# Exercise Class 7

## Objectives

The objectives of this exercise session are:

1. The students after the class should be able to trace and write programs that use references.
2. The students after the class should be able to write programs that create, modify, and iterate over vectors.
3. The students after the class should be able to trace and write programs that modify ASCII characters.

## Seen in class

During the last class, students have seen: Reference Types: Definition and Initialization, Pass By Value, Pass by Reference, Temporary Objects, Const-References. Vector Types, Sieve of Erathostenes, Memory Layout, Iteration. Characters and Texts, ASCII, UTF-8, Caesar Code.

## Contents

## 1 Homework Questions (< 10 min.)

Ask students if they have any questions about the homework exercises. If so, provide them support.

## 2 References (40 min.)

### 2.1 Explanation (20 min., blackboard)

> References allow us to build an alias for an already existing object.

Show to students how to trace the following example by using the Program tracing handout (`https://lec.inf.ethz.ch/ifmp/2023/guides/tracing/references.html`).

```
int a = 3;
int& b = a;
b = 7;
std::cout << a; // Output: 7
```

> References are usually used as function parameters or return values. If the parameters of a function are not of the reference type, we say that we "pass them to the function by value". This is what we did in all of our functions until now.
>
> In this case the function makes its own copies of the values, and uses these copies to do something in the function body.

Show to students how to trace the following program:

```
void foo (int i) {
    i = 5;
}
int main () {
    int i = 4;
    foo(i);
    std::cout << i << std::endl;
}
```

> 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".

Show to students how to trace the following program:

```
void foo (int& i) {
    i = 5;
}
int main () {
    int i = 4;
    foo(i);
    std::cout << i << std::endl;
}
```

When explaining references, please avoid using the word "pointer" because students get confused.

Streams are not covered in much detail in the lecture anymore, so the example 3 below for why references are needed might be rather confusing for some students. Students don't need to understand the details. It's enough if they are aware of it.

**Question**

Why do we need references? Why are the types we saw before not enough?

**Possible Answer**

1. References allows us to return more than one result from a function. For example, see the "Quadratic Equation" code example in [code]expert or the corresponding code snippet below:

```
// POST:  return value is the number of distinct real
//        solutions of the quadratic equation
//        ax^2 + bx + c = 0. If there are infinitely
//        many solutions (a = b = c = 0), the return
//        value is -1. Otherwise, the return value
//        is a number n from {0,1,2} and the solutions
//        are written to x1, x2 (if existent)
int solve_quadratic_equation(
    const double a, const double b, const double c,
    double& x1, double& x2)
```

2. By passing a reference to a function, we avoid copying the parameter. All a reference does, is to tell the program the location of the variable which the reference references. Now imagine you have a function that reads one entry in a gigantic vector:

```
void read_i (Vector& v, unsigned int i);
```

   The vector `v` better be a reference, else one would copy the whole thing every time an entry is read. If we use references it just tells the function where to find the one original vector so it can read the entry. An example for a function that takes a vector as reference can be found in the "Reversing Vector" example in [code]expert.

   When we deal with small data types, such as `char, short, int`, references should be used only when it is necessary. For example, if an `int` argument is only read inside a function, passing it by value is as efficient as by reference but simplifies reasoning. Calling a function that takes a reference might modify its value. However if it takes a non-reference, it is clear to callers that the value cannot be modified. Thus, passing it by value is the preferred way in this situation.

3. Sometimes it is impossible to copy something. `std::cout` is an example of this: There is only one output stream, we cannot make a copy of it like we could make a copy of an integer.

   ```
   int a = 5;
   int b = a;  // making a copy of an int
   std::ostream o = std::cout; // Error: copying std::cout
         is impossible
   std::ostream& o = std::cout; // This works, try it!
   ```

*Questions?*

Not only function parameters can be references, but also references can be used as function return types. This is, for example, useful for:

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

int main () {
  int n = 3;
  increment (increment (n));
  std::cout << n << "\n"; // outputs 5
  return 0;
}
```

Here, the return type of the inner `increment (n)` is an `int` reference, returning `n` as an lvalue. The outer `increment` function on the other hand absorbs the same l-value as its argument, resulting in $n = 5$. As it will turn out in a few weeks this is exactly how the `++` operator is built.

*Questions?*

### 2.2 Exercise: References (20 min.)

Do the exercises given in `week_7_exercises.pdf`:

1. Show a slide with a task description.
2. Ask students to think for 5-8 minutes on their own.
3. Ask some student to present his or her solution. Discuss it and compare with the master solution.

*Questions?*

## 3 Characters (40 min.)

If someone asks, ASCII means American Standard Code for Information Interchange.

## 3.1 Exercise: Converting Input to UPPER CASE (40 min.)

### 3.1.1 Reading Task and Individual Thinking (10 min.)

1. Open the task “Converting Input to UPPER CASE” in [code]expert and show the task description to students.
2. Ask students to (individually) think for 10 minutes how they would solve this task with pen-and-paper.

We repeat the task for your reference:
Write a program that reads a sequence of characters, delimited by the new-line character, as a vector of chars. Then the program should output the sequence with all lower-case letters changed to UPPER-CASE letters.
To read the sequence you can:

1. read a single character from standard input;
2. insert it into a vector of chars;
3. repeat until you find a new line character (`'\n'`).

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).
*Hint:* variables of type `char` can be treated as numbers.

In the lecture, `push_back()` is only quickly mentioned. Since students will need to use `push_back()` in the weekly exercises this week, make sure everybody has understood how to use this method.

### 3.1.2 Pair Programming (15 min., programming environment)

Ask students to discuss their ideas, and write the program. Give them 15 minutes.

### 3.1.3 Share Solution (15 min., programming environment)

Ask students to dictate the solution; write it in the project. Point out and discuss any mistakes and inefficiencies.

The master solution for your reference:

```
#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));
  }
}

int main () {
  // Say to std::cin not to ignore whitespace.
  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);
  }
  return 0;
}
```

## 4 Additional Material: Repetition (10 min.)

The additional material sections provide some additional classroom activities which you can use in case you decide that the provided lesson plan does not work for you.

This section should be covered only in the unlikely case you still have time after covering the other topics and you have the impression that the students have well understood references and vectors. Otherwise, just inform the students that they will find the exercise on the website and be prepared in case of questions next week.

### 4.1 Exercise: Floating Point Number Systems (10 min.)

#### 4.1.1 Task (5 min., slides)

1. Show the slide with the task description (**slide 1**) of `main.pdf`:.

2. Ask students to solve the exercise on their own. Give them 5 minutes.

For your reference, we repeat the task below:
Consider the normalized floating point number system

$$F^*(\beta, p, e_{\min}, e_{\max})$$

with $\beta = 2$, $p = 3$, $e_{\min} = -4$, $e_{\max} = 4$.
Compute the following expressions as the parentheses suggest, representing each intermediate result (and the final result) in the normalized floating point system according to the rules of computing with floating point numbers.

| $(10 + 0.5) + 0.5$ | | | | $(0.5 + 0.5) + 10$ | | | |
|---|---|---|---|---|---|---|---|
| | decimal | | binary | | decimal | | binary |
| | 10 | | ????? | | 0.5 | | ????? |
| $+$ | 0.5 | | ????? | $+$ | 0.5 | | ????? |
| $=$ | | | ????? | $=$ | | | ????? |
| $+$ | 0.5 | | ????? | $+$ | 10 | | ????? |
| $=$ | ?? | $\leftarrow$ | ????? | $=$ | ?? | $\leftarrow$ | ????? |

### 4.1.2 Solution (5 min., slides)

Show the solution in the remaining slides of `main.pdf`.

The master solution for your reference:

| $(10 + 0.5) + 0.5$ | | | | $(0.5 + 0.5) + 10$ | | | |
|---|---|---|---|---|---|---|---|
| | decimal | | binary | | decimal | | binary |
| | 10 | | $1.01 \cdot 2^3$ | | 0.5 | | $1.00 \cdot 2^{-1}$ |
| $+$ | 0.5 | | $0.0001 \cdot 2^3$ | $+$ | 0.5 | | $1.00 \cdot 2^{-1}$ |
| $=$ | | | $1.01 \cdot 2^3$ | $=$ | | | $1.00 \cdot 2^0$ |
| $+$ | 0.5 | | $0.0001 \cdot 2^3$ | $+$ | 10 | | $1010.0 \cdot 2^0$ |
| $=$ | 10 | $\leftarrow$ | $1.01 \cdot 2^3$ | $=$ | 12 | $\leftarrow$ | $1.10 \cdot 2^3$ |

*Questions?*