C++ lab class 1 - elements of procedural programming (I)

0) Working environment setup

1. [*nix] In the terminal/console window, enter the following commands:

```
$ cd
$ mkdir cpp-lab1
$ cd cpp-lab1
```

2. [*nix] Next enter:

```
$ which gcc
$ which g++
$ which clang
$ which clang++
```

3. [*nix] Now enter:

```
$ gcc --version
$ g++ --version
$ clang --version # only if it has been installed
$ clang++ --version # as above
```

4. ***Exercises***:

    1. Check editors/IDEs that have been installed in the system (e.g. `Qt Creator` , `VS Code` , `Atom` , `Code::Blocks` , `CLion`)
    2. Compare the installed version of `gcc` with its newest release: https://www.gnu.org/software/gcc/releases.html
    3. [*optional*] Familiarise yourself with the following `*nix` commands: `cd` , `mkdir` , `ls` , `pwd` , `touch` , `rm` , `chmod`

1) The minimal `C++` program

1. Create `ex1` - a subdirectory of `cpp-lab1`

```
$ pwd # check, if cpp-lab1 is the current/working directory
$ mkdir ex1
$ cd ex1
```

2. In `ex1` create file `main.cpp` and type into it:

```
int main() {}
```

Hint: in `*nix` to create a file (from the command line) you can use `touch` command (i.e. `touch main.cpp`)

3. In the terminal/console window, enter:

```
$ ls
$ cat main.cpp # verify the content of the file
$ g++ main.cpp # compile the program
$ ls -l # check the content of the working directory
$ ./a.out # run the program
```

4. Next enter:

```
$ rm a.out
$ ls
$ g++ -o ex1 main.cpp # specify the name of the output/executable
$ ls -l
$ ./ex1 # run the program
```

5. ***Exercises***:

    1. Look at the options available in `g++` (*hint*: use command `g++ --help`)
    2. [*optional*] Familiarise yourself with `*nix` `grep` command (e.g. `grep --help`), and use it for searching the help of `g++` , for instance:
        - `g++ --help | grep preprocess`
        - `g++ --help | grep compile`
        - `g++ --help | grep -e "-E "`
        - `g++ --help | grep -e "-S "`
        - `g++ --help | grep -e "-c "`
    3. Familiarise yourself with the process (running program) memory layout; you can read about it, for instance, here:
        - https://cpp.tech-academy.co.uk/memory-layout/
        - https://cpp.tech-academy.co.uk/program-memory/

2) Compilation of a simple multi-file `C++` program

1. In `cpp-lab1` create new directory `ex2`

```
$ pwd # check, if cpp-lab1 is the current/working directory, if not, set it
$ mkdir ex2
$ cd ex2
```

2. In `ex2` create file `main.cpp` (`$ touch main.cpp`) and type into it:

```
#include <iostream>

int main()
{
  std::cout << "Bonjour le monde!" << std::endl;
  return 0;
}
```

[save the file]

3. In the terminal/console window, enter:

```
$ pwd # be sure that it is ex2
$ ls
$ g++ -o ex2 main.cpp
$ ls -l
$ ./ex2
```

4. Next enter:

```
$ g++ -E -o main.pproc main.cpp # -E: only run the preprocessor
$ cat main.pproc
$ head -50 main.pproc # show the first 50 lines of main.pproc
$ tail -20 main.pproc # show the last 20 lines of main.pproc
```

5. Now enter:

```
$ g++ -S -o main.asm main.cpp # -S: only run preprocess and compilati
on steps
$ head -50 main.asm
$ tail -20 main.asm
```

6. After that enter:

```
$ g++ -c -o main.o main.cpp # -c: only run preprocess, compile, and a
ssemble steps
$ cat main.o
$ which objdump
$ objdump -S main.o # if objdump has been installed in the system
```

7. Finally enter:

```
$ g++ -o ex2 main.o # run the linking step / build the executable (fi
le)
$ objdump -S ex2 # if objdump has been installed in the system
```

8. Change the content of `main.cpp` to:

```
#include <iostream>

int add(int x, int y)
{
  return x + y;
}

int main()
{
  std::cout << "add(1,2) = " << add(1,2) << std::endl;
  return 0;
}
```

[save the file]

9. Compile the code and run the program:

```
$ g++ -o ex2 main.cpp
$ ./ex2
```

10. Next change the content of `main.cpp` to:

```
#include <iostream>

int main()
{
  std::cout << "add(1,2) = " << add(1,2) << std::endl;
  return 0;
}

int add(int x, int y)
{
  return x + y;
}
```

[save the file]

11. Try to compile the code:

```
$ g++ -o ex2 main.cpp # note the error description
```

12. Now change the code to:

```
#include <iostream>

int add(int, int); // function 'add' declaration

int main()
{
  std::cout << "add(1,2) = " << add(1,2) << std::endl; // function 'a
dd' call
  return 0;
}

// function 'add' definition
int add(int x, int y)
{
  return x + y;
}
```

Save the file, compile the code (`g++ -o ex2 main.cpp`) and run the program (`./ex2`)

13. Change the structure of the code:

- create a new file (`$ touch add.cpp`)
- cut and paste the definition of function `add` from `main.cpp` to `add.cpp`

After that, the files should have the following content:

```
// main.cpp
#include <iostream>

int add(int, int);

int main()
{
  std::cout << "add(1,2) = " << add(1,2) << std::endl;
  return 0;
}
```

```
// add.cpp
int add(int x, int y)
{
  return x + y;
}
```

Save the files, compile the code (`g++ -o ex2 add.cpp main.cpp`) and run the program (`./ex2`)

14. Change the structure of the code again:

- create a new file (`$ touch add.h`)
- cut and paste the declaration of function `add` from `main.cpp` to `add.h`
- add `#include "add.h"` preprocessor directive to `main.cpp`

After that, the files should have the following content:

```
// main.cpp
#include <iostream>
#include "add.h"

int main()
{
  std::cout << "add(1,2) = " << add(1,2) << std::endl;
  return 0;
}
```

```
// add.h
int add(int, int);
```

```
// add.cpp
int add(int x, int y)
{
  return x + y;
}
```

Save the files, compile the code (`g++ -o ex2 add.cpp main.cpp`) and run the program (`./ex2`)

15. Change the content of `add.h` to:

```
// add.h
#ifndef ADD_H
#define ADD_H

int add(int, int);

#ifdef USE_SQR_MACRO // conditional compilation
#define SQR(n) n * n;
#else
inline int SQR(int n) { return n * n; }
```

```
#endif // USE_SQR_MACRO

#endif // ADD_H
```

and `main.cpp` to:

```
#include <iostream>
#include "add.h"

int main()
{
  std::cout << "add(1,2) = " << add(1,2) << std::endl;
  std::cout << "SQR(2+3) = " << SQR(2+3);
  return 0;
}
```

[save the files]

16. Compile the code and run the program:

```
$ g++ -o ex2 add.cpp main.cpp
$ ./ex2
```

17. Compile the code with `-DUSE_SQR_MACRO` option and run the program:

```
$ g++ -DUSE_SQR_MACRO -o ex2 add.cpp main.cpp
$ ex2 ./ex2
```

Note the difference in the result of `SQR`

18. ***Exercises***:

    1. Fix the bug in the code (macro `SQR` seems to be the problem :)
       Hint: the analysis of the output of the following commands can help:
       - `g++ -E -o main.pproc main.cpp` vs. `g++ -E -DUSE_SQR_MACRO -o main.pproc main.cpp`
       - `g++ -E -dD -o main.pproc main.cpp` vs. `g++ -E -dD -DUSE_SQR_MACRO -o main.pproc main.cpp`

    2. Create a project it in QtCreator and start using it:
       - choose option `not-Qt project->pure C++ application`
       - name it, for instance, `ex2qt`
       - as a build system select `CMake`
       - next continue selecting `next`, and finaly click `finish`
       - copy the code from `main.cpp`, `add.h` and `add.cpp` to the files with the same names but created in the project
       - build the project (click the *hammer* in the bottom left corner), and run the program (click a *green triangle* in the same place)
       - in the left tool bar, click `Projects->Build` and in the table set the CMake key `CMAKE_BUILD_TYPE` to `Debug`, DO NOT forget to click *Apply the changes in configuration*
       - next select `Projects->Run` and unselect option `Run in terminal`
       - set a break point somewhere in the code (for instance in function `add`) and run the code in `debug mode` (the *green triangle with a bug* in the bottom left corner)
       - switch the debugger to *instruction-wise* operation mode (press the first active button in the debugger toolbar; hints can help to find the correct button)
       - find the buttons corresponding to `continue`, `stop`, `step over`, `step into`, `step out`

    3. In file `add.cpp` (of the project ex2qt) add the definition of a global variable `g_int`:

```
// add.cpp
#include "add.h"

int g_int = 3; // initialisation of a variable (declaration -> definition -> initialisation)

int add(int x, int y)
{
   return x + y;
}
```

and print its value in function `main`, i.e.:

```
int main()
{
  //...
  std::cout << "g_int = " << g_int << std::endl;
  return 0;
}
```

Hint: to declare a global variable use keyword `extern`, e.g. `extern double x;`

    4. Familiarise yourself with:
       - preprocessor directives, you can read about it, for instance, here: http://www.cplusplus.com/doc/tutorial/preprocessor/
       - [*optional*] `gdb` https://www.gnu.org/software/gdb/documentation/ and

`lldb` (https://lldb.llvm.org)
- [*optional*] `objdump` (https://linux.die.net/man/1/objdump)

3) Constants, variables and their types

1. In `cpp-lab1` create new directory `ex3`

```
$ pwd # check, if cpp-lab1 is the current/working directory, if not, set it
$ mkdir ex3
```

2. In `Qt Creator` create a new project (`not-Qt project->pure C++ application`):
   - set its location to `...cpp-lab1/ex3`
   - set its name to `ex3qt`
   - select `CMake` as the build system
   - next continue selecting `next` , and finaly click `finish`
   - in the `Projects->Build` set the CMake key `CMAKE_BUILD_TYPE` to `Debug` (do not forget to click *Apply the changes in configuration*)
   - select `Projects->Run` and unselect option `Run in terminal`

3. Change the content of `CMakeList.txt` to:

```
cmake_minimum_required(VERSION 3.1)

project(ex3qt)

set(CMAKE_CXX_STANDARD 14)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

add_executable(${PROJECT_NAME} "main.cpp")
```

4. Run the application (the green triangle in the bottom left corner)

5. Check if the debugger works properly, e.g. by setting a breakpoint in function `main` and running the application in the *debug mode* (the *green triangle with a bug* in the bottom left corner)

6. Change the content of `main.cpp` to:

```
#include <iostream>

using namespace std;

void printSizeOfTypes()
{
  cout << "size of selected C++ types:" << endl;

  cout << "sizeof(int) = " << sizeof (int) << endl;
  cout << "sizeof(int*) = " << sizeof(int*) << endl;
  cout << "sizeof(int&) = " << sizeof(int&) << endl;
  cout << "sizeof(const int) = " << sizeof(const int) << endl;
  cout << "sizeof(const int*) = " << sizeof(const int *) << endl;
  cout << "sizeof(const int&) = " << sizeof(const int&) << endl;

  cout << endl;
}

void printSizeOfVariables()
{
  cout << "size of variables and constant values:" << endl;

  int iv = 1;
  int *piv = &iv; // pointer to int
  int &riv = iv; // reference to T
  int &&rriv = static_cast<int &&>(iv); // rvalue reference to int

  cout << "sizeof(int iv) = " << sizeof(iv) << endl;
  cout << "sizeof(int *piv) = " << sizeof(piv) << endl;
  cout << "sizeof(int &riv) = " << sizeof(riv) << endl;
  cout << "sizeof(int &&rriv) = " << sizeof(rriv) << endl;

  const int ci = 2; // a named constant
  const int *pci = &ci; // pointer to constant int
  const int &rci = ci; // reference to constant int
  const int &&rrci = static_cast<const int &&>(ci); // rvalue referen
ce to constant int

  cout << "sizeof(const int ci) = " << sizeof(ci) << endl;
  cout << "sizeof(const int *pci) = " << sizeof(pci) << endl;
  cout << "sizeof(const int &rci) = " << sizeof(rci) << endl;
  cout << "sizeof(const int &&rrci) = " << sizeof(rrci) << endl;

  constexpr int cexpr = 2; // constant compile-time expression
  cout << "sizeof(cexpr) = " << sizeof(cexpr) << endl;

  int a[5] = {1,2,3,4,5}; // array of 5 ints
  cout << "sizeof(a), the array of 5 int elements = " << sizeof(a) <<
 endl;
}

int main()
```

```
{
  printSizeOfTypes();
  printSizeOfVariables();
  return 0;
}
```

7. Run the application and analyse the results

1. ***Exercises***:

    1. Check/print the sizes of the other types available in `C++` (`C++14`), i.e.: `bool` , `char` , ..., `long long int` , `unsigned char` , ..., `unsigned long long int` `float` , `double` , `long double` , `int8_t, ...,` int64_t, `uint8_t` , ...

    2. Familiarise yourself with function `printf` , you can read about it, for instance, here: http://www.cplusplus.com/reference/cstdio/printf/

    3. [*optional*] Rewrite function `printSizeOfTypes` using function `printf` instead of `cout <<`

    4. Familiarise yourself with operators available in `C++` and their precedence, you can read about it, for instance, here: https://en.cppreference.com/w/cpp/language/operator_precedence