Exercise Session
Week 07

Adel Gavranović
agavranovic@student.ethz.ch

Overview

▸ polybox for session material   ▸ Mail to TA

Today's Topics

Introduction

References

Characters

Repetition: Floating Point Numbers

The elephant in the room

That's right: we're online now!

- I want you to participate
- Turn on your camera (yes, all of you)
- There'll be more exercising and less recapping
- I won't record the sessions. I want to ensure that you come to the sessions instead of just watching it passively
- A lot might go wrong the first few times
- My internet connection is not the most stable one (send your complaints directly to SALT)
- Not much will change on how the session will be held, except now you can see my whole face
- I'm going to stick around for a while after each session for more specific questions

Questions from last Exercise Session

How will the exam look like?

I honestly don't know myself, but it will deviate from the ones in the past.

- Greater focus on actually programming stuff yourself (*practice!*)
- It will basically be entirely autograded
- The old exams are still a good source for "testing" your knowledge on certain topics
- Exam-coding-questions from past semesters should be on the course website too

Comments on last [code] expert Exercises

- When using any "fixed" value (like the error from the float-comparing exercise), make it into a const variable, i.e. const double error = 0.0001
- Name everything *exactly* like the task description named it
- If you ever receive no feedback at all, it usually means your code is very good (or that I'm running late with corrections and have to hurry a bit)
- Having a good code structure is getting much more important now. If you want anyone to read and understand your code (*or correct it*), make sure to write it in a good style (i.e sensible variable naming, structure, consistent indentations, useful comments, const-ness)

Intro
○○○○●○

References
○○○○○○○○○

Characters
○○○○○○

Floating Point Numbers
○○

Questions or Comments re: Exercises?

Learning Objectives Checklist

Now I...

- ☐ can do *Program Tracing*
- ☐ understand what vectors are and how they work conceptually
- ☐ can create, modify and iterate over std::vectors
- ☐ know how to write a program that can modify ASCII characters
- ☐ I can *trace* the aforementioned ASCII program

Code Example with Program Tracing I

```cpp
int a = 3;
int& b = a;

b = 2;

std::cout << a;
// output "2"
```

Code Example with Program Tracing II

```cpp
void foo(int i){
   i = 5;
}

int main(){
   int i = 4;
   foo(i);
   std::cout << i << std::endl;
}
```
```
// output: "4", but why?
```

References are usually used as function parameters or return values (we'll see an example of this later). If the parameters of a function are not of the reference type, we say that we *"pass them to the function by value"*, which is what we did in all of our functions so far (and in this one). In this case the function makes its own copies of the values, and uses these copies to do something in the function body.

Code Example with Program Tracing III

```cpp
void foo(int& i){
   i = 5;
}

int main(){
   int i = 4;
   foo(i);
   std::cout << i << std::endl;
}
```
```cpp
// output: "5", but why?
```

If a parameter of a function is of the reference type (`&`), hence will become an alias of the call argument, we say that we "*pass the argument by a reference*".

Exercises

Q: Why do we need references? Aren't the types we say before enough?

A: Multiple reasons:

- You can return (or rather modify) multiple results from a function
- We avoid copying parameters, which improves performance: sometimes we pass *huge* vectors to a function, and we don't want to waste performance copying the whole thing. A reference tells the function where that parameter (int, double, vector, whatever) is stored, this way the function can operate on the parameter directly
- Sometimes copying just won't work (std::cout for example, but don't worry about that for now)

Extensive Program Tracing Guide

Questions?

Intro
References
Characters
Floating Point Numbers
000000
000000●00
000000
00

References as Return Types

We've seen function parameters being of a reference type, but references can also be used for the return type of a function:

```
int& increment(int& m){
   return ++m;
}

int main(){
   int n = 3;

   increment(increment(n));

   std::cout << n <<;
}
// output: "5", but why?
```

Intro
000000

References
000000000

Characters
000000

Floating Point Numbers
00

Questions?

Exercises

Task

Solve the exercises in the following PDF with *Program Tracing*

:: open week_7_exercises.pdf now ::

Exercise

Task

Think about how to solve the [code]expert exercise
"Converting Input to UPPER CASE" with pen and paper.

Write a program that reads a sequence of characters delimited
by the new-line character ("/n") 1and then outputs the sequence
with all lower-case letters changed to UPPER-CASE letters.
Please put the code that converts the entire sequence to
upper-case and a single character to upper-case into separate
functions (you should have at least three functions).

Hints: As you've seen in the lecture, variables of type char can
be treated as numbers. Store the words in a std::vector.

How to `std::vector`

- Don't forget to `#include <vector>`
- Think of vectors as a series of slots, each containing a value of the type you've specified
- You can basically treat `std::vector` as just another type
- `std::vector<int> myvector{1,2,3};` to initialize a vector
- there are multiple ways to initialize a vector, check out the summary or search online for more
- `myvector[n-1]` to look at the n'th entry in a vector
- use `myvector.push_back(x)` to append `x` to the vector (be careful with the type)

Exercise

Task

Try to solve the [code]expert exercise "Converting Input to UPPER CASE" in the [code]expert IDE.
Optional: do it in small groups (breakout rooms)

Solution "Converting Input to UPPER CASE"

```
#include <iostream>
#include <vector>
#include <ios>

// POST: Converts the letter to upper case.
void char_to_upper(char& letter){
   if('a' <= letter && letter <= 'z'){
      letter -= 'a' - 'A'; // 'a' > 'A'
   }
}

// POST: Converts all letters to upper-case.
void to_upper(std::vector<char>& letters){
   for(unsigned int i = 0; i < letters.size(); ++i){
      char_to_upper(letters.at(i));
   }
}
```

Solution "Converting Input to UPPER CASE"

```
std::cin >> std::noskipws;

std::vector<char> letters;
char ch;

// Step 1: Read input.
do{
   std::cin >> ch;
   letters.push_back(ch);
}while(ch != '\n');

// Step 2: Convert to upper-case.
to_upper(letters);

// Step 3: Output.
for(unsigned int i = 0; i < letters.size(); ++i){
   std::cout << letters.at(i);
}
```

Questions?

Normalized Floating Point Number Systems

Task

Ask questions on stuff related to NFPNS that you didn't
understand yet. I will try my best to answer them.

Guided Exercise

::open NFPN_TASK.pdf::

Task

Try to solve the task (as a group).
I'll guide you when I think it's necessairy.