Foreword

In order to do this lab you must have access to the school's virtual machine (see wiki on Moodle), your code from the previous lab.

In this exercise, you will cover the following topics:

- Writing a template function;
- Overloading operators;
- Writing a template class. ***[BONUS]***.

Advice

- Use `man` an especially `man 3` for the development pages ;
- The website `https://en.cppreference.com` is your most valuable friend for the Teaching Unit;

Part I

Create a `smallerThan()` template

1 Writing down the template

In this section, we aim at comparing the size of two trees. To do so, we will write a template function in `utils.h` called `smallerThan()` that takes two objects as parameters.

Instruction

Write down the template that compares two objects and return `True` if the first one is smaller than the second one (see signature below).

```
template<typename T>
bool smallerThan(T a, T b);
```

In your main program, declare two integers `a` and `b`, initialize them and call your template function (see example below).

```
int main(int argc, char** argv) {
  std::cout << "Launching the main program" << std::endl;
  int a = 10;
  int b = 100;
  std::cout << "Is a smaller than b ? " << smallerThan<int>(a,b) << std::endl;

  std::cout << "End of main program - destroying heap objects" << std::endl;
  std::cout << "End of main program - destroying stack objects" << std::endl;
  return 0;
}
```

Question 1

- Do you need to compile `utils.h` ?
- What happens if you replace `a` and `b` with their values when calling smallerThan[a] ?

[a]aka using rvalues

Instruction

In your main program, declare two Pines or two Oaks `a` and `b` and call your template function (see example below).

```
int main(int argc, char** argv) {
  std::cout << "Launching the main program" << std::endl;
  Pine a;
  Pine b;

  std::cout << "a size " << a.size <<" & b size "<< b.size << std::endl;
  std::cout << "Is a smaller than b ? " << smallerThan<Pine>(a,b) << std::endl;

  std::cout << "End of main program - destroying heap objects" << std::endl;
  std::cout << "End of main program - destroying stack objects" << std::endl;
  return 0;
}
```

Question 2

- Can you compile your code ?
- What do you need to add to your `Pine` class ?

2 Overloading the < operator

In order to be able to compare two pines or two oaks we need to overload the < operator for both the classes. Here we simply chose to compare the `size` attribute of the pines (or oaks) and return `true` if one is smaller than the other.

Instruction

Overload the < operator for the `Pine` and `Oak` classes.

Try to compile and run your previous program.

Question 3

- Why are there two calls to the copy constructor of the `Pine` class ?
- What can you do to prevent the copies in your template declaration ?

Instruction

Change the signature of your template to prevent the copies.

Check that there is no more useless objects copies.

Question 4

- What happens now when you try to directly compare two values : `smallerThan<int>(10,20)`

Instruction

In your main program, now declare one `Pine` and one `Oak`.

Question 5

- What happens when you try to compare the `Pine` with the `Oak` using either : `smallerThan<Pine>(p1,o2)` or `smallerThan<Oak>(p1,o2)` ?
- Can you use `smallerThan<Tree>(p1,o2)` directly?
- If not what are the corrections ? Why is it possible in this very specific case[a] ?

[a]Think of what would happen if the $<$ operator was not comparing a member of the parent class

Part II

BONUS - Create a Tree template

Instruction

Change the Tree class so that the `size` attribute can be of any type and not just double.

Question 6

- What happens to the derived `Pine` and `Oak` classes ?
- Would it be interesting to create a `Tree` template whose parameter would be the tree type ? For example `Tree<Pine>` ?

Appendices : Useful commands

```
$man COMMAND               #display the manual page for the given COMMAND
$man 3 FUNCTION            #display the developer manual for the given FUNCTION
$g++                       #GNU project C and C++ compiler
$make                      #GNU make utility to maintain groups of programs
$ldd                       #print shared object dependencies
$strace                    #trace system calls and signals
$strings                   #print the sequences of printable characters in files
$gdb                       #The GNU Debugger
```