Universidad Zaragoza
1542

SMART POINTERS

Fundamentals on Computing for Robotics, Graphics and Computer Vision

Prerequisites

In this laboratory, you are going to write several small programs covering the basic aspects of smart pointers. Please use the same compiler and setup that in previous laboratories. Please do not forget to use these CXXFLAGS when compiling with g++: `-Wall -Wextra -pedantic -Werror`. These flags would help you to write more solid code. Ideally, you should use `cmake` as your building tool. Also, you could consider adding the following lines to your `CMakeLists.txt` to ensure you are working on C++17.

```
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
```

Do not hesitate to contact the faculty in case of doubts about the exercises.

Identify memory leaks and remove them

Identify all the memory leaks in the file `to_fix_memory_leaks.cpp` and avoid them using smart pointers. This example only requires `std::unique_ptr`.

Please understand and run the original program before starting your solution.

```
#include <iostream>


struct point
{
  int x, y;
  bool not_zero() const { return x!=0 || y!=0; };
};

struct point_node
{
  point p;
  point_node* next;

  point_node(const point& other_p, point_node* other_next) : p(other_p), ne
};

point* copy_point(const point& p)
{
  point* point_ptr = new point(p);

  if (p.x > 1)
    return nullptr;
```

```
  return point_ptr;
}

int main()
{
  const size_t n = 4;
  point points[n]={ {0, 0}, {1, 1}, {0, 0}, {1, 1} };

  // save points whose coordinates are not (0, 0)
  point_node* list = nullptr;

  for(const auto& p: points) {
    point_node* node = new point_node(p, nullptr);
    if(p.not_zero()) {
      if(list == nullptr) {
        list = node;
      } else {
        point_node* iter=list;
        while(iter->next != nullptr) {
          iter = iter->next;
        }
        iter->next = node;
      }
    }
  }

  // print non-zero nodes
  for(point_node* iter = list; iter != nullptr; iter = iter->next) {
    std::cout << "(x, y): " << iter->p.x << ", " << iter->p.y << std::endl;
  }
}
```

Using shared pointers

Please describe what the following program does and answer the following questions:

1. How many objects (variables) are allocated in the stack and in the heap?
2. Can both `sp1` and `sp2` modify the object in the heap?
3. Does the destruction of `sp2` releases the memory allocated in the heap?

```
#include <iostream>
#include <memory>

int main() {

  auto sp1 = std::make_shared<int>(1);
  std::cout << "sp1.use_count(): " << sp1.use_count() << std::endl;

  { // new scope
    std::shared_ptr<int> sp2(sp1);
    std::cout << "sp1.use_count(): " << sp1.use_count() << std::endl;
    std::cout << "sp2.use_count(): " << sp2.use_count() << std::endl;
  }
```

```
  std::cout << "sp1.use_count(): " << sp1.use_count() << std::endl;
  std::cout << "Is sp1 unique? " << sp1.unique() << std::endl;
}
```

Writing simple code with smart pointers

Starting from the `max_min_heap_2d_matrix.cpp` code, replace all dynamic memory operations with smart pointers. The resulting file should be named `max_min_sp_2d_matrix.cpp`. The maximum and minimum values should be printed by reading the value from an smart pointer as well.

Submission

Please group the all the files within a zip. In those assignments with text questions, add then at the beginning of the file as C++ comments.

Before submitting, remember to talk to your teacher and have a **small interview** describing and answering questions about the job you have done and the decissions you have made. This interview is *20%* of the total grade of the assignment, while the submitted material is evaluated for the other *80%*.

All source code files, must include a comment on top with the names and NIAs of the students involved on their development. Then, all source code files, including directory structure, must be compressed into a single `zip` file with the following naming:

- **`smart_pointers_<nia1>_<nia2>.zip`**, where `<nia1>` and `<nia2>` are the NIAs of the involved students, if the work has been done in pairs. In this case, **only one** of the students must submit the work in Moodle.
- **`smart_pointers_<nia>.zip`**, where `<nia>` is the NIA of the involved student if the work has been done individually.

The compressed file must not contain anything else besides the source code files. Particularly, do not submit any binary file, neither executable, library nor object. The submission of the `zip` file must be done through the corresponding task in Moodle before the stablished deadline.