

Modern C++ for Computer Vision and Image Processing

Lecture 5: I/O Files, Classes

Ignacio Vizzo and Cyrill Stachniss

C++ Utilities

C++ includes a variety of utility libraries that provide functionality ranging from bit-counting to partial function application.

`std::bitset`

`std::bind`

These libraries can be broadly divided into two groups:

- language support libraries.
- general-purpose libraries.

Language support

Provide classes and functions that interact closely with language features and support common language idioms.

- Type support(`std::size_t`).
- Dynamic memory management(`std::shared_ptr`).
*myint = -1;
assert(myint > 0);*
- Error handling(`std::exception`, `assert`).
↑
- Initializer list(`std::vector{1, 2}`).
*↑ =
 one in
 complex
 process*
- Much more...

General-purpose Utilities

- Program utilities(`std::abort`).
- Date and Time(`std::chrono::duration`).
- Optional, variant and any(`std::variant`).
- Pairs and tuples(`std::tuple`).
- Swap, forward and move(`std::move`).
- Hash support(`std::hash`).
- Formatting library(coming in C++20).
- Much more...

std::swap

```
1 int main() {  
2     int a = 3;  
3     int b = 5;  
4  
5     // before  
6     std::cout << a << ' ' << b << '\n';  
7  
8     std::swap(a, b);  
9  
10    // after  
11    std::cout << a << ' ' << b << '\n';  
12 }
```

Output:

```
1 3 5  
2 5 3
```

std::variant

```
1 int main() {          (int)    (float)
2     std::variant<int, float> v1;
3     v1 = 12; // v contains int
4     cout << std::get<int>(v1) << endl;
5     std::variant<int, float> v2{3.14F};
6     cout << std::get<1>(v2) << endl;
7
8     v2 = std::get<int>(v1); // assigns v1 to v2
9     v2 = std::get<0>(v1); // same as previous line
10    v2 = v1; // same as previous line
11    cout << std::get<int>(v2) << endl;
12 }
```

Output:

```
1 12
2 3.14
3 12
```

std::any

ബന്ധിക്കുന്ന ഡാറ്റ ടൈപ്
ബുൾട്ടിനിക്യൂ

```
1 int main() {  
2     std::any a; // any type  
3  
4     a = 1; // int  
5     cout << any_cast<int>(a) << endl;  
6  
7     a = 3.14; // double  
8     cout << any_cast<double>(a) << endl;  
9  
10    a = true; // bool  
11    cout << std::boolalpha << any_cast<bool>(a) << endl;  
12 }
```

Output:

```
1 1  
2 3.14  
3 true
```

std::optional

value ഫലമുണ്ട്. അവനുള്ളൂ



```
1 std::optional<std::string> StringFactory(bool create) {  
2     if (create) {  
3         return "Modern C++ is Awesome";  
4     }  
5     return {};  
6 }  
7  
8 int main() {  
9     cout << StringFactory(true).value() << '\n';  
10    cout << StringFactory(false).value_or(":(") << '\n';  
11 }
```

Output:

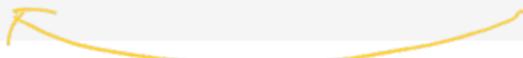
```
1 Modern C++ is Awesome  
2 :(
```

05

std::tuple

č, , 3

```
1 int main() {  
2     std::tuple<double, char, string> student1;  
3     using Student = std::tuple<double, char, string>;  
4     Student student2{1.4, 'A', "Jose"};  
5     PrintStudent(student2);  
6     cout << std::get<string>(student2) << endl;  
7     cout << std::get<2>(student2) << endl;  
8  
9     // C++17 structured binding:  
10    auto [gpa, grade, name] = make_tuple(4.4, 'B', "");  
11 }
```



Output:

```
1 GPA: 1.4, grade: A, name: Jose  
2 Jose  
3 Jose
```

std::chrono

+ wall-clock (real time clock)
off. angular movement
counting, P. measurements
regular intervals.

```
1 #include <chrono>
2
3 int main() {
4     auto start = std::chrono::steady_clock::now();
5     cout << "f(42) = " << fibonacci(42) << '\n';
6     auto end = chrono::steady_clock::now();
7
8     chrono::duration<double> sec = end - start;
9     cout << "elapsed time: " << sec.count() << "s\n";
10 }
```



Output:

```
1 f(42) = 267914296
2 elapsed time: 1.84088s
```

std::bind

(ഫീൽറ്റെഫ്: നാമം boost::bind ആ)
 കുട്ടിത്തും ലൈഖ്യിക്കും)

int f(int a, int b) എന്നും $a+b;$ } //എൻഡ്
 bind(f, -1, 5)(x) //അവന്തരം f(x, 5)
 bind(f, 5, -1)(x) // f(5, x) ആ

ഷുദ്ധിച്ചഭാഗങ്ങൾ placeholder അല്ല underscored f. argument എന്ന
 നീ ഫലം എന്നും അഭിപ്രായം ആണ്. (ദിവസം വിനാഹിനിനേം എന്നും)
 ഉദാ

bind(f, -2, -1)(x, y); $\Rightarrow f(y, x)$

bind(g, -1, 10, -1)(x); $\Rightarrow g(x, 10, x)$

bind(g, -3, -3, -3)(x, y, z); $\Rightarrow g(z, z, z)$

bind(g, -1, -1, -1)(x, y, z); $\Rightarrow g(x, x, x)$

Error handling with exceptions

- We can “**throw**” an exception if there is an error
අදාළ දය - catch දායකාරී සුදු.
- STL defines classes that represent exceptions. Base class: std::exception
තැන්.
- To use exceptions: #include <stdexcept>
- An exception can be “caught” at any point of the program (**try - catch**) and even “thrown” further (**throw**)
- The constructor of an exception receives a string error message as a parameter
- This string can be called through a member function **what()**
exception ඔබේ what() නිශ්චාලුවයි

throw exceptions

Runtime Error:

```
1 // if there is an error
2 if (badEvent) {
3     string msg = "specific error string";
4     // throw error
5     throw runtime_error(msg);
6 }
7 ... some cool code if all ok ...
```

try { ... }
catch (const exception& ex)
{ ex.what();
}

Logic Error: an error in logic of the user

```
1 throw logic_error(msg);
```

catch exceptions

- If we expect an exception, we can “catch” it
- Use `try - catch` to catch exceptions

```
1 try {  
2     // some code that can throw exceptions z.B.  
3     x = someUnsafeFunction(a, b, c);  
4 }  
5 // we can catch multiple types of exceptions  
6 catch ( runtime_error &ex ) {  
7     cerr << "Runtime error: " << ex.what() << endl;  
8 } catch ( logic_error &ex ) {  
9     cerr << "Logic error: " << ex.what() << endl;  
10 } catch ( exception &ex ) {  
11     cerr << "Some exception: " << ex.what() << endl;  
12 } catch ( ... ) { // all others  
13     cerr << "Error: unknown exception" << endl;  
14 }
```

Intuition

અધ્યાત્મિક સેવણી
Exception, throw અંશનું એવું હોય

- Only used for “exceptional behavior”
- Often misused: e.g. wrong parameter should not lead to an exception
- GOOGLE-STYLE Don’t use exceptions
- <https://en.cppreference.com/w/cpp/error>

Reading and writing to files

- Use streams from STL
- Syntax similar to `cerr`, `cout`

```
1 #include <fstream>
2 using std::string;
3 using Mode = std::ios_base::openmode;
4
5 // ifstream: stream for input from file
6 std::ifstream f_in(string& file_name, Mode mode);
7
8 // ofstream: stream for output to file
9 std::ofstream f_out(string& file_name, Mode mode);
10
11 // stream for input and output to file
12 std::fstream f_in_out(string& file_name, Mode mode);
```

in, out - --

There are many modes under which a file can be opened

Mode	Meaning
ios_base::app ✓	append output
ios_base::ate ✓	seek to EOF when opened
ios_base::binary	open file in binary mode
ios_base::in ✓	open file for reading
ios_base::out ✓	open file for writing
ios_base::trunc	overwrite the existing file

Mode of opening file
fstream obj;

obj.open("file",ios::in|ios::out);

Curser pointer position of file.

{

obj.seekg() // Seek get

obj.seekp() // Seek put

16

Regular columns

column by column var
உடலில் ஒரு கூற்றைப் பதிலளிப்பது

Use it when:

- The file contains organized data
- Every line has to have all columns

```
1 1 2.34 One 0.21
2 2 2.004 two 0.23
3 3 -2.34 string 0.22
```

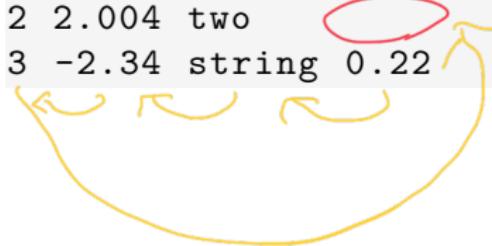
O.K.

```
1 1 2.34 One 0.21
2 2 2.004 two 0.23
3 3 -2.34 string 0.22
```

Int float string float

Fail

```
1 1 2.34 One 0.21
2 2 2.004 two 0.23
3 3 -2.34 string 0.22
```



Reading from ifstream

```
1 #include <fstream> // For the file streams.  
2 #include <iostream>  
3 #include <string>  
4 using namespace std; // Saving space.  
5 int main() {  
6     int i;  
7     double a, b;  
8     string s;  
9     // Create an input file stream.  
10    ifstream in("test_cols.txt", ios_base::in);  
11    // Read data, until it is there.  
12    while (in >> i >> a >> s >> b) {  
13        cout << i << ", " << a << ", "  
14        << s << ", " << b << endl;  
15    }  
16    return (0);  
17 }
```

ମେଟୋଡ ରୂପ

ତାଙ୍କୁ ଦେଖନ୍ତୁ-

Reading files one line at a time

- Bind every line to a `string`
- Afterwards parse the string

```
1 =====  
2 HEADER  
3 a = 4.5  
4 filename = /home/ivizzo/.bashrc  
5 =====  
6 2.34  
7 1 2.23  
8 ER SIE ES
```

line by line ~~com~~

std::string::npos is static member constant ടെക്സ്റ്റ്.

static const size_t npos = -1;

തുല്യം

നിര്

position എന്ന്



```
1 #include <fstream>           static const size_t npos = -1;
2 #include <iostream>
3 using namespace std;
4 int main() {
5     string line, file_name;
6     ifstream input("test_bel.txt", ios_base::in);
7     // Read data line-wise.
8     while (getline(input, line)) {
9         cout << "Read: " << line << endl;
10        // String has a find method.
11        string::size_type loc = line.find("DeepBlue", 0);
12        if (loc != string::npos) {
13            file_name = line.substr(line.find("=", 0) + 1,
14                                      string::npos);
15        }
16    }
17    cout << "Filename found: " << file_name << endl;
18    return (0);
19 }
```

find(" ", 1) position 1 അനുസരിച്ച്.

11-getline.cpp

Writing into text files

With the same syntax as `cerr` und `cout` streams, with `ofstream` we can write directly into files

```
1 #include <iomanip> // For setprecision.  
2 #include <fstream>  
3 using namespace std;  
4 int main() {  
5     string filename = "out.txt";  
6     ofstream outfile(filename);  
7     if (!outfile.is_open()) { return EXIT_FAILURE; }  
8     double a = 1.123123123; ← 20  
9     outfile << "Just string" << endl;  
10    outfile << setprecision(20) << a << endl;  
11    return 0;  
12 }
```

Writing to binary files

binary මුද්‍රණ
යෝගාත්.

- We write a sequence of bytes
- We must document the structure well, otherwise none can read the file
- Writing/reading is fast
- No precision loss for floating point types
- Substantially smaller than ascii-files
- **Syntax**

```
1 file.write(reinterpret_cast<char*>(&a), sizeof(a));
```

(Binary සඳහා මූලික මෙහෙයුම් ප්‍රතිඵලිත කළ යුතුවේ.)

Writing to binary files

(\n log)

```
1 #include <fstream> // for the file streams
2 #include <vector>
3 using namespace std;
4
5 int main() {
6     string file_name = "image.dat";
7     ofstream file(file_name, ios_base::out | ios_base::binary);
8     int rows = 2;
9     int cols = 3;
10    vector<float> vec(rows * cols);
11    file.write(reinterpret_cast<char*>(&rows), sizeof(rows));
12    file.write(reinterpret_cast<char*>(&cols), sizeof(cols));
13    file.write(reinterpret_cast<char*>(&vec.front()),
14                           vec.size() * sizeof(float));
15    return 0;
16 }
```

Casting → static_cast
const_cast
reinterpret_cast

13

Reading from binary files

- We read a **sequence of bytes**
- Binary files are not human-readable
- We must know the structure of the contents
- **Syntax**

```
1 file.read(reinterpret_cast<char*>(&a), sizeof(a));
```

Bylc sequence & structure mwg read nqponhaq.

Reading from binary files

```
1 #include <fstream>
2 #include <iostream>
3 #include <vector>
4 using namespace std;
5 int main() {
6     string file_name = "image.dat";
7     int r = 0, c = 0;
8     ifstream in(file_name,
9                  ios_base::in | ios_base::binary);
10    if (!in) { return EXIT_FAILURE; }
11    in.read(reinterpret_cast<char*>(&r), sizeof(r));
12    in.read(reinterpret_cast<char*>(&c), sizeof(c));
13    cout << "Dim: " << r << " x " << c << endl;
14    vector<float> data(r * c, 0);
15    in.read(reinterpret_cast<char*>(&data.front()),
16            data.size() * sizeof(data.front()));
17    for (float d : data) { cout << d << endl; }
18    return 0;
19 }
```

Important facts

Pros

- I/O Binary files is faster than ASCII format.
- Size of files is drastically smaller.
- There are many libraries to facilitate serialization.

Cons

- Ugly Syntax.
- File is not readable by human.
- You need to know the format before reading.
- You need to use this for your homeworks.

C++17 Filesystem library

- Introduced in C++17.
- Use to perform operations on:
 - paths
 - regular files
 - directories
- Inspired in `boost::filesystem`
- Makes your life easier.
- <https://en.cppreference.com/w/cpp/filesystem>

directory_iterator

```
1 #include <filesystem>
2 namespace fs = std::filesystem;
3
4 int main() {
5     fs::create_directories("sandbox/a/b");
6     std::ofstream("sandbox/file1.txt");
7     std::ofstream("sandbox/file2.txt");
8     for (auto& p : fs::directory_iterator("sandbox")) {
9         std::cout << p.path() << '\n';
10    }
11    fs::remove_all("sandbox");
12 }
```

Output:

```
1 "sandbox/a"
2 "sandbox/file1.txt"
3 "sandbox/file2.txt"
```

filename_part1

```
1 #include <filesystem>
2 namespace fs = std::filesystem;
3
4 int main() {
5     cout << fs::path("/foo/bar.txt").filename() << '\n'
6         << fs::path("/foo/.bar").filename() << '\n'
7         << fs::path("/foo/bar/").filename() << '\n'
8         << fs::path("/foo/.").filename() << '\n'
9         << fs::path("/foo/..").filename() << '\n';
10 }
```

Output:

```
1 "bar.txt"
2 ".bar"
3 ""
4 "."
5 ".."
```

filename_part2

```
1 #include <filesystem>
2 namespace fs = std::filesystem;
3
4 int main() {
5     cout << fs::path("/foo/.bar").filename() << '\n'
6         << fs::path(".").filename() << '\n'
7         << fs::path("../").filename() << '\n'
8         << fs::path("//").filename() << '\n'
9         << fs::path("//host").filename() << '\n';
10 }
```

Output:

```
1 ".bar"
2 "."
3 ".."
4 ""
5 "host"
```

extension_part1

```
1 #include <filesystem>
2 namespace fs = std::filesystem;
3
4 int main() {
5     cout << fs::path("/foo/bar.txt").extension() << '\n'
6         << fs::path("/foo/bar.").extension() << '\n'
7         << fs::path("/foo/bar").extension() << '\n'
8         << fs::path("/foo/bar.png").extension() << '\n';
9 }
```

Output:

```
1 ".txt"
2 "."
3 ""
4 ".png"
```

extension_part2

```
1 #include <filesystem>
2 namespace fs = std::filesystem;
3
4 int main() {
5     cout << fs::path("/foo/.").extension() << '\n'
6         << fs::path("/foo/..").extension() << '\n'
7         << fs::path("/foo/.hidden").extension() << '\n'
8         << fs::path("/foo/..bar").extension() << '\n';
9 }
```

Output:

```
1 ""
2 ""
3 ""
4 ".bar"
```

stem

```
1 #include <filesystem>
2 namespace fs = std::filesystem;
3
4 int main() {
5     cout << fs::path("/foo/bar.txt").stem() << endl
6         << fs::path("/foo/00000.png").stem() << endl
7         << fs::path("/foo/.bar").stem() << endl;
8 }
```

file name only w. dot no ext.

dot dot separator
string operation
return value

Output:

```
1 "bar"
2 "00000"
3 ".bar"
```

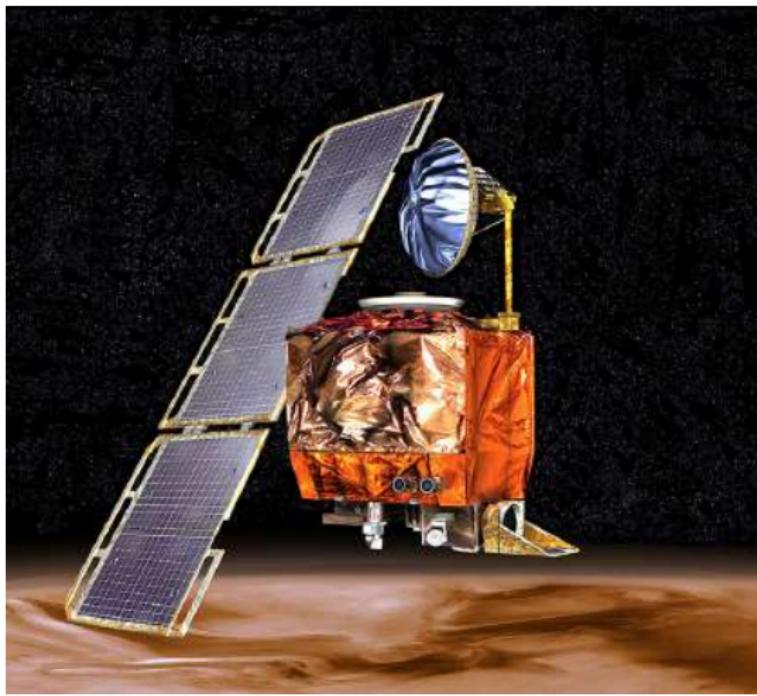
exists

```
1 void demo_exists(const fs::path& p) {
2     cout << p;
3     if (fs::exists(p))    cout << " exists\n";
4     else                  cout << " does not exist\n";
5 }
6
7 int main() {
8     fs::create_directory("sandbox");
9     ofstream("sandbox/file"); // create regular file
10    demo_exists("sandbox/file");
11    demo_exists("sandbox/cacho");
12    fs::remove_all("sandbox");
13 }
```

Output:

```
1 "sandbox/file" exists
2 "sandbox/cacho" does not exist
```

Types are indeed important



<https://www.simscale.com/blog/2017/12/nasa-mars-climate-orbiter-metric/>

Type safety

bad – the unit is ambiguous

```
1 void blink_led_bad(int time_to_blink) {  
2     // do something with time_to_blink  
3 }
```

integer or float?
what?

- What if I call `blink_led_bad()` with wrong units? ✓
- When I will detect the error? (int 1000) ✓

good – the unit is explicit

```
1 void blink_led_good(milliseconds time_to_blink) {  
2     // do something with time_to_blink  
3 }
```

ms
3700 ms

Type safety

good – the unit is explicit

```
1 void blink_led_good(milliseconds time_to_blink) {  
2     // do something with time_to_blink  
3 }
```

Usage

```
1 void use() {  
2     blink_led_good(100);           // ERROR: What unit? ✗  
3     blink_led_good(100ms);        // ✓  
4     blink_led_good(5s);          // ERROR: Bad unit ✗  
5 }
```

Example taken from: <https://youtu.be/fX2W3nNjJIo>

Want more flexibility?

lec10c9

data type ප්‍රතිඵලිත සාක්ෂිය
මෙහි template සෑයුත්
සැස තුළුවුනු flexible
අභ්‍යන්තරය

```
1 template <class rep, class period>
2 void blink_led(duration<rep, period> blink_time) {
3     // millisecond is the smallest relevant unit
4     auto x_ms = duration_cast<milliseconds>(blink_time);
5     // do something else with x_ms
6 }
7
8 void use() {
9     blink_led(2s);           // Works fine ✓
10    blink_led(150ms);       // Also, works fine ✓
11    blink_led(150);         // ERROR, which unit? ✗
12 }
```

Type safety in our field

BAD Example: ROS 1

```
1 // ...
2 //
3 // %Tag(LOOP_RATE)%
4 ros::Rate loop_rate(10);
5 // %EndTag(LOOP_RATE)%
6 //
7 // ...
```

int type ඇ කුඩා තැන්ති වෙයෙහි
මිදු යොගා යුතු හෝ

while (ros::ok)
{ // your code
loop_rate.sleep();}

loop_rate in which units? Hz, ms ???

මාශ්‍ය පිළිබඳ දැනු නොවේම හැඳුව
type නෑ මෙහෙමත් ගෙවූ තොරතුව නැතුවා

https://github.com/ros/ros_tutorials/blob/noetic-devel/roscpp_tutorials/talker/talker.cpp

Type safety in our field

GOOD Example: ROS 2

```
1 // ...
2 //
3 timer_ = create_wall_timer(100ms, timer_callback);
4 //
5 // ...
```

- Same functionality as previous example
- Better code, better readability
- Safer
- Guaranteed to run every 100ms(10 Hz)

Class Basics

"C++ classes are a tools for creating new types that can be used as conveniently as the built-in types. In addition, derived classes and templates allow the programmer to express relationships among classes and to take advantage of such relationships."

Class Basics

class car:

{

toyota;

bmw;

suzuki;

model;

brand;

call();



3

"A type is a concrete representation of a **concept** (an idea, a notion, etc.). A program that provides types that closely match the concepts of the application tends to be easier to understand, easier to reason about, and easier to modify than a program that does not."

defn {

model;

brand;

call();



Phone samsung;

Phone apple;

Phone htc;

3

Class Basics

```
class x { int a; }  
          ^ data member  
          help(); }  
          ^ member function  
          }  
          }
```

- A class is a user-defined type
- A class consists of a set of members. The most common kinds of members are data members and member functions
- Member functions can define the meaning of initialization (creation), copy, move, and cleanup (destruction)
- Members are accessed using . (dot) for objects and -> (arrow) for pointers

samsung.call();

iphone->call

Class Basics

```
public:  
int a;  
private:  
int b; }
```

- Operators, such as `+`, `!`, and `[]`, can be defined for a `class`
- A class is a namespace containing its members အုပိုင်း namespace ၏ ဖော်ပြန်မှု
- The public members provide the class's interface and the private members provide implementation details
- A struct is a class where members are by default public

Example class definition

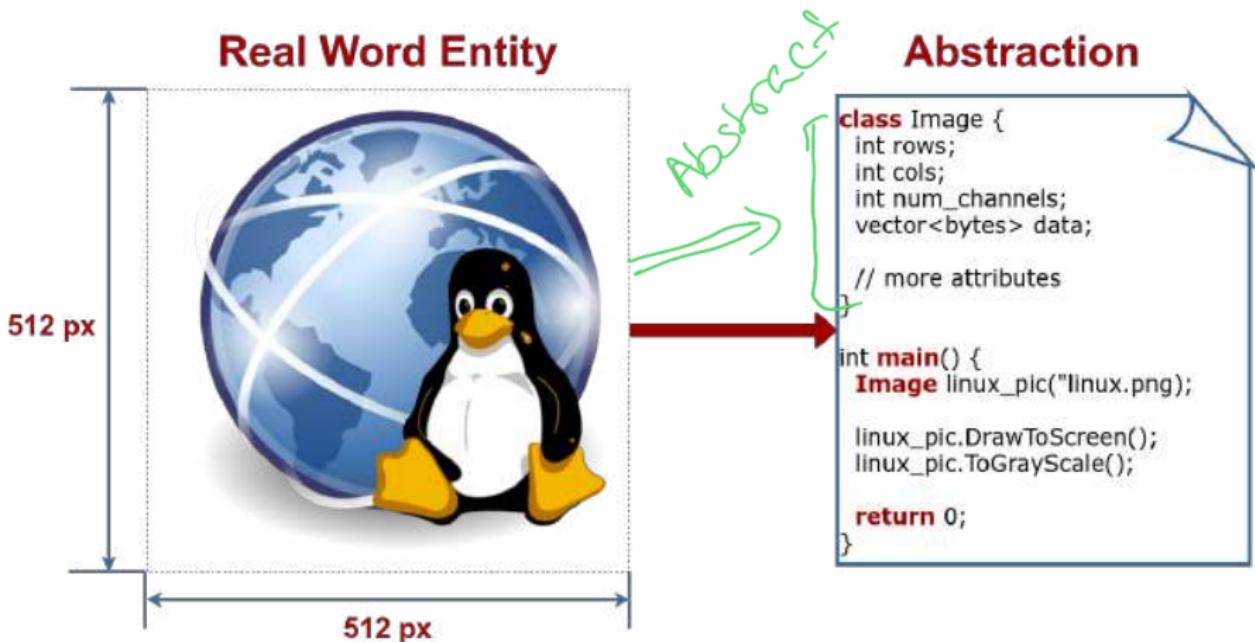
```
1 class Image { // Should be in Image.hpp
2 public:
3     Image(const std::string& file_name);
4     void Draw();
5
6 private:
7     int rows_ = 0; // New in C++11
8     int cols_ = 0; // New in C++11
9 };
10
11 // Implementation omitted here, should be in Image.cpp
12 int main() {
13     Image image("some_image.pgm");
14     image.Draw();
15     return 0;
16 }
```

Classes in our field

Inheritance
Child class (Derived class)
Parent class (Base class)

```
1 // 2D entities
2 class Image : public Geometry2D;
3 class RGBDImage : public Geometry2D;
4
5 // 3D entities
6 class Image : public Geometry2D;
7 class OrientedBoundingBox : public Geometry3D;
8 class AxisAlignedBoundingBox : public Geometry3D;
9 class LineSet : public Geometry3D;
10 class MeshBase : public Geometry3D;
11 class Octree : public Geometry3D;
12 class PointCloud : public Geometry3D;
13 class VoxelGrid : public Geometry3D;
14
15 // 3D surfaces
16 class TetraMesh : public MeshBase;
17 class TriangleMesh : public MeshBase;
```

Image class



One possible realization

Open3D::Geometry::Image

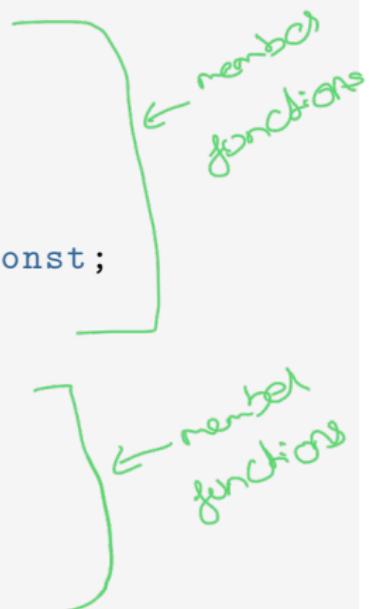
```
1 class Image : public Geometry2D {  
2     public:  
3         /// Width of the image.  
4         int width_ = 0;  
5         /// Height of the image.  
6         int height_ = 0;  
7         /// Number of channels in the image.  
8         int num_of_channels_ = 0;  
9         /// Image storage buffer.  
10        std::vector<uint8_t> data_;  
11    };
```

public data members

One possible realization

Open3D::Geometry::Image

```
1 class Image : public Geometry2D {  
2     public:  
3         void Clear() const override;  
4         bool IsEmpty() const override;  
5  
6         Image FlipHorizontal() const;  
7         Image FlipVertical() const;  
8         Image Filter(Image::FilterType type) const;  
9  
10    protected:  
11        void AllocateDataBuffer() {  
12            data_.resize(width_ *  
13                            height_ *  
14                            num_of_channels_);  
15        }  
16    }
```



C++ Visibility Modes

Public If we derive a class from a public base class, then the public members of the parent class will become public in the derived class and protected member of the base class will become protected in derived class.

Protected If we derive a child-class from a protected-base class, Then both public members and protected member of a base class will become protected in derived class.

Private If we derive a subclass from a private base class, both public & protected members of the base class will become private in derived class:

class B: public A

class B: protected A

{ };

class B : private A

{ };

Public :

ජ්‍යාමිත්‍ය මෙහෙයුම් හේතුවෙන් ප්‍රතිච්‍යාම් යොමු කළ නිශ්චිතයෙන් පෙන්වනු ලබයි. Open to all සේ. derived class, other class තුළ ඇති.

Protected :

ක්‍රියා මෙහෙයුම් derived class තුළ නොවා මැතිස් නැතුවෙන් ප්‍රතිච්‍යාම් යොමු කළ නිශ්චිතයෙන් පෙන්වනු ලබයි.

Private :

ක්‍රියා මෙහෙයුම් සෙවී ඇති නිශ්චිතයෙන් පෙන්වනු ලබයි. Class / Object නැගුණු නොවීමෙන් නොවා නොවූ නිශ්චිතයෙන් පෙන්වනු ලබයි.

Must Watch

Bag of Visual Words Introduction



<https://youtu.be/a4cFONdc6nc>

Suggested Video

Features Descriptors



<https://youtu.be/CMolhcwtGAU>

References

- **Utility Library**

<https://en.cppreference.com/w/cpp/utility>

- **Error handling**

<https://en.cppreference.com/w/cpp/error>

- **IO Library**

<https://en.cppreference.com/w/cpp/io>

- **Filesystem Library**

<https://en.cppreference.com/w/cpp/filesystem>

- **Classes**

<https://en.cppreference.com/w/cpp/classes>