POLYTECH NICE-SOPHIA

Raw Storage

Assignment #3 (1)

- Write a short program in C++ which reads an image in YUV format (YV12) and converts it in RGB format for a display on the computer screen. The program shall auto-detect the image height and width based on the supported formats shown below.
- Use the QT environment

Image 6x4

Position in byte stream:

Format	Width	Height
CIF	352	288
WVGA+	832	480
HD	1920	1080
UHD	3840	2160

Source: http://en.wikipedia.org/wiki/YUV

Assignment #3 (2)

- Formula for YUV to RGB Conversion

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \boldsymbol{M} \begin{pmatrix} Y \\ U \\ V \\ 1 \end{pmatrix}$$

$\boldsymbol{M}$ =

1.1643836	0.0000000	1.8765140	-268.2065015
1.1643836	-0.2132486	-0.5578116	82.8546329
1.1643836	2.1124018	0.0000000	-289.0175656
0.0000000	0.0000000	0.0000000	1.0000000

- Some C++ STL library functions

```
ifstream(const string& filename, ios_base::openmode mode = ios_base::in );
ifstream& read(char* s, std::streamsize count);
ifstream& seekg( pos_type pos );
pos_type tellg();
```

Solution #3 (1)

```
main() {
  string file_name = “image.yuv”;

  // image constructor
  YuvImage image(file_name);

  // image display...

}
```

```
class YuvImage : public QImage {
public:
  explicit YuvImage(const std::string &file_name);

private:

  uint8_t *y_raw_;
  uint8_t *u_raw_;
  uint8_t *v_raw_;
};
```

Solution #3 (2)

```
YuvImage::YuvImage(const std::string &file_name) {
  ...
  int y_size = width_ * height_;
  int uv_size =  y_size >> 2;

  y_raw_ = new uint8_t [y_size];
  u_raw_ = new uint8_t [uv_size];
  v_raw_ = new uint8_t [uv_size];


  yuv_strm.read(reinterpret_cast<char *>(y_raw_), y_size);
  yuv_strm.read(reinterpret_cast<char *>(u_raw_), uv_size);
  yuv_strm.read(reinterpret_cast<char *>(v_raw_), uv_size);

}
```

Using new to allocate raw storage on the heap

Pointer cast mandatory otherwise you have an error message

```
error: invalid conversion from 'uint8_t* {aka unsigned char*}' to
'std::basic_istream<char>::char_type* {aka char*}'
     yuv_strm.read(y_raw_, y_size);
```

Solution #3 (3)

```
{
  string file_name = “image.yuv”;

  // image constructor
  YuvImage image(file_name);

  // image display...

}
```

What is happening here ?

Managing Raw Storage (1)

- Raw Storage: chunk of *n* contiguous bytes on the *heap* provided by the kernel to the application.
- In C++, we use the `new` operator to *acquire* raw storage, it is not initialized.
- We use the `delete` operator to *release* raw storage to the kernel.
  - The application must ensure that allocated memory is also deleted otherwise you will have a memory leakage

```
{
  int *my_int_ptr = new int [10];

  ...

  delete [] my_int_ptr;
}
```

```
{
  auto my_obj_ptr = new Object();

  ...

  delete my_opt_ptr;
}
```

Managing Raw Storage (2)

- Always ask your self the key question:

Do you really need
to use raw storage ?

```
{
  Object my_obj{};

  ...

}
```

```
{
  int *my_int_ptr = new int [10];

  ...

  delete [] my_int_ptr;
}
```

```
{
  auto my_obj_ptr = new Object();

  ...

  delete my_opt_ptr;
}
```

Solution #3 (4)

```
{
  string file_name = “image.yuv”;

  // image constructor
  YuvImage image(file_name);

  // image display...

}
```

Destructor is called when living the scope

```
class YuvImage : public QImage {
public:
  explicit YuvImage(const std::string &file_name);
  ~YuvImage() {
    delete [] y_raw_;
    delete [] u_raw_;
    delete [] v_raw_;
  }
```

Raw Storage Usage (1)

1. Do you really need to use raw storage ?
2. You must write your own destructor.

Robust Solution (1)

```
class YuvImage : public QImage {
public:
  explicit YuvImage(const std::string &file_name);
  ~YuvImage() {
    delete [] y_raw_;
    delete [] u_raw_;
    delete [] v_raw_;
  }
private:
  uint8_t *y_raw_;
  ...
};
```

```
{
 ...
  // image constructor
  YuvImage image(file_name);
  YuvImage image1 = image;
  ...
}
```

Can you spot the bug ?
Hint1: implicitly declared and defined copy constructor
Hint2: is the raw data copied ?

Robust Solution (2)

```
#include <cstring>
...
class YuvImage : public QImage {
public:
  explicit YuvImage(const std::string &file_name);
  ~YuvImage() {
    delete [] y_raw_;
    delete [] u_raw_;
    delete [] v_raw_;
  }
  YuvImage(const YuvImage &image) { ... }
    -or-
  YuvImage(const YuvImage &image) = delete;

  YuvImage(YuvImage &&image) noexcept { ... }

private:
  int width_;
  uint8_t *y_raw_;
...
};
```

You must have a copy constructor either defined or marked delete

You may also need a move constructor

Robust Solution (3)

```
#include <cstring>
...
class YuvImage : public QImage {
public:
  explicit YuvImage(const std::string &file_name);
  ~YuvImage() {
    delete [] y_raw_;
    delete [] u_raw_;
    delete [] v_raw_;
  }
  YuvImage(const YuvImage &image) {
    width_ = image.width_;
    height_ = image.height;
    auto y_size = width_ * height_;
    auto uv_size =  y_size >> 2;
    y_raw_ = new uint8_t [y_size];
    u_raw_ = new uint8_t [uv_size];
    v_raw_ = new uint8_t [uv_size];

    std::memcpy(y_raw_, image.y_raw_, y_size);
    ...
```

Anything missing ?

How to copy raw storage ?
2 steps process
- acquire memory
- initialize memory

Memory acquisition

Memory initialization

Robust Solution (4)

```
#include <cstring>
...
class YuvImage : public QImage {
public:
  explicit YuvImage(const std::string &file_name);
  ~YuvImage() {
    delete [] y_raw_;
    delete [] u_raw_;
    delete [] v_raw_;
  }
  YuvImage(const YuvImage &image) : QImage(image) {
    width_ = image.width_;
    height_ = image.height;
    auto y_size = width_ * height_;
    auto uv_size =  y_size >> 2;
    y_raw_ = new uint8_t [y_size];
    u_raw_ = new uint8_t [uv_size];
    v_raw_ = new uint8_t [uv_size];

    std::memcpy(y_raw_, image.y_raw_, y_size);
    ...
```

YuvImage is a derived class => the base class must also be initialized.

Robust Enough ?

- What if
  - file can't be opened ?
  - file size is incorrect ?
  - file can't be read ?
  - raw memory can't be allocated ?
- Design issue: what must be done in case of *exceptions* ?
  - ignore & return silently to caller
  - notify & return to caller
  - exit program
- Language issue: how can we detect and deal with exception ?
  - Using `try { } catch() { }` syntax

Smart Pointer [1]

- Alternative to Raw Storage
  - Unique Pointer or Shared Pointer
  - Object encapsulating a pointer
  - Declaration

```
std::unique_ptr<T> my_uptr;
std::unique_ptr<T[]> my_uptrs;
```

```
private:
  int width_;
  int height_;

  std::unique_ptr<uint8_t []> y_raw_;
  std::unique_ptr<uint8_t []> u_raw_;
  std::unique_ptr<uint8_t []> v_raw_;
};
```

```
private:
  int width_;
  int height_;

  uint8_t *y_raw_;
  uint8_t *u_raw_;
  uint8_t *v_raw_;
};
```

SmartPointer [2]

- RAII

```
std::unique_ptr<T> my_uptr(new T());
std::unique_ptrs<T[]> my_uptrs(new T [size]);
```

Default constructor of T

- Allocation

```
my_uptr = std::make_unique<T>()
my_uptrs = std::make_unique<T[]>(size)
```

Default constructor of T

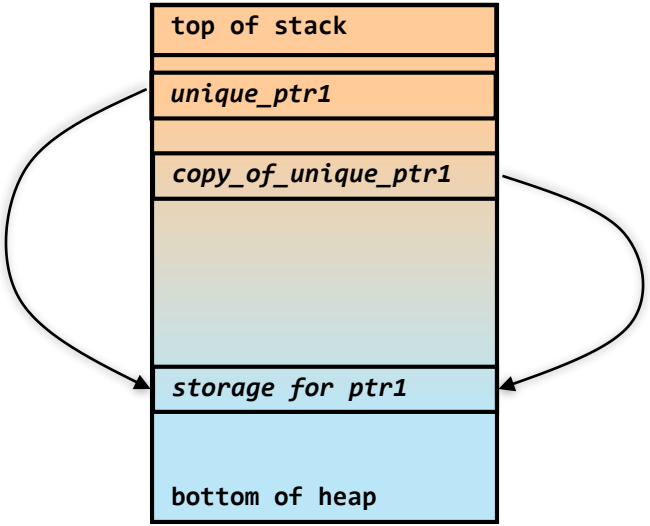
  - Destruction
    - implicit
  - Dereferencing

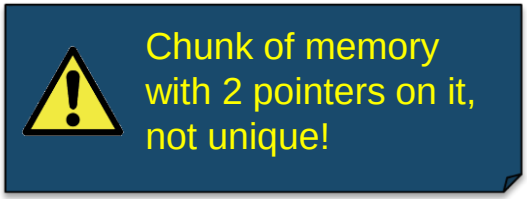
```
*my_uptr = a_t_obj;     a_t_obj = *my_uptr
my_uptrs[4] = a_t_obj   a_t_obj = my_upts[4];
```

SmartPointer [3]

- Beware: unique pointer are not copiable

```
std::unique_ptr(const unique_ptr&) = delete;
std::unique_ptr& operator=(const unique_ptr&) = delete;
```





POLYTECH
NICE-SOPHIA

Exceptions

Try & Catch: Overview (1)

```
YuvImage(const std::string &file_name) : QImage(300, 100, QImage::Format_RGB32) {


  ...


  try {
    std::ifstream yuv_strm(file_name, std::ios::in | std::ios::binary);
    yuv_strm.exceptions(std::ifstream::eofbit | std::ifstream::failbit);
    ...



    load_from_stream(yuv_strm);
    ...
  }
  catch(const std::ios_base::failure &error) {
    std::cerr << "Error: i/o file error = " << error.what() << std::endl;
  }
  catch(const std::bad_alloc &error) {
    std::cerr << "Error: memory error = " << error.what() << std::endl;
  }
  catch(const std::exception &error) {
    std::cerr << "Error: other error = " << error.what() << std::endl;
  }
```

Try & Catch: Overview (2)

```
YuvImage(const std::string &file_name) : QImage(300, 100, QImage::Format_RGB32) {


  ...


  try {
    std::ifstream yuv_strm(file_name, std::ios::in | std::ios::binary);
    yuv_strm.exceptions(std::ifstream::eofbit | std::ifstream::failbit);
    ...



    load_from_stream(yuv_strm);
    ...
  }
  catch(const std::ios_base::failure &error) {
    std::cerr << "Error: i/o file error = " << error.what() << std::endl;
  }
  catch(const std::bad_alloc &error) {
    std::cerr << "Error: memory error = " << error.what() << std::endl;
  }
  catch(const std::exception &error) {
    std::cerr << "Error: other error = " << error.what() << std::endl;
  }
```

Beware of partially constructed object => destructor behavior

Try & Catch: Overview (3)

```
YuvImage(const std::string &file_name) : QImage(300, 100, QImage::Format_RGB32) {

  y_raw_ = nullptr;
  u_raw_ = nullptr;
  v_raw_ = nullptr;

  try {
    std::ifstream yuv_strm(file_name, std::ios::in | std::ios::binary);
    yuv_strm.exceptions(std::ifstream::eofbit | std::ifstream::failbit);
    ...



    load_from_stream(yuv_strm);
    ...
  }
  catch(const std::ios_base::failure &error) {
    std::cerr << "Error: i/o file error = " << error.what() << std::endl;
  }
  catch(const std::bad_alloc &error) {
    std::cerr << "Error: memory error = " << error.what() << std::endl;
  }
  catch(const std::exception &error) {
    std::cerr << "Error: other error = " << error.what() << std::endl;
  }
```

prevent runtime error on delete []

Try & Catch: Throw (1)

```
void YuvImage::load_from_stream(std::ifstream &yuv_strm) {

  auto y_size = width_ * height_;
  auto uv_size =  y_size >> 2;

  // new uint8_t []  may throw and
  // we have a catch std::bad_alloc in the caller

  y_raw_ = new uint8_t [y_size];
  u_raw_ = new uint8_t [uv_size];
  v_raw_ = new uint8_t [uv_size];

  // yuv_strm.read()  may throw and
  // we have a catch ios_base::failure in the caller

  yuv_strm.read(reinterpret_cast<char *>(y_raw_), y_size);
  yuv_strm.read(reinterpret_cast<char *>(u_raw_), uv_size);
  yuv_strm.read(reinterpret_cast<char *>(v_raw_), uv_size);
}
```

Plenty of throw are possible in these methods

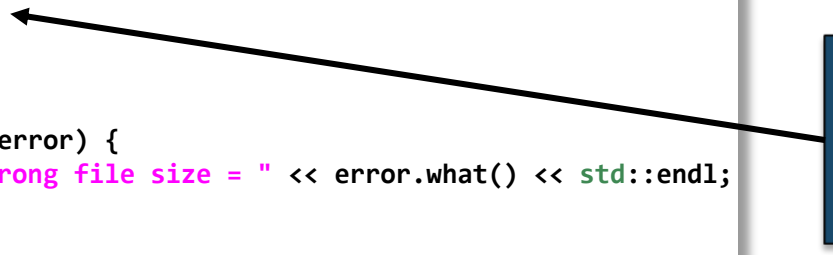
Try & Catch: Custom (1)

```
YuvImage(const std::string &file_name) : QImage(300, 100, QImage::Format_RGB32) {

  ...
  try {
    std::ifstream yuv_strm(file_name, std::ios::in | std::ios::binary);
    yuv_strm.exceptions(std::ifstream::eofbit | std::ifstream::failbit);
    ...
    if (size_is_incorrect) {
      throw wrong_size{};
    }
    load_from_stream(yuv_strm);
    ...
  }
  catch(const std::ios_base::failure &error) {
    std::cerr << "Error: i/o file error = " << error.what() << std::endl;
  }
  catch(const std::bad_alloc &error) {
    std::cerr << "Error: memory error = " << error.what() << std::endl;
  }
  catch(const wrong_size &error) {
    std::cerr << "Error: wrong file size = " << error.what() << std::endl;
  }
  catch(const std::exception &error) {
    std::cerr << "Error: other error = " << error.what() << std::endl;
  }
```

Try & Catch: Custom (2)

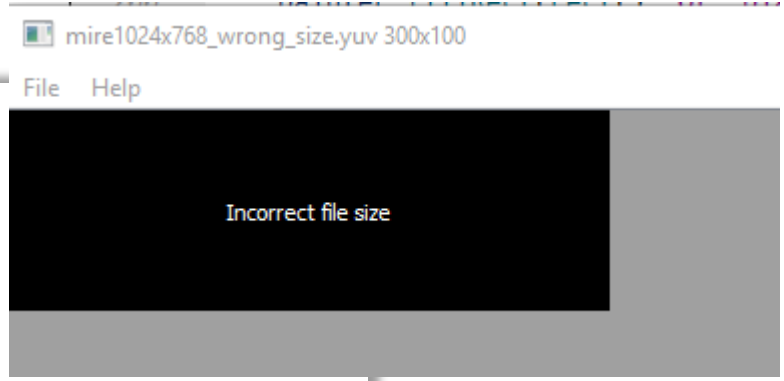
```
class YuvImage : public QImage {
 public:
  void load_image();

  class wrong_size : public std::exception {
   public:
    const char *what() const noexcept override {
      return "File has wrong size";
    }
  };
  ...
};
```

```
void YuvImage::load_image() {
  ...
  try {
    bool incorrect_size = false;

    ...

    if (incorrect_size) {
      throw wrong_size{};
    }
    ...
  }
  catch(const wrong_size &error) {
    std::cerr << "Error: wrong file size = " << error.what() << std::endl;
  }
}
```



What after a catch? [1]

```
void YuvImage::load_image() {
  ...
  try {
    bool incorrect_size = false;
    ...
    if (incorrect_size) {
      throw wrong_size{};
    }
    ...
  }
  catch(const wrong_size &error) {
    std::cerr << "Error: wrong file size = " << error.what() << std::endl;
    QPainter painter(this);
    painter.fillRect(rect(), Qt::black);
    painter.setPen(Qt::white);
    painter.drawText(rect(), Qt::AlignCenter, "Incorrect file size");
  }
  ...
}
```



Let the program continue after a specific processing

What after a catch? [2]

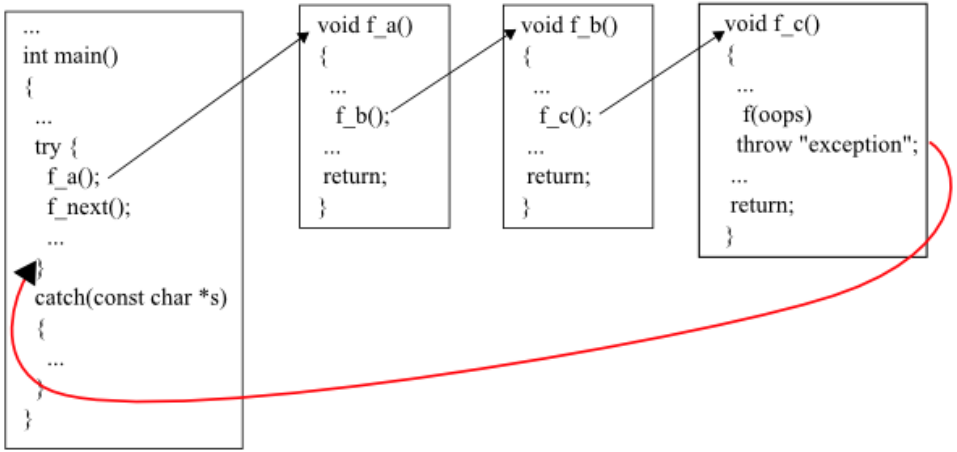
```
void YuvImage::load_image() {
  ...
  try {
    bool incorrect_size = false;
    ...
    if (incorrect_size) {

      throw wrong_size(my_str);
    }
    ...
  }
  catch(const wrong_size &error) {
    std::cerr << "Error: wrong file size = " << error.what() << '\n';
    std::cerr << "Error at line " << __LINE___ << '\n';
    std::cerr << "Error in funct " << __func___ << '\n';
    throw generic_error()
  }
  ...
}
```

```
  // in caller block
  ...
  try {
    ...
    load_image();
  }
  catch(const generic_error &error) {
    std::cerr << "Error at line " << __LINE___ << '\n';
    std::cerr << "Error in funct " << __func___ << '\n';
    throw generic_error();
  }
  ...
}
```

Rethrow the exception

Exceptions: Stack Unwinding [1]

- Stack unwinding occurs when exception is thrown
  - Destructors are called on all objects created in scope**s**
  - From current stack level to level where the matching catch occurs.



https://www.bogotobogo.com/cplusplus/stackunwinding.php

Exceptions: Stack Unwinding [2]

```
291
292  Edge &get_edge(uint id_src, uint id_dst) {
293    try {
294      if (not edge_exist(id_src, id_dst)) {
295        throw MapError::no_such_edge(id_src, id_dst);
296      }
297      EdgeKey key(id_src, id_dst);
298      auto result =  edge_map_.find(key);
299      return result->second;
300    }
301    catch(MapError::no_such_edge &e) {
302      std::cerr << e.what() << std::endl;
303      std::cerr << "Exception detected in function " << __FUNCTION__ << "() at line " << __LINE__ << std::endl;
304      throw;
305    }
306  }
```

```
Error 1003: Edge from 0 to 0 does not exist
Exception detected in function get_edge() at line 303
Exception detected in function update_edge_length() at line 350
Exception detected in function generate_graph_aux() at line 176
Exception detected in function generate_graph() at line 202
Exception detected in function RandomMap() at line 239
Exception detected in function main() at line 319
```

Exceptions: Stack Unwinding [3]

```
291
292   Edge &get_edge(uint id_src, uint id_dst) {
293     try {
294       if (not edge_exist(id_src, id_dst)) {
295         throw MapError::no_such_edge(id_src, id_dst);
```

```
335 void update_edge_length(uint id_src, uint  id_dst) {
336   try {
337     Vertex &v_src =  get_vertex(id_src);
338     Vertex &v_dst =  get_vertex(id_dst);
339
340     auto xy_src = v_src.get_xy_coordinates();
341     auto xy_dst = v_dst.get_xy_coordinates();
342                                                                    __LINE__ << std::endl;
343     auto distance = std::hypot(xy_src.first - xy_dst.first,
344                                xy_src.second - xy_dst.second);
345
346     Edge &edge = get_edge(id_src, id_src);
347     edge.set_length(distance);
348   }
349   catch(...) {
350     std::cerr << "Exception detected in function " << __FUNCTION__ << "() at line " << __LINE__ << std::endl;
351     throw;
352   }
353 }
```

```
Error 1003: Edge from 0 to 0 does not exist
Exception detected in function get_edge() at line 303
Exception detected in function update_edge_length() at line 350
Exception detected in function generate_graph_aux() at line 176
Exception detected in function generate_graph() at line 202
Exception detected in function RandomMap() at line 239
Exception detected in function main() at line 319
```

More on Exceptions [2a]

```
class generic_error: public std::exception {
 public:
  const char *what() const noexcept override {
    return "Generic_error";
  }
};

struct String {
  string str_;
  String(const string &str) : str_{str} {
    cout << "Constructing String " << str_
         << endl;
  }
  ~String()  {
    cout << "Destroying String " << str_
         << endl;
  }
  int length() const {
    throw generic_error();
    return str_.size();
  }
};
```

```
void test_raii() {
  try {
    String *sptr = new String("Hello");

    int sum = sptr->length();
    cout << "Length of string: " << sum << endl;

    delete sptr;
  }
  catch(generic_error &ex) {
    cout << "Generic_error: e.what is"
         << ex.what()
         << endl;

  }
}
```

Any problem here?

More on Exceptions [2b]

```
class generic_error: public std::exception {
 public:
  const char *what() const noexcept override {
    return "Generic_error";
  }
};

struct String {
  string str_;
  String(const string &str) : str_{str} {
    cout << "Constructing String " << str_
         << endl;
  }
  ~String()  {
    cout << "Destroying String " << str_
         << endl;
  }
  int length() const {
    throw generic_error();
    return str_.size();
  }
};
```

```
void test_raii_v2() {
  try {
    unique_ptr<String> suptr(new String("Hello"));

    int sum = suptr->length();
    cout << "Length of string: " << sum << endl;


  }
  catch(generic_error &ex) {
    cout << "Generic_error: e.what is"
         << ex.what()
         << endl;

  }
}
```

More on Exceptions [3]

- Exception during object construction
  - Possible to signal problem during creation
  - Destructor is not called, but...
    - ...destructors on member variables are called.
    - ... allocated memory is released.
- Never throw an exception in a destructor
  - Herb Sutter: Destructors that throw and why they're evil
  - In general, this prevents class objects to be fully deleted.

More on Exceptions [4]

```
 4 struct String {
 5   string str_;
 6   String(const string &str) : str_{str} {
 7     cout << "Constructing String "
          << str_ << endl;
 8   }
 9   ~String()  {
10     cout << "Destroying String "
          << str_ << endl;
11   }
...
23 };
24
25 class generic_error: public std::exception {
26  public:
27   const char *what() const noexcept override {
28     return "Generic_error";
29   }
30 };
```

```
32 struct Player {
33   String name_;
34   Player(const string &name) : name_{name}  {
35     cout << "Constructing Player "
          << name_ << endl;
36     if (name_ == "FOOBAR") {
37       throw generic_error();
38     }
39   }
..
115 void test2() {
116   try {
117     cout << "============= TEST2" << endl;
118     cout << "creating FOOBAR" << endl;
119     Player player2("FOOBAR");
120     cout << "end creating FOOBAR" << endl;
121   }
122   catch(generic_error &error) {
123     cout << "Generic_error: e.what is "
          << error.what() << endl;
124  }
128 }
```

More on Exceptions [3]

```
============= TEST2
creating FOOBAR
Constructing String FOOBAR
Constructing Player FOOBAR
Destroying String FOOBAR
Generic_error: e.what is Generic_error
```

Static Objects

Static Variable (1)

- File scope static variable
  - Only visible within current file
- Function scope static variable
  - Only visible within current function, initialized once
  - Destructor called after main()

main.cpp

```
extern int counter_b;

int main() {
  incr_counter();
  counter_b++;
}
```

print.cpp

```
static int counter_a;
int counter_b;

void incr_counter() {
  static bool must_init = true;
  if (must_init) {
    counter_a = 0;
    counter_b = 0;
    must_init = false;
  } else {
    counter_a++;
    counter_b++;
  }
}
```

Static Variable (2.)

- Class scope static variable
  - Common to all objects of the class.
  - Can not be initialized

error: ISO C++ forbids in-class initialization of non-const static member 'MyObj::cnt_'

myobj.cpp

```
#include <vector>
#include <iostream>
using namespace std;

class MyObj {
 private:
  static int cnt_ = 0;
  vector<int> v_;
 public:
  MyObj() {
    cnt_++;
    v_ = {1, 4, 9};
  }
  void print_cnt() {
    cout << “obj id = " << cnt_
         << "\n";
  }
};
```

Static Variable (2..)

- Class scope static variable
  - Common to all objects of the class.
  - Can not be initialized
  - Initialization must be done outside the class

myobj.cpp

```
#include <vector>
#include <iostream>
using namespace std;

class MyObj {
 private:
  static int cnt_;
  vector<int> v_;
 public:
  MyObj() {
    cnt_++;
    v_ = {1, 4, 9};
  }
  void print_cnt() {
    cout << “obj id = " << cnt_
         << "\n";
  }
};

int MyObj::cnt_ = 0;
```

Static Function (1)

- File scope static function
  - Only visible within current file

main.cpp

```
#include <vector>
#include "print.h"

int main() {
  vector<int> v = { 1, 4, 9, 16};
  print_avg();
}
```

print.cpp

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

using namespace std;

static int
average(const vector<int> &v) {
  int sum = 0;
  for ( int x : v ) {
    sum += x;
  }
  return v.empty() ? 0 : sum / v.size();
}

void print_avg(const vector<int> &v) {
  auto avg = average(v);
  cout << "Average = " << avg << '\n';
}
```

Static Function (2.)

- Class scope static function

main.cpp

```
int main() {
  std::cout << "Program start" << std::endl;
  MyObj &obj1 = MyObj::instance();
  MyObj &obj2 = MyObj::instance();

  obj1.print();
  obj2.print();

  std::cout << "Program end" << std::endl;
}
```

```
Program start
Constructor called (1)
Print id = 1
Print id = 1
Program end
Destructor called
```

print.cpp

```
...
static int cnt = 0;

class MyObj {
 private:
  int cnt_;
  std::string name_;

  MyObj(std::string name): name_{name} {
    cnt++;
    cnt_ = cnt++;
    std::cout << "Constructor called (" << cnt_ << ")“
              << std::endl;
  }


 public:
  static MyObj &instance() {
    static MyObj obj("single");

    return obj;
  }
  void print() {
    std::cout << "Print id = "<<  cnt_ << std::endl;
  }
  ~MyObj() {
    std::cout << "Destructor called" << std::endl;
  }
};
```

Static Function (2..)

- Class scope static function
  - this pointer not available
  - no access to class members
  - cannot be const

What is missing ?

Typical e
singleton design pattern:
only one object of the
class can be created

print.cpp

```
...
static int cnt = 0;

class MyObj {
 private:
  int cnt_;
  std::string name_;

  MyObj(std::string name): name_{name} {
    cnt++;
    cnt_ = cnt++;
    std::cout << "Constructor called (" << cnt_ << ")“
              << std::endl;
  }


 public:
  static MyObj &instance() {
    static MyObj obj("single");

    return obj;
  }
  void print() {
    std::cout << "Print id = "<<  cnt_ << std::endl;
  }
  ~MyObj() {
    std::cout << "Destructor called" << std::endl;
  }
};
```

Static Function (2...)

- Must ensure that the singleton object can not be copied
  - Copy constructor is private
  - Copy assignment is private

print.cpp

```
...
static int cnt = 0;

class MyObj {
 private:
  int cnt_;
  std::string name_;

  MyObj(std::string name): name_{name} {
    cnt++;
    cnt_ = cnt++;
    std::cout << "Constructor called (" << cnt_ << ")“
              << std::endl;
  }
  MyObj(const MyObj &obj);
  MyObj &operator=(const MyObj &obj);
 public:
  static MyObj &instance() {
    static MyObj obj("single");

    return obj;
  }
  void print() {
    std::cout << "Print id = "<<  cnt_ << std::endl;
  }
  ~MyObj() {
    std::cout << "Destructor called" << std::endl;
  }
};
```

POLYTECH NICE-SOPHIA

Casting

Static Cast (1)

- Conversion between *compatible* types.
  - `char` to `int` is OK
  - `char*` to `int*` is KO
  - More restrictive than the C-style cast.

`static_cast<type>(v)`

```
void demo1(const int i, const char c) {
  cout << "Integer as char = " << static_cast<char>(i) << endl;
  cout << "Char as Integer = " << static_cast<int>(c) << endl;
}

void demo2(const unsigned int ui) {
  cout << "Unsigned Integer                = " << ui << endl;
  cout << "Signed   Integer from Unsigned = " << static_cast<int>(ui) << endl;
}
```

Static Cast (2)

- Conversion between *compatible* types.
  - `char` to `int` is OK
  - `char*` to `int*` is KO
  - More restrictive than the C-style cast.

`static_cast<type>(v)`

```
void demo3(int *pi, char *pc) {
  cout << "char ptr from int  ptr = " << static_cast<char *>(pi) << endl;
  cout << "int  ptr from char ptr = " << static_cast<int *>(pc) << endl;
}
```

```
casting.cpp: In function 'void demo3(int*, char*)':
casting.cpp:15:65: error: invalid static_cast from type 'int*' to type 'char*'
   cout << "Pointer Integer as char = " << static_cast<char *>(pi) << endl;
                                                                 ^
casting.cpp:16:64: error: invalid static_cast from type 'char*' to type 'int*'
   cout << "Pointer Char as Integer = " << static_cast<int *>(pc) << endl;
                                                               ^
```

Reinterpret Cast

- Conversion between *incompatible* types.
  - No check done by the compiler
  - Be very cautious
  - `char*` to `int*` is OK.

`reinterpret_cast<type>(v)`

```
void demo3(int *pi, char *pc) {
  cout << "char ptr from int  ptr = " << reinterpret_cast<char *>(pi) << endl;
  cout << "int  ptr from char ptr = " << reinterpret_cast<int *>(pc) << endl;
}
```

```
Assume  i = 16961; pi = &i

char ptr from int  ptr = AB
int  ptr from char ptr = 0x61fd3b
```

Can you explain the "AB" ?

Const Cast (1)

- Remove constness `const_cast<type>(v)`
  - Can't do it directly!

```
void demo5(const int i) {
  cout << "Initial Value     = " << i << endl;
  cout << "Incremented Value = " << (++i) << endl;
}
```

```
casting.cpp:29:40: error: increment of read-only
parameter 'i'
   cout << "Incremented Value = " << (++i) << endl;
```

```
void demo6(const int i) {
  cout << "i     = " << i << endl;
  int k = const_cast<int>(i);
  cout << "++k   = " << (++k) << endl;
  cout << "i     = " << i << endl;
}
```

```
casting.cpp:35:28: error: invalid use of const_cast
with type 'int', which is not a pointer, reference,
nor a pointer-to-data-member type
   int k = const_cast<int>(i);
```

Const Cast (2)

- Remove constness `const_cast<type>(v)`
  - Can't do it directly!

```
void demo6(const int i) {
  cout << "i     = " << i << endl;
  int k = const_cast<int>(i);
  cout << "++k   = " << (++k) << endl;
  cout << "i     = " << i << endl;
}
```

```
void demo6(const int i) {
  cout << "i     = " << i << endl;
  int &k = const_cast<int &>(i);
  cout << "++k   = " << (++k) << endl;
  cout << "i     = " << i << endl;
}
```

Using reference is fine

Dynamic Cast (1)

- Type-safe downcast operation.
  - Returns **`nullptr`** if cast fail.

`dynamic_cast<type>(v)`

```
class Car {
 public:
  virtual ~Car() = default;
  virtual void get_info() = 0;
};

class Dacia : public Car {
 public:
  void get_info() override {
    cout << "A solid car" << endl;
  }
};

class Porsche : public Car {
 public:
  void get_info() override {
    cout << "A fancy car" << endl;
  }
};
```

```
int test_dynamic_cast() {

  Car *p = new Porsche();
  Car *d = new Dacia();

  p->get_info(); // porsche
  d->get_info(); // dacia

  Porsche *p_from_d = dynamic_cast<Porsche *>(d);
  if (p_from_d == nullptr)  {
    cout << "Sorry! Can't pass a Dacia for a Porsche!" << endl;
    return -1;
  }
  p_from_d->get_info();

}
```

Explicit Cast

- Convert obj from a class to another one.

```
class Audi : public Car {
 public:
  void get_info() override {
    cout << "It's an Audi" << endl;
  }
};

class Porsche : public Car {
 public:
  void get_info() override {
    cout << "It's a Porsche" << endl;
  }
  operator Audi() const {
    cout << "Transforming a Porsche into an Audi" << endl;
    return Audi{};
  }

};
```

```
operator class() { }
```

```
void test_explicit_cast() {

  Porsche p;
  p.get_info(); // Porsche

  Audi a = p;

  a->get_info(); // Audi

}
```

Class Design Principles

- SRP The Single Responsibility Principle
- OCP The Open/Close Principle
- LSP The Liskov Substitution Principle
- ISP The Interface Segregation Principle
- DIP The Dependency Inversion Principle

⇒ The S.O.L.I.D Principles

- Hundreds of pages, tutorials and videos on the web
  - https://leanpub.com/design-patterns-modern-cpp/read_sample

Single Responsibility Principle

```
class Book {
 public:
  string getTitle();
  string getAuthor();
  Position getLocation();
  bool printCurrentPage();
...
};
```

```
class Book {
 public:
  string getTitle();
  string getAuthor();
  ~~Position getLocation();~~
  string getText(int page);
...
};

class PlainTextPrinter: public Printer {
  void printPage(Book book, int page) {
     string txt = book.getText(page);
     cout << txt << endl;
  }
};

class BookLocator: public Db {
  Position locate(Book book) {
     return db.select(book.getTitle());
  }
};
```