create a function that creates multiple wallets. Loops are forbidden
- create a function that destroys multiple wallets. Loops are forbidden

In C we used to allocate and free memory with `malloc` and `free` functions. Forget it. Now you are in C++ world.

SYNOPSIS

```
struct Wallet {
    int septims;
};
```

```cpp
#include "walletManager.h"

int main() {
    Wallet* wallet = createWallet(10);
    std::cout << "I've got " << wallet->septims << " septims in the wallet." << std::endl;
    destroyWallet(wallet);

    int amount = 5;
    Wallet* wallets = createWallets(amount);
    for (int i = 0; i < amount; i++) {
        wallets[i].septims = i * i;
        std::cout << i << " wallet: " << wallets[i].septims << " septims." << std::endl;
    }
    destroyWallets(wallets);
}
```

CONSOLE OUTPUT

```
>./walletManager | cat -e
I've got 10 septims in the wallet.$
0 wallet: 0 septims.$
1 wallet: 1 septims.$
2 wallet: 4 septims.$
3 wallet: 9 septims.$
4 wallet: 16 septims.$
>
```

SEE ALSO

Dynamic memory
Data structures

Act: Task 03

NAME

Cast Spells

DIRECTORY

`t03/`

SUBMIT

`castSpells.h, castSpells.cpp`

LEGEND

"Welcome, welcome! We were just beginning. Please, stay and listen. So, as I was saying, the first thing to understand is that magic is, by its very nature, volatile and dangerous. Unless you can control it, it can and will destroy you."

- Tolfdir

DESCRIPTION

Create two functions:

- `castFloatToInt` casts `float` to `int`
- `castToNonConstIntPtr` casts `const int*` to `int*`

Forget about C-like casting. Don't do that here.

SYNOPSIS

```
int castFloatToInt(float number);
int* castToNonConstIntPtr(const int* ptr);
```

SEE ALSO

Type Casting

Act: Task 04

NAME

Reference Or Pointer

DIRECTORY

`t04/`

SUBMIT

`referenceOrPointer.h, referenceOrPointer.cpp`

LEGEND

"Those born under the Mage have more magicka and talent for all kinds of spellcasting, but are often arrogant and absent-minded."

- The Firmament

DESCRIPTION

Create two functions that multiply the integer value that is stored in the pointer and in the reference by `mult`.

SYNOPSIS

```
void multiplyByPointer(int* ptr, int mult);
void multiplyByReference(int& ref, int mult);
```

SEE ALSO

Pointers
References

Act: Task 05

NAME

Automaton

DIRECTORY

`t05/`

SUBMIT

`main.cpp, CMakeLists.txt`

BINARY

automaton

LEGEND

"Dwarven military machines also range from the human-sized "Sphere" warrior, which patrols the interiors of the ruins as a harmless ball only to emerge from it as a fully armed and armored automaton fighter, to the justly feared "Centurion" whose height ranges from twice to several hundred times human size depending on which reports you believe."

- Calcelmo

DESCRIPTION

Create a program that:

- receives unit specifications as command-line arguments.
- casts name into a `std::string`, level into an `int`, health into a `float`, and stamina into a `double`
- prints unit specifications to the standard output. See the output format in the CONSOLE OUTPUT
- uses `auto` instead of any other variable types
- informs a user if not enough command-line arguments have been provided with the message `usage:./automaton [name] [level] [health] [stamina]` to the standard error
- informs a user if invalid command-line arguments have been provided with the message `Invalid argument: <value>` to the standard error

Try using `auto` whenever you can.

CONSOLE OUTPUT

```
>./automaton "Dwarven Centurion Master" 36 "153.139" "184.932" | cat -e
Name = Dwarven Centurion Master$
Level = 36$
Health = 153.139$
Stamina = 184.932$
```

```
>./automaton | cat -e
usage:./automaton [name] [level] [health] [stamina]
>./automaton "Dwarven Centurion" -2 "+3.4434" "yo" | cat -e
Invalid argument: yo
>./automaton "Dwarven Centurion Master" ++36 "153.139" "184.932" | cat -e
Invalid argument: ++36
>./automaton "Dwarven Centurion Master" 36 "+-153.139" "184.932" | cat -e
Invalid argument: +-153.139
>
```

SEE ALSO

auto
std::cerr

Act: Task 06

NAME

Mead Song

DIRECTORY

`t06/`

SUBMIT

`meadSong.h, main.cpp, meadSong.cpp, CMakeLists.txt`

BINARY

meadSong

LEGEND

"I wonder if Vilod is still making that mead with Juniper Berries mixed in."

- Ralof

DESCRIPTION

Create a program that outputs the lyrics of a song. Pattern is simple. Look at the CONSOLE OUTPUT carefully to see the lyrics. You need to recognize the pattern by yourself.

Start with 99 bottles of mead, until the bottles run out. Instead of `...` output all song lines. Think wise! You need to remove as much code duplication as you possibly can.

Hardcoding of entire song is forbidden!

CONSOLE OUTPUT

```
>./meadSong
99 bottles of mead on the wall, 99 bottles of mead.
Take one down and pass it around, 98 bottles of mead on the wall.

98 bottles of mead on the wall, 98 bottles of mead.
Take one down and pass it around, 97 bottles of mead on the wall.

...

2 bottles of mead on the wall, 2 bottles of mead.
Take one down and pass it around, 1 bottle of mead on the wall.

1 bottle of mead on the wall, 1 bottle of mead.
Take it down and pass it around, no more bottles of mead on the wall.

No more bottles of mead on the wall, no more bottles of mead.
Go to the store and buy some more, 99 bottles of mead on the wall.
>
```

Act: Task 07

NAME

Inventory

DIRECTORY

`t07/`

SUBMIT

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

BINARY

`inventory`

LEGEND

Every traveller needs an inventory! The number of items that a character may carry is based on their weight and how much the character can hold. In most cases, the more items you have, the more weight you have (arrows, quest-related items with some exceptions, and keys do not have any weight). The inventory is divided into such categories: Favorites, Weapons, Apparel, Potions, Scrolls, Food, Ingredients, Books, Keys and Miscellaneous.

DESCRIPTION

Create a program that manages an inventory. Implement a simple inventory manager using the standard input stream `stdin`. The inventory has a limited size - only 12 items.

There are 4 item types:

- `w` - weapon
- `f` - food
- `a` - armor
- `p` - potion

The program handles 5 commands:

- `insert <itemType>` - adds the item to the inventory
- `remove <itemType>` - removes the item from the inventory
- `show` - shows the inventory contents
- `help` - shows available commands
- `exit` - quits the program

The program prints:

- `Enter command:>` - before each command
- `<itemType> was inserted.` - in case of successful item insertion
- `<itemType> was removed.` - in case of successful item removal

- `Inventory { w f a p w w - - - - - - }` - inventory example with 5 items where `-` is the status of empty slots
- errors:
  - `Invalid command.` - in case of invalid commands
  - `Invalid item.` - in case of invalid items
  - `Inventory is full.` - in case of full inventory

    If there is more than 1 error, the program prints only 1 error message according to the priority listed above.
- the help message - see in the CONSOLE OUTPUT for detailed example
- `Bye.` - in case of exit

For nice implementation, look at `sequential containers` and choose one to store your items in.

CONSOLE OUTPUT

```
>./inventory
Enter command:> help
Available commands:
1. insert <itemType>
2. remove <itemType>
3. show inventory
4. help
5. exit
Enter command:> insert w
w was inserted.
Enter command:> insert a
a was inserted.
Enter command:> insert p
p was inserted.
Enter command:> insert w
w was inserted.
Enter command:> insert f
f was inserted.
Enter command:> insert w
w was inserted.
Enter command:> insert a
a was inserted.
Enter command:> insert a
a was inserted.
Enter command:> insert p
p was inserted.
Enter command:> insert p
p was inserted.
Enter command:> insert w
w was inserted.
Enter command:> insert p
p was inserted.
Enter command:> insert a
Inventory is full.
```

```
Enter command:> show
Inventory { w a p w f w a a p p w p }
Enter command:> remove f
f was removed.
Enter command:> show
Inventory { w a p w w a a p p w p - }
Enter command:> remove f
Invalid item.
Enter command:> yo
Invalid command.
Enter command:> insert z
Invalid item.
Enter command:> insert f yo
Invalid command.
Enter command:> exit
Bye.
>
```

SEE ALSO

std::cin
containers

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.