30-raii-reminders

June 2, 2024

1 Pointers & dynamic memory management

Scott Meyers: *A resource is something that, once you're done using it, you need to return to the system. If you don't, bad things happen. In C++ programs, the most commonly used resource is dynamically allocated memory (if you allocate memory and never deallocate it, you've got a memory leak), but memory is only one of many resources you must manage. Other common resources include file descriptors, mutex locks, fonts and brushes in graphical user interfaces (GUIs), database connections, and network sockets. Regardless of the resource, it's important that it be released when you're finished with it. Trying to ensure this by hand is difficult under any conditions, but when you consider exceptions, functions with multiple return paths, and maintenance programmers modifying software without fully comprehending the impact of their changes, it becomes clear that ad hoc ways of dealing with resource management aren't sufficient.*

1.1 Motivation

Have a look at this toy library for particles.

[1]:
```
%%file tmp.particles.h

class Particle {
  public :
    virtual ~Particle() {}
    virtual char const * name() const =0 ;
} ;

class Electron : public Particle {
  public :
    virtual char const * name() const
     { return "Electron" ; }
} ;

class Proton : public Particle {
  public :
    virtual char const * name() const
     { return "Proton" ; }
} ;

class Neutron : public Particle {
  public :
    virtual char const * name() const
     { return "Neutron" ; }
} ;

Particle * new_particle() {
  int const NB = 3 ;
  int num = NB*(std::rand()/(double(RAND_MAX)+1)) ;
  switch (num)
   {
    case 0: return new Electron() ;
    case 1: return new Proton() ;
    case 2: return new Neutron() ;
   }
  return 0 ;
}
```

```
Overwriting tmp.particles.h
```

[2]:

```
%%file tmp.raii.cpp

#include "tmp.particles.h"
#include <iostream>
#include <ctime>

void print( Particle const * p ) {
  std::cout<<p->name()<<std::endl ;
}

int main() {
  std::srand(std::time(0)) ;
  for ( int i = 0 ; i < 5 ; ++i ) {
    Particle * p = new_particle() ;
    print(p) ;
    delete p ;
  }
}
```

```
Overwriting tmp.raii.cpp
```

[3]:

```
!rm -f tmp.raii.exe && g++ -std=c++17 tmp.raii.cpp -o tmp.raii.exe
```

```
In file included from tmp.raii.cpp:2:
tmp.particles.h: In function ‘Particle* new_particle()’:
tmp.particles.h:28:22: error: ‘rand’ is not a member of ‘std’
   28 |   int num = NB*(std::rand()/(double(RAND_MAX)+1)) ;
      |                      ^~~~
tmp.particles.h:28:37: error: ‘RAND_MAX’ was not declared in this scope
   28 |   int num = NB*(std::rand()/(double(RAND_MAX)+1)) ;
```

```
      |                                     ^~~~~~~~
tmp.particles.h:28:37: note: the macro ‘RAND_MAX’ had not yet been defined
In file included from /usr/local/include/c++/13.2.0/cstdlib:79,
                 from /usr/local/include/c++/13.2.0/ext/string_conversions.h:43,
                 from /usr/local/include/c++/13.2.0/bits/basic_string.h:4097,
                 from /usr/local/include/c++/13.2.0/string:54,
                 from /usr/local/include/c++/13.2.0/bits/locale_classes.h:40,
                 from /usr/local/include/c++/13.2.0/bits/ios_base.h:41,
                 from /usr/local/include/c++/13.2.0/ios:44,
                 from /usr/local/include/c++/13.2.0/ostream:40,
                 from /usr/local/include/c++/13.2.0/iostream:41,
                 from tmp.raii.cpp:3:
/usr/include/stdlib.h:87: note: it was later defined here
   87 | #define RAND_MAX        2147483647
      |
```

[4]:
```
!./tmp.raii.exe
```

```
sh: 1: ./tmp.raii.exe: not found
```

In real-life long functions, the `delete` can be bypassed, due to: * a premature call to `return`, `break`, `continue`… ; * an exception is raised.

1.2 Solution : RAII idiom

In order to make sure that a *resource* is released at the end of its utilization, one can wrap this resource into a *guard* object, whose **destructor** will release the resource. There is no more possible bypass, because whenever one leave a function, all local objects are **always destructed**.

[5]:
```
%%file tmp.guards.h

class ParticleGuard {
  public :
    explicit ParticleGuard( Particle * p ) : m_p(p) {}
    ~ParticleGuard() { delete m_p ; }
  private :
    Particle * m_p ;
} ;
```

```
Overwriting tmp.guards.h
```

[6]:
```
%%file tmp.raii.cpp

#include "tmp.particles.h"
#include "tmp.guards.h"
#include <iostream>
#include <ctime>

void print( Particle const * p ) {
```

```
  std::cout<<p->name()<<std::endl ;
}

int main() {
  std::srand(std::time(0)) ;
  for ( int i = 0 ; i < 5 ; ++i ) {
    Particle * p = new_particle() ;
    ParticleGuard pg(p) ;
    print(p) ;
  }
}
```

```
Overwriting tmp.raii.cpp
```

[7]:
```
!rm -f tmp.raii.exe && g++ -std=c++17 tmp.raii.cpp -o tmp.raii.exe
```

```
In file included from tmp.raii.cpp:2:
tmp.particles.h: In function ‘Particle* new_particle()’:
tmp.particles.h:28:22: error: ‘rand’ is not a member of ‘std’
   28 |   int num = NB*(std::rand()/(double(RAND_MAX)+1)) ;
      |                      ^~~~
tmp.particles.h:28:37: error: ‘RAND_MAX’ was not declared in this scope
   28 |   int num = NB*(std::rand()/(double(RAND_MAX)+1)) ;
      |                                     ^~~~~~~~
tmp.particles.h:28:37: note: the macro ‘RAND_MAX’ had not yet been defined
In file included from /usr/local/include/c++/13.2.0/cstdlib:79,
                 from /usr/local/include/c++/13.2.0/ext/string_conversions.h:43,
                 from /usr/local/include/c++/13.2.0/bits/basic_string.h:4097,
                 from /usr/local/include/c++/13.2.0/string:54,
                 from /usr/local/include/c++/13.2.0/bits/locale_classes.h:40,
                 from /usr/local/include/c++/13.2.0/bits/ios_base.h:41,
                 from /usr/local/include/c++/13.2.0/ios:44,
                 from /usr/local/include/c++/13.2.0/ostream:40,
                 from /usr/local/include/c++/13.2.0/iostream:41,
                 from tmp.raii.cpp:4:
/usr/include/stdlib.h:87: note: it was later defined here
   87 | #define RAND_MAX        2147483647
      |
```

[8]:
```
!./tmp.raii.exe
```

```
sh: 1: ./tmp.raii.exe: not found
```

BEWARE: one should not create multiple guards for a single resource, or this resource will be released several times…

[9]:
```
%%file tmp.raii.cpp

#include "tmp.particles.h"
```

```
#include "tmp.guards.h"
#include <iostream>
#include <ctime>

void print( Particle const * p ) {
  std::cout<<p->name()<<std::endl ;
}

int main() {
  std::srand(std::time(0)) ;
  for ( int i = 0 ; i < 5 ; ++i ) {
    Particle * p = new_particle() ;
    ParticleGuard pg1(p) ;
    ParticleGuard pg2(p) ;
    print(p) ;
  }
}
```

```
Overwriting tmp.raii.cpp
```

[10]:
```
!rm -f tmp.raii.exe && g++ -std=c++17 tmp.raii.cpp -o tmp.raii.exe
```

```
In file included from tmp.raii.cpp:2:
tmp.particles.h: In function ‘Particle* new_particle()’:
tmp.particles.h:28:22: error: ‘rand’ is not a member of ‘std’
   28 |   int num = NB*(std::rand()/(double(RAND_MAX)+1)) ;
      |                      ^~~~
tmp.particles.h:28:37: error: ‘RAND_MAX’ was not declared in this scope
   28 |   int num = NB*(std::rand()/(double(RAND_MAX)+1)) ;
      |                                     ^~~~~~~~
tmp.particles.h:28:37: note: the macro ‘RAND_MAX’ had not yet been defined
In file included from /usr/local/include/c++/13.2.0/cstdlib:79,
                 from /usr/local/include/c++/13.2.0/ext/string_conversions.h:43,
                 from /usr/local/include/c++/13.2.0/bits/basic_string.h:4097,
                 from /usr/local/include/c++/13.2.0/string:54,
                 from /usr/local/include/c++/13.2.0/bits/locale_classes.h:40,
                 from /usr/local/include/c++/13.2.0/bits/ios_base.h:41,
                 from /usr/local/include/c++/13.2.0/ios:44,
                 from /usr/local/include/c++/13.2.0/ostream:40,
                 from /usr/local/include/c++/13.2.0/iostream:41,
                 from tmp.raii.cpp:4:
/usr/include/stdlib.h:87: note: it was later defined here
   87 | #define RAND_MAX        2147483647
      |
```

[11]:
```
!./tmp.raii.exe
```

```
sh: 1: ./tmp.raii.exe: not found
```

1.3 Access to the resource

To avoid this type of errors, the resource should be directly given to the guard... but we cannot access it any more ! Hence the need for some way to retreive the resource from the guard:

[12]:
```
%%file tmp.guards.h

class ParticleGuard {
  public :
    explicit ParticleGuard( Particle * p ) : m_p(p) {}
    ~ParticleGuard() { delete m_p ; }
    Particle * get() { return m_p ; }
  private :
    Particle * m_p ;
} ;
```

```
Overwriting tmp.guards.h
```

[13]:
```
%%file tmp.raii.cpp

#include "tmp.particles.h"
#include "tmp.guards.h"
#include <iostream>
#include <ctime>

void print( Particle const * p ) {
  std::cout<<p->name()<<std::endl ;
}

int main() {
  std::srand(std::time(0)) ;
  for ( int i = 0 ; i < 5 ; ++i ) {
    ParticleGuard pg(new_particle()) ;
    print(pg.get()) ;
  }
}
```

```
Overwriting tmp.raii.cpp
```

[14]:
```
!rm -f tmp.raii.exe && g++ -std=c++17 tmp.raii.cpp -o tmp.raii.exe
```

```
In file included from tmp.raii.cpp:2:
tmp.particles.h: In function ‘Particle* new_particle()’:
tmp.particles.h:28:22: error: ‘rand’ is not a member of ‘std’
   28 |   int num = NB*(std::rand()/(double(RAND_MAX)+1)) ;
      |                      ^~~~
tmp.particles.h:28:37: error: ‘RAND_MAX’ was not declared in this scope
   28 |   int num = NB*(std::rand()/(double(RAND_MAX)+1)) ;
      |                                     ^~~~~~~~
tmp.particles.h:28:37: note: the macro ‘RAND_MAX’ had not yet been defined
```

```
In file included from /usr/local/include/c++/13.2.0/cstdlib:79,
                 from /usr/local/include/c++/13.2.0/ext/string_conversions.h:43,
                 from /usr/local/include/c++/13.2.0/bits/basic_string.h:4097,
                 from /usr/local/include/c++/13.2.0/string:54,
                 from /usr/local/include/c++/13.2.0/bits/locale_classes.h:40,
                 from /usr/local/include/c++/13.2.0/bits/ios_base.h:41,
                 from /usr/local/include/c++/13.2.0/ios:44,
                 from /usr/local/include/c++/13.2.0/ostream:40,
                 from /usr/local/include/c++/13.2.0/iostream:41,
                 from tmp.raii.cpp:4:
/usr/include/stdlib.h:87: note: it was later defined here
   87 | #define RAND_MAX        2147483647
      |
```

[15]:
```
!./tmp.raii.exe
```

```
sh: 1: ./tmp.raii.exe: not found
```

One may be tempted to improve the guard with some conversion operator.

[16]:
```
%%file tmp.guards.h

class ParticleGuard {
  public :
    explicit ParticleGuard( Particle * p ) : m_p(p) {}
    ~ParticleGuard() { delete m_p ; }
    operator Particle *() { return m_p ; }
  private :
    Particle * m_p ;
} ;
```

```
Overwriting tmp.guards.h
```

[17]:
```
%%file tmp.raii.cpp

#include "tmp.particles.h"
#include "tmp.guards.h"
#include <iostream>
#include <ctime>

void print( Particle const * p ) {
  std::cout<<p->name()<<std::endl ;
}

int main() {
  std::srand(std::time(0)) ;
  for ( int i = 0 ; i < 5 ; ++i ) {
    ParticleGuard p(new_particle()) ;
    print(p) ;
  }
}
```

```
Overwriting tmp.raii.cpp
```

[18]:
```
!rm -f tmp.raii.exe && g++ -std=c++17 tmp.raii.cpp -o tmp.raii.exe
```

```
In file included from tmp.raii.cpp:2:
tmp.particles.h: In function ‘Particle* new_particle()’:
tmp.particles.h:28:22: error: ‘rand’ is not a member of ‘std’
   28 |   int num = NB*(std::rand()/(double(RAND_MAX)+1)) ;
      |                      ^~~~
tmp.particles.h:28:37: error: ‘RAND_MAX’ was not declared in this scope
   28 |   int num = NB*(std::rand()/(double(RAND_MAX)+1)) ;
      |                                     ^~~~~~~~
tmp.particles.h:28:37: note: the macro ‘RAND_MAX’ had not yet been defined
In file included from /usr/local/include/c++/13.2.0/cstdlib:79,
                 from /usr/local/include/c++/13.2.0/ext/string_conversions.h:43,
                 from /usr/local/include/c++/13.2.0/bits/basic_string.h:4097,
                 from /usr/local/include/c++/13.2.0/string:54,
                 from /usr/local/include/c++/13.2.0/bits/locale_classes.h:40,
                 from /usr/local/include/c++/13.2.0/bits/ios_base.h:41,
                 from /usr/local/include/c++/13.2.0/ios:44,
                 from /usr/local/include/c++/13.2.0/ostream:40,
                 from /usr/local/include/c++/13.2.0/iostream:41,
                 from tmp.raii.cpp:4:
/usr/include/stdlib.h:87: note: it was later defined here
   87 | #define RAND_MAX        2147483647
      |
```

[19]:
```
!./tmp.raii.exe
```

```
sh: 1: ./tmp.raii.exe: not found
```

BEWARE: in real-life long functions, this opens the door to tricky errors, when a guard is implicitly and invisibly transformed into its resource. That's why many library designers prefer some explicit `get()` function.

1.4 How to safely copy the guards ?

What happens if we copy a guard ? This code will crash:

[20]:
```
%%file tmp.guards.h

class ParticleGuard {
  public :
    explicit ParticleGuard( Particle * p ) : m_p(p) {}
    ~ParticleGuard() { delete m_p ; }
    Particle * get() { return m_p ; }
```

```
  private :
    Particle * m_p ;
} ;
```

```
Overwriting tmp.guards.h
```

[21]:
```
%%file tmp.raii.cpp

#include "tmp.particles.h"
#include "tmp.guards.h"
#include <iostream>
#include <ctime>

void print( ParticleGuard pg ){
  std::cout<<pg.get()->name()<<std::endl ;
}

int main() {
  std::srand(std::time(0)) ;
  for ( int i = 0 ; i < 5 ; ++i ) {
    ParticleGuard pg(new_particle()) ;
    print(pg) ;
  }
}
```

```
Overwriting tmp.raii.cpp
```

[22]:
```
!rm -f tmp.raii.exe && g++ -std=c++17 tmp.raii.cpp -o tmp.raii.exe
```

```
In file included from tmp.raii.cpp:2:
tmp.particles.h: In function ‘Particle* new_particle()’:
tmp.particles.h:28:22: error: ‘rand’ is not a member of ‘std’
   28 |   int num = NB*(std::rand()/(double(RAND_MAX)+1)) ;
      |                      ^~~~
tmp.particles.h:28:37: error: ‘RAND_MAX’ was not declared in this scope
   28 |   int num = NB*(std::rand()/(double(RAND_MAX)+1)) ;
      |                                     ^~~~~~~~
tmp.particles.h:28:37: note: the macro ‘RAND_MAX’ had not yet been defined
In file included from /usr/local/include/c++/13.2.0/cstdlib:79,
                 from /usr/local/include/c++/13.2.0/ext/string_conversions.h:43,
                 from /usr/local/include/c++/13.2.0/bits/basic_string.h:4097,
                 from /usr/local/include/c++/13.2.0/string:54,
                 from /usr/local/include/c++/13.2.0/bits/locale_classes.h:40,
                 from /usr/local/include/c++/13.2.0/bits/ios_base.h:41,
                 from /usr/local/include/c++/13.2.0/ios:44,
                 from /usr/local/include/c++/13.2.0/ostream:40,
                 from /usr/local/include/c++/13.2.0/iostream:41,
                 from tmp.raii.cpp:4:
/usr/include/stdlib.h:87: note: it was later defined here
```

```
   87 | #define RAND_MAX        2147483647
      |
```

[23]:
```
!./tmp.raii.exe
```

```
sh: 1: ./tmp.raii.exe: not found
```

1.5 Minimal strategy: forbid the copy

[24]:
```
%%file tmp.guards.h

class ParticleGuard {
  public :
    explicit ParticleGuard( Particle * p ) : m_p(p) {}
    ~ParticleGuard() { delete m_p ; }
    Particle * get() { return m_p ; }
  private :
    ParticleGuard( const ParticleGuard & ) ;
    ParticleGuard & operator=( const ParticleGuard & ) ;
    Particle * m_p ;
} ;
```

```
Overwriting tmp.guards.h
```

Our modified `print()` is not any more allowed:

[25]:
```
%%file tmp.raii.cpp

#include "tmp.particles.h"
#include "tmp.guards.h"
#include <iostream>
#include <ctime>

void print( ParticleGuard pg ){
  std::cout<<pg.get()->name()<<std::endl ;
}

int main() {
  std::srand(std::time(0)) ;
  for ( int i = 0 ; i < 5 ; ++i ) {
    ParticleGuard pg(new_particle()) ;
    print(pg) ;
  }
}
```

```
Overwriting tmp.raii.cpp
```

[26]:
```
!rm -f tmp.raii.exe && g++ -std=c++17 tmp.raii.cpp -o tmp.raii.exe
```

```
In file included from tmp.raii.cpp:2:
tmp.particles.h: In function ‘Particle* new_particle()’:
tmp.particles.h:28:22: error: ‘rand’ is not a member of ‘std’
   28 |   int num = NB*(std::rand()/(double(RAND_MAX)+1)) ;
      |                      ^~~~
tmp.particles.h:28:37: error: ‘RAND_MAX’ was not declared in this scope
   28 |   int num = NB*(std::rand()/(double(RAND_MAX)+1)) ;
      |                                     ^~~~~~~~
tmp.particles.h:28:37: note: the macro ‘RAND_MAX’ had not yet been defined
In file included from /usr/local/include/c++/13.2.0/cstdlib:79,
                 from /usr/local/include/c++/13.2.0/ext/string_conversions.h:43,
                 from /usr/local/include/c++/13.2.0/bits/basic_string.h:4097,
                 from /usr/local/include/c++/13.2.0/string:54,
                 from /usr/local/include/c++/13.2.0/bits/locale_classes.h:40,
                 from /usr/local/include/c++/13.2.0/bits/ios_base.h:41,
                 from /usr/local/include/c++/13.2.0/ios:44,
                 from /usr/local/include/c++/13.2.0/ostream:40,
                 from /usr/local/include/c++/13.2.0/iostream:41,
                 from tmp.raii.cpp:4:
/usr/include/stdlib.h:87: note: it was later defined here
   87 | #define RAND_MAX        2147483647
      |
tmp.raii.cpp: In function ‘int main()’:
tmp.raii.cpp:15:10: error: ‘ParticleGuard::ParticleGuard(const ParticleGuard&)’
is private within this context
   15 |     print(pg) ;
      |     ~~~~~^~~~
In file included from tmp.raii.cpp:3:
tmp.guards.h:8:5: note: declared private here
    8 |     ParticleGuard( const ParticleGuard & ) ;
      |     ^~~~~~~~~~~~~
tmp.raii.cpp:7:27: note:   initializing argument 1 of ‘void
print(ParticleGuard)’
    7 | void print( ParticleGuard pg ){
      |             ~~~~~~~~~~~~~~^~
```

In our example, we can easily get the expected result with some reference instead:

[27]:
```
%%file tmp.raii.cpp

#include "tmp.particles.h"
#include "tmp.guards.h"
#include <iostream>
#include <ctime>

void print( ParticleGuard & pg ){
  std::cout<<pg.get()->name()<<std::endl ;
}
```

```
int main() {
  std::srand(std::time(0)) ;
  for ( int i = 0 ; i < 5 ; ++i ) {
    ParticleGuard pg(new_particle()) ;
    print(pg) ;
  }
}
```

```
Overwriting tmp.raii.cpp
```

[28]:
```
!rm -f tmp.raii.exe && g++ -std=c++17 tmp.raii.cpp -o tmp.raii.exe
```

```
In file included from tmp.raii.cpp:2:
tmp.particles.h: In function ‘Particle* new_particle()’:
tmp.particles.h:28:22: error: ‘rand’ is not a member of ‘std’
   28 |   int num = NB*(std::rand()/(double(RAND_MAX)+1)) ;
      |                      ^~~~
tmp.particles.h:28:37: error: ‘RAND_MAX’ was not declared in this scope
   28 |   int num = NB*(std::rand()/(double(RAND_MAX)+1)) ;
      |                                     ^~~~~~~~
tmp.particles.h:28:37: note: the macro ‘RAND_MAX’ had not yet been defined
In file included from /usr/local/include/c++/13.2.0/cstdlib:79,
                 from /usr/local/include/c++/13.2.0/ext/string_conversions.h:43,
                 from /usr/local/include/c++/13.2.0/bits/basic_string.h:4097,
                 from /usr/local/include/c++/13.2.0/string:54,
                 from /usr/local/include/c++/13.2.0/bits/locale_classes.h:40,
                 from /usr/local/include/c++/13.2.0/bits/ios_base.h:41,
                 from /usr/local/include/c++/13.2.0/ios:44,
                 from /usr/local/include/c++/13.2.0/ostream:40,
                 from /usr/local/include/c++/13.2.0/iostream:41,
                 from tmp.raii.cpp:4:
/usr/include/stdlib.h:87: note: it was later defined here
   87 | #define RAND_MAX        2147483647
      |
```

[29]:
```
!./tmp.raii.exe
```

```
sh: 1: ./tmp.raii.exe: not found
```

1.6 Robust (but slow) strategy: internally count the guards

All the guards must share a common control block.

[30]:
```
%%file tmp.guards.h

class ParticleControlBlock {
  public:
    unsigned int m_nb_guards ;
```

```
    explicit ParticleControlBlock( Particle * p )
     : m_p(p), m_nb_guards(1) {}
    ~ParticleControlBlock()
     { delete m_p ; }
    Particle * get() { return m_p ; }
  private:
    ParticleControlBlock( const ParticleControlBlock & ) ;
    ParticleControlBlock & operator=( const ParticleControlBlock & ) ;
    Particle * m_p ;
} ;

void increment_guards( ParticleControlBlock * block ) {
  block->m_nb_guards++ ;
}

void decrement_guards( ParticleControlBlock * block ) {
  block->m_nb_guards-- ;
  if (block->m_nb_guards==0)
   { delete block ; }
}

class ParticleGuard {
  public:
    explicit ParticleGuard( Particle * p )
     { m_block = new ParticleControlBlock(p) ; }
    ParticleGuard( const ParticleGuard & pg )
     {
      m_block = pg.m_block ;
      increment_guards(m_block) ;
     }
    ParticleGuard & operator=( ParticleGuard const & pg )
     {
      if (this==&pg) return *this ;
      decrement_guards(m_block) ;
      m_block = pg.m_block ;
      increment_guards(m_block) ;
      return *this ;
     }
    ~ParticleGuard()
     { decrement_guards(m_block) ; }
    Particle * get()
     { return m_block->get() ; }
  private :
    ParticleControlBlock * m_block ;
} ;
```

```
Overwriting tmp.guards.h
```

[31]:
```
%%file tmp.raii.cpp

#include "tmp.particles.h"
#include "tmp.guards.h"
#include <iostream>
#include <ctime>

void print( ParticleGuard pg ){
  std::cout<<pg.get()->name()<<std::endl ;
}

int main() {
  std::srand(std::time(0)) ;
  for ( int i = 0 ; i < 5 ; ++i ) {
    ParticleGuard pg(new_particle()) ;
    print(pg) ;
  }
}
```

```
Overwriting tmp.raii.cpp
```

[32]:
```
!rm -f tmp.raii.exe && g++ -std=c++17 tmp.raii.cpp -o tmp.raii.exe
```

```
In file included from tmp.raii.cpp:2:
tmp.particles.h: In function ‘Particle* new_particle()’:
tmp.particles.h:28:22: error: ‘rand’ is not a member of ‘std’
   28 |   int num = NB*(std::rand()/(double(RAND_MAX)+1)) ;
      |                      ^~~~
tmp.particles.h:28:37: error: ‘RAND_MAX’ was not declared in this scope
   28 |   int num = NB*(std::rand()/(double(RAND_MAX)+1)) ;
      |                                     ^~~~~~~~
tmp.particles.h:28:37: note: the macro ‘RAND_MAX’ had not yet been defined
In file included from /usr/local/include/c++/13.2.0/cstdlib:79,
                 from /usr/local/include/c++/13.2.0/ext/string_conversions.h:43,
                 from /usr/local/include/c++/13.2.0/bits/basic_string.h:4097,
                 from /usr/local/include/c++/13.2.0/string:54,
                 from /usr/local/include/c++/13.2.0/bits/locale_classes.h:40,
                 from /usr/local/include/c++/13.2.0/bits/ios_base.h:41,
                 from /usr/local/include/c++/13.2.0/ios:44,
                 from /usr/local/include/c++/13.2.0/ostream:40,
                 from /usr/local/include/c++/13.2.0/iostream:41,
                 from tmp.raii.cpp:4:
/usr/include/stdlib.h:87: note: it was later defined here
   87 | #define RAND_MAX        2147483647
      |
```

[33]:
```
!./tmp.raii.exe
```

```
sh: 1: ./tmp.raii.exe: not found
```

1.7 Simple (but not universal) strategy : deep copy

Sometimes it is preferable to duplicate the resource each time its guard is copied.

In our example, there is the additional difficulty that the real class of the pointed particle is not known, hence the need for some virtual `clone` function in the `Particle` inheritance tree.

[34]:
```
%%file tmp.particles.h

#include <cstdlib> // rand()

class Particle {
  public :
    virtual ~Particle() {}
    virtual const char * name() const =0 ;
    virtual Particle * clone() const =0 ;
} ;

class Electron : public Particle {
  public :
    virtual const char * name() const { return "Electron" ; }
    virtual Particle * clone() const { return new Electron(*this) ; }
} ;

class Proton : public Particle {
  public :
    virtual const char * name() const { return "Proton" ; }
    virtual Particle * clone() const { return new Proton(*this) ; }
} ;

class Neutron : public Particle {
  public :
    virtual const char * name() const { return "Neutron" ; }
    virtual Particle * clone() const { return new Neutron(*this) ; }
} ;

Particle * new_particle() {
  const int NB = 3 ;
  int num = NB*(std::rand()/(double(RAND_MAX)+1)) ;
  switch (num)
   {
    case 0: return new Electron() ;
    case 1: return new Proton() ;
    case 2: return new Neutron() ;
   }
  return 0 ;
}
```

```
Overwriting tmp.particles.h
```

[37]:
```
%%file tmp.guards.h

class ParticleGuard {
  public:
    explicit ParticleGuard( Particle * p ) : m_p(p) {}
    ParticleGuard( const ParticleGuard & pg )
     { m_p = pg.m_p->clone() ; }
    ParticleGuard & operator=( const ParticleGuard & pg )
     {
      if (this==&pg) return (*this) ;
      delete m_p ; m_p = pg.m_p->clone() ;
      return (*this) ;
     }
    ~ParticleGuard() { delete m_p ; }
    Particle * get() { return m_p ; }
  private :
    Particle * m_p ;
} ;
```

```
Overwriting tmp.guards.h
```

[38]:
```
!rm -f tmp.raii.exe && g++ -std=c++17 tmp.raii.cpp -o tmp.raii.exe
```

[40]:
```
!./tmp.raii.exe
```

```
Electron
Electron
Proton
Neutron
Proton
```

NOTE: `std::string` is a kind of front-end for a dynamically allocated array of characters. That's why it has no maximum size. When a string is duplicated, there is a deep copy.

1.8 Subtle (but unnatural) strategy : transfer the resource

In rare cases, we want to allow the copy of a guard, stealing the resource from the former guard, and letting it *empty*.

[42]:
```
%%file tmp.guards.h

class ParticleGuard {
  public:
    explicit ParticleGuard( Particle * p ) : m_p(p) {}
    ParticleGuard( ParticleGuard & pg )
     { m_p = pg.m_p ; pg.m_p = 0 ; }
    ParticleGuard & operator=( ParticleGuard & pg )
     {
      if (this==&pg) return (*this) ;
      delete m_p ; m_p = pg.m_p ; pg.m_p = 0 ;
      return (*this) ;
     }
    ~ParticleGuard() { delete m_p ; }
    Particle * get() { return m_p ; }
  private :
    Particle * m_p ;
} ;
```

```
Overwriting tmp.guards.h
```

[43]:
```
!rm -f tmp.raii.exe && g++ -std=c++17 tmp.raii.cpp -o tmp.raii.exe
```

[45]:
```
!./tmp.raii.exe
```

```
Neutron
Neutron
Electron
Electron
Neutron
```

BEWARE: this requires an unusual copy constructor, which modify the original guard. The infamous class `std::auto_ptr` was implemented this way, with the major limitation that one cannot store some `std::auto_ptr` in some `std::vector`. Solving this issue requires a new feature : the ***move semantic*** .

2 Take away

- Pointers stay both the blessing and the malediction of the language.
- Modern C++ will make them a lot less necessary or a lot more hidden in the standard library.
- A key elegant feature is the new **rvalue reference** and the associated **move semantic**.

3 Questions ?