Poznan University of Technology

Object Oriented Programming
RAII

Jakub Piotr Hamerliński, M.Eng.

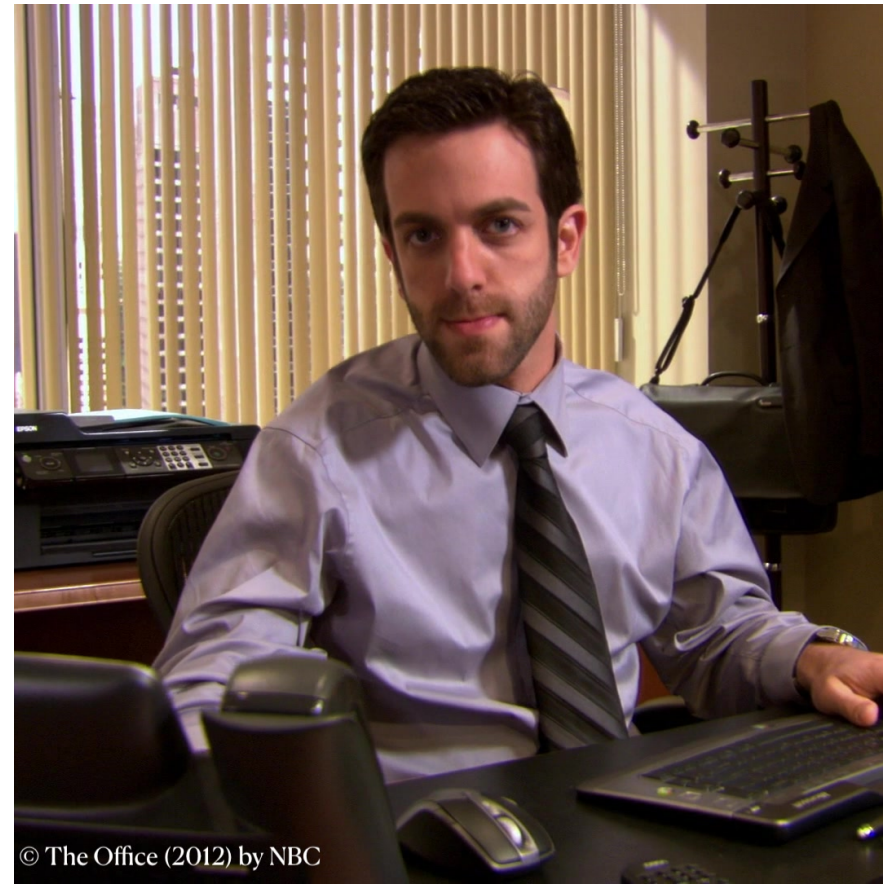
Object Oriented Programming

Agenda

01. RAII
02. Task

> "I'd like to make a toast. To the troops...all the troops... both sides"

Ryan Howard



RAII

Object Oriented Programming

RAII

Unlike managed languages, C++ doesn't have automatic garbage collection, an internal process that releases heap memory and other resources as a program runs. A C++ program is responsible for returning all acquired resources to the operating system. Failure to release an unused resource is called a leak. Leaked resources are unavailable to other programs until the process exits. Memory leaks in particular are a common cause of bugs in C-style programming.

Object Oriented Programming

RAII

Modern C++ avoids using heap memory as much as possible by declaring objects on the stack. When a resource is too large for the stack, then it should be owned by an object. As the object gets initialized, it acquires the resource it owns. The object is then responsible for releasing the resource in its destructor. The owning object itself is declared on the stack. The principle that objects own resources is also known as "resource acquisition is initialization," or RAII.

Object Oriented Programming

RAII

RAII (Resource Acquisition Is Initialization) is a programming technique in C++ where resources (memory, file handles, sockets, etc.) are acquired during the initialization of an object and released automatically when the object is destroyed. This technique helps to manage resources, prevent resource leaks, and simplify exception handling.

RAII is used in C++ by creating a class that encapsulates the resource, acquires the resource in the constructor, and releases it in the destructor. The object of the class is typically created on the stack so that its lifetime is bound to the scope it's created in.

Object Oriented Programming

RAII

Example

```cpp
#include <iostream>
#include <stdexcept>
#include <mutex>
#include <thread>
// Permit class is responsible for automatically locking and unlocking the mutex.
class Permit {
 public:
  // Constructor takes a reference to a mutex and locks it.
  explicit Permit(std::mutex& mtx) : lock_(mtx) {
    // Mutex is locked automatically by the std::unique_lock constructor.
  }
  // Destructor of Permit will be called when the object goes out of scope.
  ~Permit() {
    // Mutex is automatically unlocked when std::unique_lock destructor is called.
  }
 private:
  std::unique_lock<std::mutex> lock_; // RAII wrapper for the mutex.
};
```

Object Oriented Programming

RAII

Example

```
class Foo {
 public:
  // Print method is responsible for demonstrating the RAII usage in conjunction with the Permit class.
  void Print(int x) {
    Permit permit(mutex_); // Permit object is created, locking the mutex.
    // When an exception is thrown or the function scope is exited,
    // the Permit object's destructor will be called, unlocking the mutex.
    try {
      if (x > 1000) {
        throw std::runtime_error("Too large!");
      }
      std::cout << "x = " << x << std::endl;
    } catch (const std::runtime_error& e) {
      std::cerr << "Error: " << e.what() << std::endl;
    }
  }
 private:
  std::mutex mutex_; // Mutex to synchronize access to the shared resource (console output).
};
```

Object Oriented Programming

RAII

Example

```
int main() {
  Foo foo;
  std::vector<std::thread> threads;
  // Launching multiple threads to demonstrate synchronization using RAII.
  for (int i = 0; i < 10; ++i) {
    threads.emplace_back([&foo, i]() { foo.Print(i * 100); });
  }
  // Waiting for all threads to finish.
  for (auto& t : threads) {
    t.join();
  }
  return 0;
}
```

Task

Object Oriented Programming

Task

Implement a simple FASTA file reader using RAII

FASTA is a text-based format used to represent nucleotide or protein sequences. A FASTA file consists of one or more sequences, each starting with a description line that begins with a '>' character, followed by the sequence itself on subsequent lines. The sequence lines can have varying lengths. The goal of this task is to create a simple FASTA file reader that utilizes RAII for resource management.

Object Oriented Programming

Task

Implement a simple FASTA file reader using RAII

Create a class Fasta that reads a FASTA file and stores the description and sequence for each entry in the file. Implement the class using RAII principles to manage the file resource. The Fasta constructor should take the filename as an argument, open the file, and read the contents into memory. The destructor of Fasta should close the file if it is still open. Add a method Sequences that returns a vector of pairs, where each pair consists of a description and a sequence.

Object Oriented Programming

Task

Implement a simple FASTA file reader using RAII

Hints:

01. Use the `<fstream>` library to handle file input.
02. Remember to close the file in the destructor.
03. Use a std::vector to store the sequences and their descriptions.
04. After implementing the Fasta class, write a simple program that utilizes the class to read a FASTA file and print its contents.

Thank you

Feel free to reach me via LinkedIn

Fin