UNIVERSITÄT BASEL

Thorsten Möller
Salome Müller - salo.mueller@stud.unibas.ch
Maarten Schenk - maarten.schenk@unibas.ch

UNI BASEL

Programming Paradigms – C++ FS 2020

Exercise 2 Due: 19.04.20109 23:59:59

Upload answers to the questions **and source code** before the deadline via courses.cs.unibas.ch. Due to the measures taken to curb the Coronavirus pandemic, programs do **not** have to be demonstrated during the exercise slots. Instead, for **every** task and subtask you **must explain** your solution in detail. If a task involves writing or completing some code, you **must** provide code that is **commented in detail** (how it works, things that need to be done to make it working, things that must be satisfied, and so on).

Also note, of all mandatory exercises given throughout the course, you must score at least 2/3 of the total sum of their points to be allowed to take the final exam.

Modalities of work: The exercise can be completed in groups of at the most 2 people. Do not forget to provide the full name of all group members together with the submitted solution.

Question 1: Pointers (6 points)

In this exercise you are asked to analyze the following, admittedly contrived, C++ code. What is its output? Explain what is happening in each line. Also, the last line contains the expression `((p + 5) - p)`. Is there a difference to using `(p + 5 - p)`? On the other hand, what would be the problem if we would write `(p + (5 - p))` instead?

Hint: If the output is not uniquely determinable describe of what kind it would be.

```
int a = 3;
int b = 2;
int *p = &a, **q = &p;
*p *= 2**&b***q; // Never write such minified lines in your code!
                 // This is just for the sake of it.
p = &b;
(*p)++;
p -= 5;
cout << a << " " << b << " " << p << " " << ((p + 5) - p) << endl;
```

Question 2: Pointers (8 points)

a) The following C++ code is supposed to retrieve an address value created within the function `foo`. Does that make sense? Explain your answer in any case.

```
int& foo() {
   int x = 3;
   return x;
}

int main() {
   int print_out = foo();
   cout << &print_out << endl;
}
```

(2 points)

b) Write a C++ function that takes two ordered Integer arrays of arbitrary size, computes the ordered array that contains the elements of both arrays and returns the pointer to the beginning of the new array. Write a main function to test your implementation. You are not allowed to use square brackets to access array values. Use this function declaration:
`int* firstCommonSequence(int* a, int alen, int* b, int blen, int& len);`

Example: Let's say that we get the arrays $[1, 2, 3]$ and $[1, 2, 4]$ as input. Then we get the pointer to the first element of the array $[1, 2, 3, 4]$ as output.

(3 points)

c) Consider your answer to b). Is it possible to write a reliable function fulfilling the requirements of b) such that the pointer to the first element of the new array is the same as the pointer to the first element of the first given array? If yes, implement a function that guarantees this and explain your implementation. If not, explain why.

Please provide a detailed explanation.

(2 points)

d) The following C++ fragment contains some errors. Find, explain and fix them.

```
int a = 1, int b = 1;
int* p1, p2;
p1 = &a, p2 = b;

if (*p1 == &p2) {
  cout << "a != b" << endl;
} else {
  cout << "a == b" << endl;
}
```

(1 points)

Question 3: Function Pointers (8 points)

This exercise is about function pointers. In practice you often need to define an interface that can use different functions within an algorithm. Design an algorithm that can *compare* two arrays of the same length according to different *comparator* functions. More precisely, the algorithm takes two `double` arrays of size 3 and a *comparator* function as input and returns whether one array is larger than the other in regard to the given comparator; that is, return either of $-1, 0, 1$ if the first is smaller, equal, or larger than the second one.

Implement two comparators: One that compares the two arrays according to the sinus value of the 3rd element (see `math.h`). The other comparator considers an array to represent a point in the Euclidian space $\mathbb{R}^3$ and compares the points (arrays) according to the Euclidean distance (see `https://en.wikipedia.org/wiki/Euclidean_distance`) from the point of origin $(0, 0, 0)$; i.e., the first point is larger than the second if it is more distant from $(0, 0, 0)$ regarding the Euclidian norm.

Use function pointers to pass the comparator function into the function.

Question 4: Structures and Pointer Arithmetic (10 points)

In this task you are asked to implement a *circular buffer* of arbitrary size whose elements are `double` numbers. The implementation should be done similar to a unidirectional linked list. Of course, rather than a first and last element, a cirucular buffer does not know the notion of a start and end - it is a ring. Therefore think about how a unidirectional linked list can be extended/changed to make it circular. Instead of a start and end element, our cirular buffer shall have a *next* element, which is the element that is to be returned *next* when cycling through the buffer. It is put forward either implicitly or explicitly as defined by functions below. For instance, if put forward implicitly by one step as defined by the `next` function, consecutive calls to this function do not access the same position again, except it is a ring of size $s = 1$. Finally, the implementation should allow repositioning the next index at any time to another element in the ring.

All in all, your implementation must provide the following functions at least; you may however add more functions to create and destroy your buffers. Finally, keep memory management in mind.

```
void add(CircularBuffer& buffer, double value);
// Inserts the value into the ring at the position before the next element.

double take(CircularBuffer& buffer);
// Removes the next element from the ring and returns its value; thus,
// also putting forward the position of the next element by one step.
// If the buffer is empty the function returns NaN (not a number) and
// otherwise does nothing.

double next(CircularBuffer& buffer);
```

```
// Returns the next element and puts forward the position of the next
// element by one step. If the ring is empty, the method returns NaN.

unsigned int size(CircularBuffer& buffer);
// Computes the current size of the ring. You may not design the
// structure such that it explicitly represents the size. Instead you
// need to perform one cycle through the ring to compute its size (i.e.,
// the function has a complexity linear in the size of the ring).

void forward(CircularBuffer& buffer, unsigned int steps);
// Moves forward the next element by the specified number of steps
// modulo the size of the ring, and provided the ring is non-empty.
// If the ring is empty, the function does nothing.
```

Question 5: Match Three in Python (10 points)

In this exercise you will implement the well known game *Match Three* on the terminal. The rules are as follows:

- A single player tries to score points by matching three of a kind
- The player specifies which two adjacent blocks he/she wants to swap
- When the swap results in three of a kind in a straigt line (horizontal or vertical) the blocks are destroyed and the player receives points
- Normally, the player can only swap blocks when the swap leads to a match. In our case you are allowed to disregard this constraint
- Blocks fall from above into the now empty slots
- If this results in more matches of three of a kind or more, these matching blocks are again destroyed and new blocks fall into their place
- Normally, the game runs until the player cannot create any more matches. In our case you are allowed to disregard this constraint

Important: You can find an incomplete implementation in the file `matchThree.py`.