Introduction to Object-Oriented Programming with C++

Olivier Mattelaer
UCLouvain
CP3 & CISM
Programming paradigm

Paradigm = style of computer programming

- **Procedural languages:**
  - Describe step by step the procedure that should be followed to solve a specific problem.

- **Declarative programming**
  - The computer is told what the problem is, not how to solve the problem.

- **Object-oriented programming:**
  - Data and methods of manipulating data are kept as single unit called object.
  - A user can access the data via the object’s method.
  - The internal working of an object maybe changed without any code that uses the object.
## Why C++

### Tiobe Ranking

<table>
<thead>
<tr>
<th>Oct 2017</th>
<th>Oct 2016</th>
<th>Change</th>
<th>Programming Language</th>
<th>Ratings</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td>Java</td>
<td>12.431%</td>
<td>-6.37%</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td>C</td>
<td>8.374%</td>
<td>-1.46%</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
<td>C++</td>
<td>5.007%</td>
<td>-0.79%</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
<td>C#</td>
<td>3.858%</td>
<td>-0.51%</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td></td>
<td>Python</td>
<td>3.803%</td>
<td>+0.03%</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td></td>
<td>JavaScript</td>
<td>3.010%</td>
<td>+0.26%</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td></td>
<td>PHP</td>
<td>2.790%</td>
<td>+0.05%</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td></td>
<td>Visual Basic .NET</td>
<td>2.735%</td>
<td>+0.08%</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>↑</td>
<td>Assembly language</td>
<td>2.374%</td>
<td>+0.14%</td>
</tr>
<tr>
<td>10</td>
<td>13</td>
<td>↑</td>
<td>Ruby</td>
<td>2.324%</td>
<td>+0.32%</td>
</tr>
</tbody>
</table>

- Extension of C (originally called “C with Classes”)
- Compiled, high level language, strongly-typed unsafe language, static and dynamic type checking, supports many paradigm, is portable
Program of today

• Basic of C++
  ➡ Presentation of concept
  ➡ Code presentation
  ➡ Exercise

• Introduction to Class in C++
  ➡ Presentation of concept
  ➡ Code presentation
  ➡ Exercise

• (Multi) Inheritance
  ➡ Presentation of concept
  ➡ Code presentation
  ➡ Exercise
Hello World

- **Line 1**: Comment
  - also /* … */

- **Line 2**: Preprocessor directive:
  - Include a section of standard C++ code in the code

- **Line 3**: Empty line: do nothing (but clarity for human reader)

- **Line 4**: Declaration of a function
  - `main` is a special function which is run automatically
  - starts and stops with the braces (line 5 and 7)

- **Statement**: Send character to the output device
  - Note the semi-column at the end of the line
Compile the code

**C++**

Hmem/linux

```
g++ -o EXECNAME input.cpp
```

Mac

```
g++ -o EXECNAME input.cpp
```

**C++11**

Hmem/linux

```
g++ -std=c++11 -o EXECNAME input.cpp
```

Mac

```
clang++ -std=c++11 -stdlib=libc++ -o EXECNAME input.cpp
```

**Problem**

https://ideone.com/

- Select C++ (bottom left)

http://www.cpp.sh/2dd

**Problem**

https://ideone.com/

- Select C++14 (bottom left)

http://www.cpp.sh/2dd
Basic of C++ : variables

- C++ is strongly typed
  - Need to know the type of variable
  - The type determines the size of the house

### Variable = portion of memory storing a value

#### C++11

**Variable Types**

<table>
<thead>
<tr>
<th>Group</th>
<th>Type names*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character types</td>
<td>char, char16_t, char32_t, wchar_t</td>
</tr>
<tr>
<td>Integer types (signed)</td>
<td>signed char, signed short int, signed int, signed long int, signed long long int</td>
</tr>
<tr>
<td>Integer types (unsigned)</td>
<td>unsigned char, unsigned short int, unsigned int, unsigned long int, unsigned long long int</td>
</tr>
<tr>
<td>Floating-point types</td>
<td>float, double, long double</td>
</tr>
<tr>
<td>Boolean type</td>
<td>bool</td>
</tr>
<tr>
<td>Void type</td>
<td>void</td>
</tr>
<tr>
<td>Null pointer</td>
<td>decltype(nullptr)</td>
</tr>
</tbody>
</table>

```cpp
// initialization of variables
#include <iostream>
using namespace std;

int main ()
{
    int a=5; // initial value: 5
    int b(3); // initial value: 3
    int c(2); // initial value: 2
    int result; // initial value undetermined
    a = a + b;
    result = a - c;
    cout << result;
    return 0;
}
```

```cpp
// my first string
#include <iostream>
#include <string>
using namespace std;

int main ()
{
    string mystring;
    mystring = "This is a string";
    cout << mystring;
    return 0;
}
```

http://cpp.sh/8yl
http://cpp.sh/7d4
Basic of C++: pointer

Pointer = position in memory of the variable

Due to dereference, pointers also have typed:

- Those are the type of the variable suffix by a star

```c
int * number;
char * character;
double * decimals;
```
Basic of C++: functions

Function = group of statements
- that is given a name,
- which can be called from some point of the program

Passing Parameters by Variable

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

int addition (int a, int b)
{
    int r;
    r=a+b;
    return r;
}

int main ()
{
    int z;
    z = addition (5,3);
    cout << "The result is " << z;
}
```

Passing Parameters by reference

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

void duplicate (int& a, int& b, int& c)
{
    a*=2;
    b*=2;
    c*=2;
}

int main ()
{
    int x=1, y=3, z=7;
    duplicate (x, y, z);
    cout << "x=" << x << " , y=" << y << " , z=" << z;
    return 0;
}
```
Basic of C++: Array

Array = sequential memory space of the same type

```cpp
int foo [5] = { 16, 2, 77, 40, 12071 };
```

- Note the syntax to receive array in a function!
- Array behaves like pointer!
Exercise 1

• Check that you can compile the Hello World example

• Define a function that take 3 float and return the average

• For an array of integer and a given value.
  ➡ Return the pointer where this value is.
  ➡ Use this pointer to get the value of the next two entry of the array
  ➡ Example {1,2,3,4,5} and val=3 -> should return 4/5

• Have Fun
  ➡ Useful resources:
    ✦ http://www.cplusplus.com/reference
    ✦ http://www.cplusplus.com/doc/tutorial/
Solution

```cpp
// function example
#include <iostream>
using namespace std;

int* cut_before_val ( int sequence[], int val) { 
    int i = 0;
    while(true){
        if(sequence[i] == val){
            return &sequence[i];
        }
        i++;
    }
}

int main (){ 
    int a[]={1,2,3,4,5};
    int* z;
    z = cut_before_val(a,3);
    cout << "The result is " << z[1] << "next" << z[2]<<endl;
    cout << "The result is " << *(++z) << "next" << *(++z)<<endl;
}
```
Classes

- We can define a **class** “Car”
  - Defines the structure
    - Which property available: **attribute**
      - model, colour, has_autodrive, nb_door
    - Which function can be applied.
      - change_battery, add_fuel,…
  - Class is a new type like “int/float”
    - Car mytesla;
    - “mytesla” is an **instance** of the class CAR

```cpp
class Rectangle {
  int width, height;
  public:
    void set_values (int, int);
    int area (void);
} rect;
```
## Visibility of attribute/function

<table>
<thead>
<tr>
<th></th>
<th>private</th>
<th>protected</th>
<th>public</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessible</td>
<td>Only accessible from other instance of the same class</td>
<td>Accessible from other instance of the same class</td>
<td>Accessible from everywhere where the object is visible</td>
</tr>
<tr>
<td>Accessible</td>
<td>Accessible from friends</td>
<td>Accessible from friends</td>
<td>READ and WRITE!</td>
</tr>
<tr>
<td>Accessible</td>
<td>Accessible from instance of the derived/child class</td>
<td>Accessible from instance of the derived/child class</td>
<td></td>
</tr>
<tr>
<td>DEFAULT</td>
<td></td>
<td>DEFAULT</td>
<td></td>
</tr>
</tbody>
</table>

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

class Rectangle{
    private:
        int width, height;
};

int main(){
    Rectangle A;
    A.width = 3;
    A.height = 2;
    cout << "width" << A.width << endl;
};
```

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

class Rectangle{
    public:
        int width, height;
};

int main(){
    Rectangle A;
    A.width = 3;
    A.height = 2;
    cout << "width" << A.width << endl;
};
```

```
simple.cpp:11:5: error: 'width' is a private member of 'Rectangle'
  A.width = 3;
^`
```
First Example

http://cpp.sh/8ac

```cpp
// example: one class, two objects
#include <iostream>
using namespace std;

class Rectangle {
    int width, height;
    public:
        void set_values (int, int);
        int area () {return width*height;}
};

void Rectangle::set_values (int x, int y) {
    width = x;
    height = y;
}

int main () {
    Rectangle rect, rectb;
    rect.set_values (3,4);
    rectb.set_values (5,6);
    cout << "rect area: " << rect.area() << endl;
    cout << "rectb area: " << rectb.area() << endl;
    return 0;
}
```

• width/height are private
• A public function allows to set those values!
• private attribute ensure that no one mess up those variables.
Code Structure

```
#include "CLASS1.H"
......

class Class1{
     .........
};
```

```
#include "CLASS2.H"
......

class Class2{
     .........
};
```

```
#include "CLASS1.H"
#include "CLASS2.H"
......

void main()
{
     .........
}
```
Constructor

constructor = function called after the object is created

- The name of the constructor is the name of the function itself!

```cpp
// example: class constructor
#include <iostream>
using namespace std;

class Rectangle {
    int width, height;
    public:
        Rectangle (int,int);
        int area () {return (width*height);}
    }

Rectangle::Rectangle (int a, int b) {
    width = a;
    height = b;
}

int main () {
    Rectangle rect (3,4);
    Rectangle rectb (5,6);
    cout << "rect area: " << rect.area() << endl;
    cout << "rectb area: " << rectb.area() << endl;
    return 0;
}
```

- Shortcut for setting attribute

```cpp
Rectangle::Rectangle (int x, int y) : width(x), height(y) {} 

Rectangle::Rectangle (int x, int y) : width(x) { height=y; }
```
Overloading

Overloading = more than one function with the same name

- The name of two functions **CAN** be the same if the number of argument or the type of argument are different.

```cpp
// example: class constructor
#include <iostream>
using namespace std;

class Rectangle {
    int width, height;
    public:
        Rectangle(int, int);
        Rectangle(int l): width(l), height(l) {};
    int area () {return width*height;}
};

Rectangle::Rectangle(int a, int b) {
    width = a;
    height = b;
}

int main () {
    Rectangle rect (3);
    Rectangle rectb (5,6);
    cout << "rect area: " << rect.area() << endl;
    cout << "rectb area: " << rectb.area() << endl;
    return 0;
}
```

- Any function can be overloaded.
- You can overload basic operation between object like addition:
  - Operator +
Overloading

Overloading = more than one function with the same name

<table>
<thead>
<tr>
<th>Overloadable operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>+  -  *  /  =  &lt;  &gt;  +=  -=  *=  /=  &lt;&lt;=  &gt;&gt;=</td>
</tr>
<tr>
<td>&lt;=  &gt;=  ==  !=  &lt;=  &gt;=  ++  --  %  &amp;  ^  !</td>
</tr>
<tr>
<td>~  &amp;=  ^=</td>
</tr>
</tbody>
</table>

delete new[] delete[]

cpp.sh/271

```cpp
// overloading operators example
#include <iostream>
using namespace std;

class CVector {
    public:
        int x,y;
        CVector () {}
        CVector (int a, int b) : x(a), y(b) {}
        CVector operator+ (const CVector&);
    }

    CVector CVector::operator+ (const CVector& param) {
        CVector temp;
        temp.x = x + param.x;
        temp.y = y + param.y;
        return temp;
    }

    int main () {
        CVector foo (3,1);
        CVector bar (1,2);
        CVector result;
        result = foo + bar;
        cout << result.x << ',' << result.y << '\n';
        return 0;
    }
```
Special members

Special members = member functions implicitly defined

- Default constructor:
  - Present only if no other constructor exists!

- Destructor ~CLASSNAME:
  - Perform cleanup (remove dynamically allocated memory) when the object is deleted/out of scope

- Copy Constructor:
  - Called when you call that class (by value) in a function.
  - Perform shallow copy of all attributes

```cpp
MyClass::MyClass(const MyClass& x) : a(x.a), b(x.b), c(x.c) {}
```
```cpp
// example: class constructor
#include <iostream>
using namespace std;

class Rectangle {
    int width, height;
    public:
    Rectangle();
    Rectangle (int, int);
    Rectangle (int a, int b, int c): Rectangle(a,b){cout << c << endl;};
    Rectangle (int l){width=l; height=l;};
    Rectangle(const Rectangle& x){width=x.width; height=x.height; cout<<"copy " <<x.width<<" " <<x.height<<endl;};
    int area () {return (width*height);}
    Rectangle intersection(Rectangle);
};

Rectangle::Rectangle (int a, int b) {
    width = a;
    height = b;
}

Rectangle Rectangle::intersection(Rectangle B){
    //returns a rectangle with the smallest width and height
    Rectangle out;
    if (width < B.width){
        out.width = width;
    }else{
        out.width = B.width;
    }
    if (height < B.height){
        out.height = height;
    }else{
        out.height = B.height;
    }
    return out;
}

int main () {
    Rectangle rect (3);
    Rectangle rectb (2,6,30);
    Rectangle small = rect.intersection(rectb);
    cout << "rect area: " << rect.area() << endl;
    cout << "small area: " << small.area() << endl;
    return 0;
}
```
Exercise II

• Create a class for three dimensional vector
• Define function to get/set each component
• Define a function returning the norm(squared) of the vector
  ➡ $x[0]**2 + x[1]**2 + x[2]**2$
• Define the scalar product between two vector:
  ➡ $x[0]*y[0] + x[1]*y[1] + x[2]*y[2]$

• Define a Class parallelogram
  ➡ Can be initialised by two vector
  ➡ Set a function to compute the associated area
Solution

```cpp
// example: ThreeVector
#include <iostream>
#include <math.h>
using namespace std;

class ThreeVector{
  float v[3];

public:
  ThreeVector(){};
  ThreeVector(float x, float y, float z){ v[0]=x; v[1]=y; v[2]=z;};
  float get_x(){return v[0];};
  float get_y(){return v[1];};
  float get_z(){return v[2];};
  void set_x(float x){v[0] = x;};
  void set_y(float y){v[1] = y;};
  void set_z(float z){v[2] = z;};
  float norm(){return sqrt(v[0]*v[0]+v[1]*v[1]+v[2]*v[2]);};
  float operator *(const ThreeVector& y){return v[0]*y.v[0] + v[1]*y.v[1] +v[2]*y.v[2];};
};

int main () {
  ThreeVector a(1,2,3);
  ThreeVector b(1,0,0);
  cout << "norm a" << a.norm() << endl;
  cout << "norm b" << b.norm() << endl;
  cout << "a*b=" << a*b << endl;
}
```
```cpp
class Parallelogram{
    ThreeVector first;
    ThreeVector second;
    public:
        Parallelogram(ThreeVector f, ThreeVector second): first(f), second(second){};
        float get_area() {return first*second;}
};

int main () {
    ThreeVector a(1,2,3);
    ThreeVector b(1,0,0);
    cout << "norm a " << a.norm() << endl;
    cout << "norm b " << b.norm() << endl;
    cout << "a*b= " << a*b << endl;
    Parallelogram P(a,b);
    cout << "area of parallelogram " << P.get_area()<<endl;
}
```
Inheritance

Inheritance = new classes which retain characteristics of the base class.

- The idea is the heritage. What a parent can do, their child can do it too.

```cpp
// example: class constructor
#include <iostream>
using namespace std;

class Mother{
public:
    void hello()
    {
        cout<< "hello from Mother"<<endl;}
};
class Child1: public Mother{}
class Child2: public Mother{
public:
    void hello() {
        Mother::hello();
        cout<< "and from Child2" << endl;}
};

int main () {
    Child1 test;
    test.hello();
    Child2 test2;
    test2.hello();
}
```
Inheritance

Inheritance = new classes which retain characteristics of the base class.

• The idea is the heritage. What a parent can do, their child can do it too.

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

class Mother{
public:
    void hello()
    { cout<< "hello from Mother"<<endl; }
};

class Child1: public Mother{};

class Child2: public Mother{
public:
    void hello() {
        Mother::hello();
        cout<< "and from Child2" << endl; }
    
};

int main () {
    Child1 test;
    test.hello();

    Child2 test2;
    test2.hello();
}
```

• “public” tells the maximum level of visibility of the attribute coming from the base class
• Private argument are not passed to the child (but they still exits!)
• Constructor/Destructor are not passed to the child
• Assignment operator (operator =) are not passed to the child
Multi-inheritance

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

class Mother{
public:
    void hello()
    {
        cout << "hello from Mother" << endl;
    }
};

class Father{
protected:
    int age;
public:
    Father(){}
    Father(int x): age(x){}
};

class Child: public Mother, public Father{
public:
    Child(int x){age=x;}
    void print() {hello(); cout<<"my age is " << age;}
    void set_age(int x){age=x;}
};

int main () {
    Child test(3);
    test.hello();
    test.print();
    test.set_age(4);
    test.print();
}
```
Multi-inheritance

```cpp
// example: class constructor
#include <iostream>
using namespace std;

class Mother{
 public:
  void hello(){
    cout << "hello from Mother" << endl;}
};

class Father{
  int age;
 public:
  Father(){};
  Father(int x): age(x){};
  void set_age(int x){age=x;};
  int get_age(){return age;};
};

class Child: public Mother, public Father{
 public:
  Child(int x){set_age(x);};
  void print() {hello(); cout<<"my age is " << get_age();}
};

int main () {
  Child test(3);
  test.hello();
  test.print();
  test.set_age(4);
  test.print();
}
```

Father

- Age (priv)
- set_age()
- get_age()

Inherit from

Child

- print()

Can call hello()

Can not call age (since private)
But can call the public routine of father which set/get the age variable
Exercise III

- Define a class Four-Vector which inherit from your class 3 vector
  - Define the norm like in special relativity

- Define a class ParticleInfo
  - Has some attribute (mass/width)

- Define a class Particle which inherit from both class
  - Define a function which computes the difference between the mass square and the norm squared.
class ThreeVector{
    protected:
        float v[3];

    public:
        ThreeVector{};
        ThreeVector(float x, float y, float z){ v[0]=x; v[1]=y; v[2]=z;};
        ThreeVector(float x[3]){v = x};

        float get_x(){return v[0];};
        float get_y(){return v[1];};
        float get_z(){return v[2];};

        void set_x(float x){v[0] = x};
        void set_y(float y){v[1] = y};
        void set_z(float z){v[2] = z};

        float norm2(){return v[0]*v[0]+v[1]*v[1]+v[2]*v[2]};
        float operator * (const ThreeVector & y){return v[0]*y.v[0] + v[1]*y.v[1] +v[2]*y.v[2]};
};

class FourVector: public ThreeVector{
    // a four Vector in special-relativity: E^2= mc^2
    float E;

    public:
        FourVector{};
        FourVector(float e, ThreeVector p): E(e), ThreeVector(p){};
        FourVector(float e, float x, float y, float z): E(e), ThreeVector(x,y,z){
            float norm2(){return E*E-ThreeVector::norm2();}
            float operator * (const FourVector & y) {return E*y.E - ThreeVector(v)*ThreeVector(y);}
};

class ParticleInfo{
    protected:
        float mass;
    public:
        void set_mass(float x){ mass=x; }
        float get_mass(){ return mass; }
};

class Particle: public ParticleInfo, public FourVector{
    public:
        Particle{};
        Particle(FourVector p): FourVector(p){mass=0;};
        float mass_gap(){return norm2()-mass*mass;}
};

int main(){
    FourVector a (100.,1.,1.,1.);
    FourVector b (100., 0.,0.,0.);
    cout << a*b << endl;
    Particle A(a);
    A.set_mass(75);
    cout<< "A "<< A.mass_gap() << endl;
    Particle B(b);
    B.set_mass(100);
    cout<< "B "<< B.mass_gap() << endl;
    return 0;
};
```cpp
// example: class constructor
#include <iostream>
using namespace std;

class Ancestor{
public:
    int year;
    void tell_something(){cout<<"In the year "+year<<endl;};
};

class Mother: public Ancestor{
public:
    void hello(){
        tell_something();
        cout<< "hello from Mother"<< endl;
    }
};

class Father:public Ancestor{
protected:
    int age;
public:
    Father(){};
    Father(int x): age(x){};
};

class Child: public Mother, public Father{
};

int main () {
    Child test;
    test.Mother::year = 1980;
    test.Father::year = 1950;
    test.hello();
    test.Father::tell_something();
}
```
Two copy of the Ancestor class

- test.Mother::year
- test.Father::year

You can use virtual inheritance to have a single copy

Consider as bad design in C++
Template

Template = define functions class with generic type

• Repeat yourself is bad but often you have to have the exact same definition but for different type
  ➞ Template is the solution

```cpp
// overloaded functions
#include <iostream>
using namespace std;

int sum (int a, int b)
{ return a+b;
}

double sum (double a, double b)
{ return a+b;
}

int main ()
{ cout << sum (10,20) << 'n';
 cout << sum (1.0,1.5) << 'n';
 return 0;
}
```
```cpp
// function template
#include <iostream>
using namespace std;

template <class T>
T sum (T a, T b)
{
 T result;
 result = a + b;
 return result;
}

int main ()
{ int i=5, j=6, k;
 double f=2.0, g=0.5, h;
 k=sum<int>({i,j});
 h=sum<double>({f,g});
 cout << k << 'n';
 cout << h << 'n';
 return 0;
}
```
Polymorphism

A pointer to a derived class is type-compatible with a pointer to its base class.

```cpp
// pointers to base class
#include <iostream>
using namespace std;

class Polygon {
    protected:
        int width, height;
    public:
        void set_values (int a, int b)  
                        { width=a; height=b; } 
};
class Rectangle: public Polygon {
    public:
        int area()  
                { return width*height; }
};
class Triangle: public Polygon {
    public:
        int area()  
                { return width*height/2; }
};

int main () {
    Rectangle rect;
    Triangle trgl;
    Polygon * ppoly1 = &rect;
    Polygon * ppoly2 = &trgl;
    ppoly1->set_values (4,5);
    ppoly2->set_values (4,5);
    cout << rect.area() << '\n';
    cout << trgl.area() << '\n';
    return 0;
}
```

- We can use a pointer of the class CPolygon (CPolygon*) with object from his derived class.
- Note that from pointer you can access attribute/member function with ->
- Carefully which function you access with polymorphism.
Exercise IV

• Update your four-vector class to include
  ➡ Scalar Multiplication via Template Method

• Test polymorphism on your class
Conclusion

• Oriented Object
  ➡ Are a nice way to separate the inner work from the way the object are called
  ➡ Inheritance allows you to build/expand without the need to restart from scratch
  ➡ Private argument help you to sand box yourself

• You need to play with it