C++ lab class 2 - elements of procedural programming (II)

0) Working environment setup

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

```
$ cd
$ mkdir cpp-lab2
$ cd cpp-lab2
```

1) Scope and lifetime of variables

1. In `cpp-lab2` create sub-directory `ex1` (*hint*: `$ mkdir ex1`) and set it as the working directory (`$ cd ex1`)

2. In `ex1` create file `main.cpp` (*hint*: `$ touch main.cpp`) and type into it:

```
#include <iostream>

// A namespace is a (named) scope
namespace MyLib {
    int add(int x, int y) { return x + y; } // definition of function "add"
    int sub(int x, int y); // declaration of function "sub"
    extern int v; // declaration of variable "x"
}

int MyLib::v = 1;

int MyLib::sub(int x, int y)
{
    return x - y;
}

/*
 * A namespace is open - you can add names to it from several separate
 * namespace declarations (they can also be written in different files)
 */
namespace MyLib {
    int mul(int x, int y) { return x * y; }
    int aToB(int a, int b);
}

int g_counter = 0; // a global variable

// a global function
int aToB(int a, int b)
{
    if (b <= 0) { return 1; }
    else { return a * aToB(a, b - 1); }
}

int MyLib::aToB(int a, int b)
{
    int r = 1; // local variable
    static int sl_aToBCallCounter = 0; // static local variable

    for (int i = 1; i <= b; ++i) { r *= a; }

    sl_aToBCallCounter++;

    { // it is a new scope!
      using namespace std;
      cout << "sl_aToBCallCounter = " << sl_aToBCallCounter << endl;
    }

    return r;
}

using MyLib::add;

int main()
{
    using namespace std; // using-directive: make ALL names from std accessible
    using MyLib::mul; // using-declaration: add a name ("mul") to a local scope
    cout << "add(1,2) = " << add(1,2) << endl;
    cout << "mul(3,5) = " << mul(3,5) << endl;
    cout << "MyLib::sub(5,2) = " << MyLib::sub(5,2) << endl;

    { // it is a new scope!
        using MyLib::aToB;
        cout << "aToB(5,2) = " << aToB(5,2) << endl;
        cout << "aToB(3,3) = " << aToB(3,3) << endl;

        // to access the global function aToB we use "::" prefix
        cout << "::aToB(3,4) = " << ::aToB(3,4) << endl;
    }

    // now we need the fully qualified name (no using MyLib::aToB in
this scope)
    cout << "MyLib::aToB(2,6) = " << MyLib::aToB(2,6) << endl;

    int *p = new int; // new-allocated/dynamic memory/data/object
    *p = 5;
    cout << "*p = " << *p << endl;
    delete p;

    return 0;
}
```

[save the file]

3. Compile and run the program:

```
$ g++ -o ex1 main.cpp
$ ./ex1
```

4. ***Exercises***:

    1. With the help of a debugger (e.g. in *Qt Creator*) analyse the “lifetime” of all variables used in `main.cpp`
    2. Go through the above code (section 2 above) and note in which part of the process memory (i.e. data, stack, heap) each of the variables will be stored/allocated
    3. Explain the difference between *using-directive* and *using-declaration*
    4. [*optional*] Explain how a *namespace* can be used to define a *module*
    5. [*optional*] Compare a *class scope* with the *namespace scope* (*note*: classes will be discussed later in the course)
    6. Compare *local static variable* with *local variable* and with *global variable* (consider *scope*, “*lifetime*“ and *time of initialisation*)

2) C-style arrays, vectors and strings (*short introduction*)

1. In `cpp-lab2` create sub-directory `ex2` (*hint*: `$ mkdir ex2`) and set it as the working directory (`$ cd ex2`)

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

```
#include <iostream>
#include <string>
#include <vector>

int main()
{
  using namespace std;

  char cStyleStr1[] = "Salut?"; // C-style/null-terminated string
  cout << "sizeof(cStyleStr1) = " << sizeof(cStyleStr1) << endl;
  cStyleStr1[5] = '!';
  cout << "cStyleStr1 = " << cStyleStr1 << endl;

  const char* cStyleStr2 = "Salut!";
  cout << "sizeof(cStyleStr2) = " << sizeof(cStyleStr2) << endl;
  // explain why is sizeof(cStyleStr2) != sizeof(cStyleStr1)

  char cStyleStr3[7] = {'S', 'a', 'l', 'u', 't', '!', '\0'};
  char cStyleStr4[] = {'S', 'a', 'l', 'u', 't', '!'};
  cout << "cStyleStr3 = " << cStyleStr3
       << " and its size = " << sizeof(cStyleStr3) << endl;
  cout << "cStyleStr4 = " << cStyleStr4
       << " and its size = " << sizeof(cStyleStr4) << endl;

  string myName {"Luka"}; // or myName("Luka"), myName = "Luka"
  string s1 = "bon";
  s1 += s1; // s1 = s1 + s1
  cout << "s1 = " << s1 << " and its length is = " << s1.length() <<
endl;

  int a1[3];
  a1[0] = 1;
  a1[1] = 2;
  a1[2] = 3;

  int a2[] = {1, 2, 3}; // or int a3[] {1, 2, 3}

  vector<int> v1 {1,2,3};
  v1.emplace_back(4);
  cout << "v1[3] = " << v1[3] << ", v1.length() = " << v1.size() << e
ndl;
  v1.push_back(5);
  cout << "v1[4] = " << v1[4] << ", v1.length() = " << v1.size() << e
ndl;
  v1.pop_back();
  cout << "v1[3] = " << v1[4] << ", v1.length() = " << v1.size() << e
ndl;

  return 0;
}
```

[save the file]

3. Compile and run the program:

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

4. ***Exercises***:

    1. Familiarise yourself with type/class `string` , you can read about it, for instance, here: http://www.cplusplus.com/reference/string/string/
    2. Familiarise yourself with type/class `vector` , you can read about it, for instance, here: http://www.cplusplus.com/reference/vector/vector/
    3. Compare properties/functionality of c-style strings and `C++` strings
    4. Compare properties/functionality of c-style arrays and `C++` vectors

3) Type inference and type aliases

1. In `cpp-lab2` create sub-directory `ex3` (*hint*: `$ mkdir ex3`) and set it as the working directory (`$ cd ex3`)

2. In `ex3` create file `main.cpp` (*hint*: `$ touch main.cpp`) and type into it:

```
#include <iostream>

int main()
{
  using namespace std;
  auto x1 = 1; // int x1 = 1;
  auto x2 = 2uL; // unsigned long x2 = 2;
  auto x3 = 2.71; // double x3 = 2.71;
  auto x4 = 3.14f; // float x4 = 3.14;

  decltype (x3) x5 = 1.23; // double x5 = 1.23;
  using t1 = decltype (x2);
  t1 x6 = 5; // unsigned long x6 = 5

  typedef unsigned long t2;
  t2 x7 = 6;

  cout << "sizeof(x1) = " << sizeof(x1) << endl;
  cout << "sizeof(x2) = " << sizeof(x2) << endl;
  cout << "sizeof(x3) = " << sizeof(x3) << endl;
  cout << "sizeof(x4) = " << sizeof(x4) << endl;
  cout << "sizeof(x5) = " << sizeof(x5) << endl;
  cout << "sizeof(x6) = " << sizeof(x6) << endl;
  cout << "sizeof(x7) = " << sizeof(x7) << endl;
```

```
}
```

[save the file]

3. Compile and run the program:

```
$ g++ -o ex3 main.cpp
$ ./ex3
```

4. ***Exercises***:

    1. Check the result of type inference in the case of constant values, i.e.

    ```
    const int i = 3;
    auto j = i; // is j a constant value or not?
    ```

    2. Consider advantages and disadvantages of using `auto` keyword (and so the type inference mechanism) instead of specifying the type explicitly

4) Control structures 1: `if-then-else` and `switch`

1. In `cpp-lab2` create sub-directory `ex4` (*hint*: `$ mkdir ex4`) and set it as the working directory (`$ cd ex4`)

2. In `ex4` create file `main.cpp` (*hint*: `$ touch main.cpp`) and type into it:

```
#include <iostream>

using namespace std;

void fun1()
{
  float f1 = 0.1f, f2 = 0.3f, f3 = 0.4f;
  float f4 = (f1 + f2) + f3;
  float f5 = f1 + (f2 + f3);

  if (f4 == f5) {
      cout << "f4 == f5" << endl;
  } else {
      cout << "f4 != f5" << endl;
  }
}

int absInt(int x)
{
  return (x >= 0) ? x : -x;
```

```
}

void switchDemo(char c)
{
  switch (c) {
  case 'y':
     cout << "'y' matched" << endl;
     break;
  case 'n':
     cout << "'n' matched" << endl;
     break;
  case 'c':
     cout << "'c' matched" << endl;
     break;
  default:
     cout << "Default case, please select 'y','n' or 'c'" << endl;
  }
}

int main()
{
  fun1();
  switchDemo('y');
  cout << "absInt(-3) = " << absInt(-3) << endl;

  return 0;
}
```

[save the file]

3. Compile and run the program:

```
$ g++ -o ex4 main.cpp
$ ./ex4
```

4. ***Exercises***:

    1. Explain why `f4 != f5` (in function `fun1`)
    2. Compare `if-then-else` instruction with the conditional expression/ternary operator (`?:`)
    3. Change the code of `switchDemo` to:

```
void switchDemo2(char c)
{
  switch (c) {
  case 'y':
     cout << "'y' matched" << endl;
  case 'n':
     cout << "'n' matched" << endl;
  case 'c':
     cout << "'c' matched" << endl;
  default:
     cout << "Default case, please select 'y','n' or 'c'" << end
l;
  }
}
```

and test it for different arguments; explain the results

5) Control structures 2: *loops*

1. In `cpp-lab2` create sub-directory `ex5` (*hint*: `$ mkdir ex5`) and set it as the working directory (`$ cd ex5`)

2. In `ex5` create file `main.cpp` (*hint*: `$ touch main.cpp`) and type into it:

```
#include <iostream>
#include <vector>

using std::vector;

int sum1(int elems[], size_t size)
{
  int sum = 0;
  size_t i = 0;

  while (i < size) {
    sum += elems[i];
    ++i;
  }

  return sum;
}

int sum2(int elems[], size_t size)
{
 int sum = 0;

 if (size > 0) {
   size_t i = 0;
   do {
     sum += elems[i];
     ++i;
   } while (i < size);
 }

 return sum;
}

int sum3(int elems[], size_t size)
{
  int sum = 0;

  for (size_t i = 0; i < size; ++i) {
    sum += elems[i];
  }

  return sum;
}

int sum4(const vector<int> &elems)
{
  int sum = 0;

  for (auto e : elems) {
    sum += e;
  }

  return sum;
}

int sumEven(int elems[], size_t size)
{
  int sum = 0;

  for (size_t i = 0; i < size; ++i) {
    if (elems[i] % 2 == 1) {
      // the continue statement 'jumps over' one iteration in the loo
p
      continue;
    }
    sum += elems[i];
  }

  return sum;
}

int indexOfM(int m, int elems[], size_t size)
{
  int idxOfM = -1;

  for (size_t i = 0; i < size; ++i) {
    if (elems[i] == m) {
      idxOfM = static_cast<int>(i);
      break; // the break statement 'jumps out' of a loop
    }
  }

  return idxOfM;
}

int main()
{
  using namespace std;

  int a1[] = {0,1,2,3,4};
  vector<int> v1{0,1,2,3,4};
  constexpr size_t a1Size = sizeof(a1) / sizeof (a1[0]);

  vector<int> sums = {
     sum1(a1, a1Size),
     sum2(a1, a1Size),
     sum3(a1, a1Size),
     sum4(v1)
  };

  cout << "sums for a1/v1 = [0,1,2,3,4]: " << endl;
  for (size_t i = 0; i < sums.size(); ++i) {
     cout << "sum" << i + 1 << " = " << sums[i] << endl;
  }

  cout << "sumEven of a1 = [0,1,2,3,4] = " << sumEven(a1, a1Size) <<
endl;

  for (int i = 0; i < 6; ++i) {
    cout << "indexOfM(" << i << ", [0,1,2,3,4], 5) = "
         << indexOfM(i, a1, a1Size) << endl;
  }

  return 0;
}
```

[save the file]

3. Compile and run the program:

```
$ g++ -o ex5 main.cpp
$ ./ex5
```

4. ***Exercises***:

   1. Test the following function:

      ```
      float fun2()
      {
         float x = 0.0f;
         while (x != 1.0f) {
           x += 0.1f;
         }
         return x;
      }
      ```

      and explain the result

   2. Write functions to calculate the arithmetic mean of a given array (name it, for instance, `meanOfA`) and of a given vector (name it, for instance, `meanOfV`)

   3. [*optional*] Write functions to calculate the standard deviation of a given array (name it, for instance, `stdOfA`) and of a given vector (name it, for instance, `stdOfV`)

   4. Write functions to calculate the sum of odd elements of a given array (name it, for instance, `sumOddsOfA`) and of a given vector (name it, for instance, `sumOddsOfV`)

6) Simple recursive functions

1. In `cpp-lab2` create sub-directory `ex6` (*hint*: `$ mkdir ex6`) and set it as the working directory (`$ cd ex6`)

2. In `ex6` create file `main.cpp` (*hint*: `$ touch main.cpp`) and type into it:

```
#include <iostream>
#include <vector>
#include <chrono>

int sum5(int elems[], size_t size)
{
  if (size == 0) { return 0; }
  else { return sum5(elems, size - 1) + elems[size - 1]; }
}

long factR(int n)
{
  if (n <= 1) { return 1; }
  else { return n * factR(n - 1); }
}

int fibR(int n)
{
  if (n <= 0) { return 0;}
  else if (n == 1) { return 1; }
  else { return fibR(n - 2) + fibR(n - 1); }
}

int main()
{
  using namespace std;

  int a1[] = {0,1,2,3,4};
  constexpr size_t a1Size = sizeof(a1) / sizeof (a1[0]);

  cout << "factR(20) = " << factR(20) << endl;
  cout << "sum5(10) = " << sum5(a1, a1Size) << endl;

  std::chrono::steady_clock::time_point start, end;

  int n4Fib = 45;
  start = std::chrono::steady_clock::now();
  fibR(n4Fib);
  end = std::chrono::steady_clock::now();

  cout << "fibR(" << n4Fib << ") took "
       << std::chrono::duration_cast<std::chrono::microseconds>(end -
 start).count()
       << "us.\n";
}
```

[save the file]

3. Compile and run the program:

```
$ g++ -o ex6 main.cpp
$ ./ex6
```

4. ***Exercises***:

1. Write and test an iterative version (i.e. using a loop) of `factR`
2. Write and test an iterative version (i.e. using a loop) of `fibR` ; compare the execution times of these two versions
3. Write a recursive function that finds the maximal element in a given vector