Exercise 10:

Submit your solutions (source code) to the questions below by email to your instructor and TA(s) by Monday, December 10th (16:30).

Question 1: Defining a template class (35 points).

This question will test your understanding on how to create a template class.

Please define a pair of elements named MyPair and parameterized over two types named FirstType and SecondType. MyPair contains two fields named first and second. Please provide a constructor for the class MyPair that takes both elements of the pair as arguments (see the code below). Save the code for MyPair in a file named MyPair.h. The code below is an example of how to manipulate objects of type MyPair:

```
#include <string>
#include <iostream>
#include "MyPair.h"

using namespace std;

int main(void) {
  MyPair<string, int> apair("one", 1);

  cout << "Pair: " << apair.first
```

```
    << " " << apair.second << endl;

  return 0;
}
```

Question 2: Defining a template function (35 points).

As seen during lecture, template functions have type inference while template classes do not. For that reason, it is common to find a template class paired with template functions that produce instances of the template class.

Write a template function: make_myPair() that takes as arguments the two elements of the pair and return a pair made with these two elements. Save the code for the function make_myPair() in the file MyPair.h. The code below shows how to use this function to make a pair:

```
#include <string>
#include "MyPair.h"

template < typename F, typename S >
void print_pair(const MyPair<F, S>& pair) {
    cout << pair.first << " "
         << pair.second << endl;
}

int main(void) {
  print_pair(make_myPair(string("one"), 1));
  return 0;
}
```

The advantage of using such template function is that if we need the pair as a parameter of a function, we can save typing by using a function such as make_myPair because we do not have to type the type of the pair's elements.

Question 3: More template class (30 points).

In exercise 05, you wrote code for a stack of integers. Transform this code such that the stack holds elements of a given parametric type. The code below illustrates how to use this parametric type.

```
// test_ArrayStack.cpp
#include <iostream>
#include "ArrayStack.h"

int main(void) {
  ArrayStack<int> stack;
  stack.push(1);
  stack.push(4);
  stack.push(5);

  while (!stack.isEmpty()) {
    cout << stack.pop() << endl;
  }

  return 0;
}
```