

# **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.
- 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 affecting any code that uses the object

# Why C++

## Tiobe Ranking

Sep 2019	Sep 2018	Change	Programming Language	Ratings	Change
1	1		Java	16.661%	-0.78%
2	2		C	15.205%	-0.24%
3	3		Python	9.874%	+2.22%
4	4		C++	5.635%	-1.76%
5	6	↑	C#	3.399%	+0.10%
6	5	↓	Visual Basic .NET	3.291%	-2.02%
7	8	↑	JavaScript	2.128%	-0.00%
8	9	↑	SQL	1.944%	-0.12%
9	7	↓	PHP	1.863%	-0.91%
10	10		Objective-C	1.840%	+0.33%

- 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
- Introduction to Class/object in C++
  - ➔ Presentation of concept
  - ➔ Code presentation
  - ➔ Exercise
- (Multi-processor) Parallel programming
  - ➔ Presentation of concept
  - ➔ Code presentation
  - ➔ Exercise

Slides and examples/solutions are on indico

# Hello World

```
1 // my first program in C++
2 #include <iostream>
3
4 int main()
5 {
6     std::cout << "Hello World!";
7 }
```

[cpp.sh/2dd](http://www.cpp.sh/2dd)

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

- 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++

### Cluster/linux

#### Run Once

```
module load GCC
```

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

### Mac

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

Note some C++11 syntax supported

### Problem

<https://ideone.com/>

Select C++ (bottom left)

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

[https://www.tutorialspoint.com/compile\\_cpp\\_online.php](https://www.tutorialspoint.com/compile_cpp_online.php)

## C++11

### Cluster/linux

#### Run Once

```
module load GCC
```

```
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++14 (bottom left)

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

[https://www.tutorialspoint.com/compile\\_cpp\\_online.php](https://www.tutorialspoint.com/compile_cpp_online.php)

# Basic of C++ : variables



Variable = portion of memory storing a value

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

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

```
1 // initialization of variables
2
3 #include <iostream>
4 using namespace std;
5
6 int main ()
7 {
8     int a=5;           // initial value: 5
9     int b(3);         // initial value: 3
10    int c{2};         // initial value: 2
11    int result;       // initial value undetermined
12
13    a = a + b;
14    result = a - c;
15    cout << result;
16
17    return 0;
18 }
```

<http://cpp.sh/8yl>

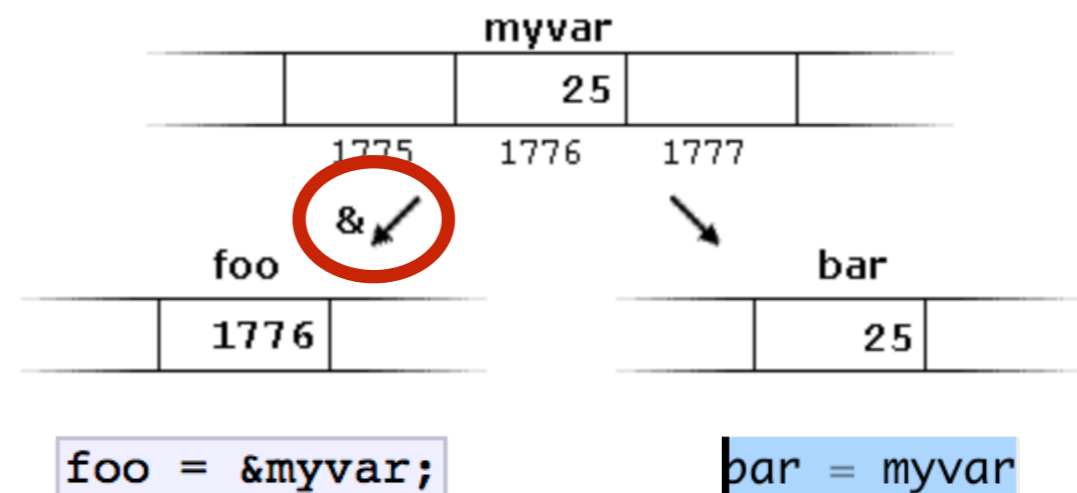
```
1 // my first string
2 #include <iostream>
3 #include <string>
4 using namespace std;
5
6 int main ()
7 {
8     string mystring;
9     mystring = "This is a string";
10    cout << mystring;
11    return 0;
12 }
```

<http://cpp.sh/7d4>

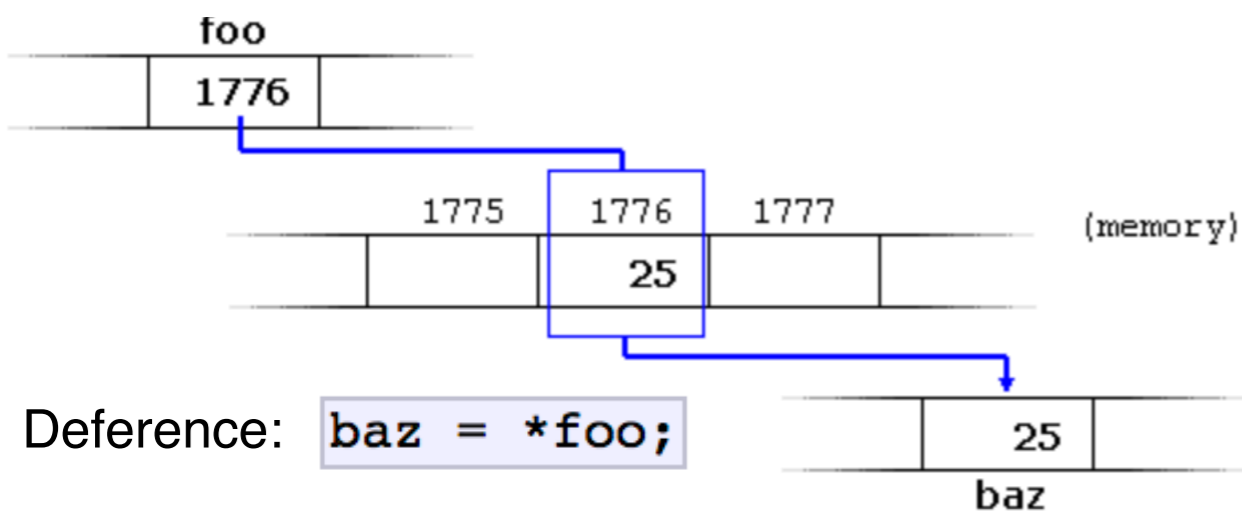
# Basic of C++: pointer



Pointer = variable containing the address of another variable



```
foo = &myvar;
```



- Due to dereference pointer also have typed:
  - ➔ Those are the type of the variable suffix by a star

```
1 int * number;  
2 char * character;  
3 double * decimals;
```





# Basic of C++: functions

Function = group of statements

- that has a given name,
- which can be called from some point of the program

## Passing Parameters by Value

[cpp.sh/2lp](http://cpp.sh/2lp)

```
1 // function example
2 #include <iostream>
3 using namespace std;
4
5 int addition (int a, int b)
6 {
7     int r;
8     r=a+b;
9     return r;
10 }
11
12 int main ()
13 {
14     int z;
15     z = addition (5,3);
16     cout << "The result is " << z;
17 }
```

Input Variable **CAN not** be changed by the function

## Passing Parameters by reference

<http://cpp.sh/9b2>

```
1 // passing parameters by reference
2 #include <iostream>
3 using namespace std;
4
5 void duplicate (int& a, int& b, int& c)
6 {
7     a*=2;
8     b*=2;
9     c*=2;
10 }
11
12 int main ()
13 {
14     int x=1, y=3, z=7;
15     duplicate (x, y, z);
16     cout << "x=" << x << ", y=" << y << ", z=" << z;
17     return 0;
18 }
```

Input Variable **CAN** be changed by the function

# Data structure

- Can we have a special data-type with metadata
  - ➔ Like a “formation”
    - ◆ With the number of student
    - ◆ The name of the formation
    - ◆ The name of the teacher

```
struct Formation {  
    char title[50];  
    char speaker[50];  
    int nb_student;  
};
```

```
int main( ) {  
  
    struct Formation Lect_C;  
    struct Formation Lect_Cpp;  
  
    /* Formation C initialization*/  
    strcpy( Lect_C.title, "C Programming");  
    strcpy( Lect_C.speaker, "O. Mattelaer");  
    Lect_C.nb_student = 10;  
  
    /* print Book1 info */  
    printf( " Formation \"%s\" given by \"%s\" has %d student",  
           Lect_C.title, Lect_C.speaker, Lect_C.nb_student);  
}
```

# More on Data structure

- Access data:

- ➔ From variable use the “.”
- ➔ From pointer use the “->”

```
struct Formation myformation;  
formation.title;|  
(&formation)->title;
```



# Classes

classes = data structure with functions

data structure = group of data elements grouped together under a single name

<http://cpp.sh/34lna>

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

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

int main()
{
    Rectangle myrect;
    myrect.width = 5;
    myrect.height = 10;
    cout<<"area is "<< myrect.area()<< endl;
}
```

- “myrect” is an object
  - ➔ Also called instance
- Call to function “similar” to accessing attribute (“.” Or “->”)
- Simpler syntax than structure for the creation of the object

# Classes

classes = data structure with functions

data structure = group of data elements grouped together under a single name

```
// my first program in C
#include <iostream>
#include <stdio.h>
using namespace std;

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

void Rectangle::info(){
    printf("Rectangle(%d,%d)\n", width, height);
    printf("    associated area is %d", area());
}

int main()
{
    Rectangle myrect;
    myrect.width = 5;
    myrect.height = 10;
    //cout<<"area is "<< myrect.area()<< endl;
    myrect.info();
}
```

- As for normal function, you do not have to define the full function in the class definition you can postpone it.
- Note that we do not define width/height inside the function

<http://tpcg.io/bKCfmxxQ>

# C++ classes have private attribute/fct

- Public attribute are readable and writable
  - ➔ Can be annoying in large code

```
// my first program in C
#include <iostream>
#include <stdio.h>
using namespace std;

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

void Rectangle::info(){
    printf("Rectangle(%d,%d)\n", width, height);
    printf("    associated area is %d", area());
}

int main()
{
    Rectangle myrect;
    myrect.width = 5;
    myrect.height = 10;
    //cout<<"area is "<< myrect.area()<< endl;
    myrect.info();
}
```

- Allows distinction between
  - Visible information
  - Internal mechanism



# Visibility of attribute/function

private

Only accessible from other instance of the same class

Accessible from friends

DEFAULT

```
#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;
};
```

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

protected

Accessible from other instance of the same class

Accessible from friends

Accessible from instance of the **derived**/child class

public

Accessible from everywhere where the object is visible

READ and WRITE!

```
#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;
};
```

# Private argument

```
2 #include <iostream>
3 #include <stdio.h>
4 using namespace std;
5
6 class Rectangle{
7     int current_area=0;
8     int width, height;
9     public:
10    void set_width(int w) {width=w; current_area=0;}
11    void set_height(int h) {height=h; current_area=0;}
12    int get_width(){ return width;}
13    int get_height(){return height;}
14    int area();
15    void info();
16 };
17
18 int Rectangle::area(){
19     if (current_area!=0){
20         return current_area;
21     }
22     cout<<"computing area ... please wait"<< endl;
23     current_area = width*height;
24     return width*height;
25 }
26
27 void Rectangle::info(){
28
29     printf("Rectangle(%d,%d)\n", width, height);
30     printf("    associated area is %d \n", area());
31 }
32
```

- Use get/set public attribute to allow to read/write attribute
- Allow to “cache” some result
- Function can also be private

<http://tpcg.io/bKCfmxxQ>

# Constructor

constructor = function called after the object is created

[cpp.sh/8lr](http://cpp.sh/8lr)

```
1 // example: class constructor
2 #include <iostream>
3 using namespace std;
4
5 class Rectangle {
6     int width, height;
7     public:
8     Rectangle (int,int);
9     int area () {return (width*height);}
10 };
11
12 Rectangle::Rectangle (int a, int b) {
13     width = a;
14     height = b;
15 }
16
17 int main () {
18     Rectangle rect (3,4);
19     Rectangle rectb (5,6);
20     cout << "rect area: " << rect.area() << endl;
21     cout << "rectb area: " << rectb.area() << endl;
22     return 0;
23 }
```

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

- Shortcut for setting attribute

```
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**.

```
1 // example: class constructor
2 #include <iostream>
3 using namespace std;
4
5 class Rectangle {
6     int width, height;
7     public:
8     Rectangle (int,int);
9     Rectangle (int l): width(l), height(l){};
10    int area () {return (width*height);}
11 };
12
13 Rectangle::Rectangle (int a, int b) {
14     width = a;
15     height = b;
16 }
17
18 int main () {
19     Rectangle rect (3);
20     Rectangle rectb (5,6);
21     cout << "rect area: " << rect.area() << endl;
22     cout << "rectb area: " << rectb.area() << endl;
23     return 0;
24 }
```

- 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

Overloadable operators												
+	-	*	/	=	<	>	+=	-=	*=	/=	<<	>>
<<=	>>=	==	!=	<=	>=	++	--	%	&	^	!	
~	&=	^=	=	&&		%=	[]	()	,	->*	->	new
delete		new[]		delete[]								

[cpp.sh/271](http://cpp.sh/271)

```
1 // overloading operators example
2 #include <iostream>
3 using namespace std;
4
5 class CVector {
6     public:
7         int x,y;
8         CVector () {};
9         CVector (int a,int b) : x(a), y(b) {}
10        CVector operator + (const CVector&);
11 };
12
13 CVector CVector::operator+ (const CVector& param) {
14     CVector temp;
15     temp.x = x + param.x;
16     temp.y = y + param.y;
17     return temp;
18 }
19
20 int main () {
21     CVector foo (3,1);
22     CVector bar (1,2);
23     CVector result;
24     result = foo + bar;
25     cout << result.x << ',' << result.y << '\n';
26     return 0;
27 }
```

# Special members

Special members = member functions implicitly defined

Member function	typical form for class C:
Default constructor	<code>C::C();</code>
Destructor	<code>C::~~C();</code>
Copy constructor	<code>C::C (const C&amp;);</code>
Copy assignment	<code>C&amp; operator= (const C&amp;);</code>
Move constructor	<code>C::C (C&amp;&amp;);</code>
Move assignment	<code>C&amp; operator= (C&amp;&amp;);</code>

- Default constructor:
  - ➔ Present only if no other constructor exists!
- Destructor `~CLASSNAME`:
  - ➔ Perform cleanup (remove dynamical 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 attribute

```
MyClass::MyClass(const MyClass& x) : a(x.a), b(x.b), c(x.c) {}
```

```
1 MyClass fn();           // function returning a MyClass object
2 MyClass foo;           // default constructor
3 MyClass bar = foo;     // copy constructor
4 MyClass baz = fn();    // move constructor
5 foo = bar;             // copy assignment
6 baz = MyClass();       // move assignment
```

# Example

```

1 // example: class constructor
2 #include <iostream>
3 using namespace std;
4
5 class Rectangle {
6     int width, height;
7     public:
8     Rectangle();
9     Rectangle (int,int);
10    Rectangle (int a, int b, int c): Rectangle(a,b){cout << c<<endl;};
11    Rectangle (int l){width=l; height=l;};
12    Rectangle(const Rectangle& x){width=x.width; height=x.height; cout<<"copy "<<x.width<<" "<<x.height<<endl;};
13    int area () {return (width*height);}
14    Rectangle intersection(Rectangle);
15 };
16
17 Rectangle::Rectangle (int a, int b) {
18     width = a;
19     height = b;
20 }
21
22 Rectangle Rectangle::intersection(Rectangle B){
23     //returns a rectangle with the smallest width and height
24     Rectangle out;
25     if (width < B.width){
26         out.width = width;
27     }else{
28         out.width = B.width;
29     };
30     if (height < B.height){
31         out.height = height;
32     }else{
33         out.height = B.height;
34     };
35     return out;
36 };
37
38
39
40 int main () {
41     Rectangle rect (3);
42     Rectangle rectb (2,6,30);
43     Rectangle small = rect.intersection(rectb);
44     cout << "rect area: " << rect.area() << endl;
45     cout << "small area: " << small.area() << endl;
46     return 0;
47 }

```

# Exercise I

- 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 the vectoriel product of two vector
- Define a Class parallelogram
  - ➔ Can be initialised by two vector
  - ➔ Set a function to compute the associated area (norm of vectoriel product)



# Solution

[cpp.sh/6vgu2c](#)

```
1 // example: ThreeVector
2 #include <iostream>
3 #include <math.h>
4 using namespace std;
5
6 class ThreeVector{
7     float v[3];
8
9 public:
10    ThreeVector(){};
11    ThreeVector(float x, float y, float z){ v[0]=x; v[1]=y; v[2]=z;};
12
13    float get_x(){return v[0];};
14    float get_y(){return v[1];};
15    float get_z(){return v[2];};
16
17    void set_x(float x){v[0] = x;};
18    void set_y(float y){v[1] = y;};
19    void set_z(float z){v[2] = z;};
20
21    float norm(){return sqrt(v[0]*v[0]+v[1]*v[1]+v[2]*v[2]);};
22    float operator * (const ThreeVector& y){return v[0]*y.v[0] + v[1]*y.v[1] +v[2]*y.v[2];}
23 };
24
25 int main () {
26     ThreeVector a(1,2,3);
27     ThreeVector b(1,0,0);
28     cout << "norm a" << a.norm() << endl;
29     cout << "norm b" << b.norm() << endl;
30     cout << "a*b=" << a*b << endl;
31 }
```

# Solution

```
class ThreeVector{  
    float v[3];
```

```
    ThreeVector vmult(ThreeVector);
```

```
    ThreeVector ThreeVector::vmult(ThreeVector second){  
        ThreeVector out;  
        out.v[0] = v[1]*second.v[2] - v[2]*second.v[1];  
        out.v[1] = v[2]*second.v[0] - v[0]*second.v[2];  
        out.v[2] = v[0]*second.v[1] - v[1]*second.v[0];  
        return out;  
    };
```

<http://cpp.sh/3pj6pp>

```
class Parralelogram{  
    ThreeVector first;  
    ThreeVector second;  
public:  
    Parralelogram(ThreeVector f, ThreeVector second): first(f), second(second){};  
    float get_area() {return first.vmult(second).norm();}  
};
```

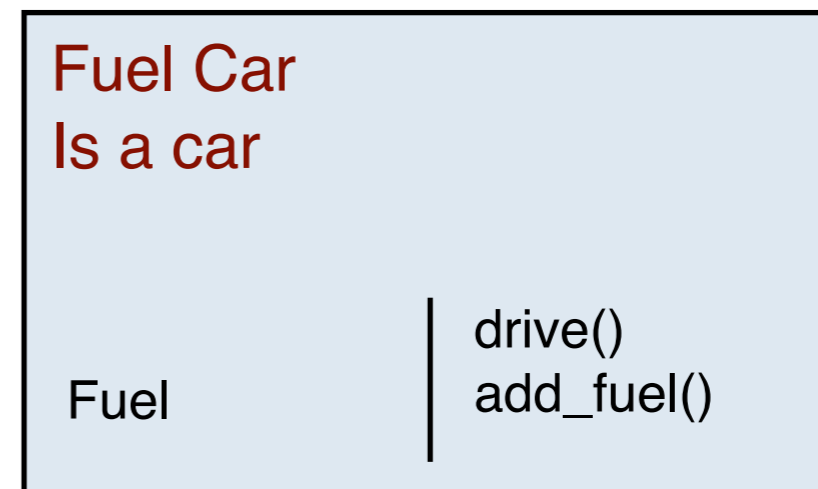
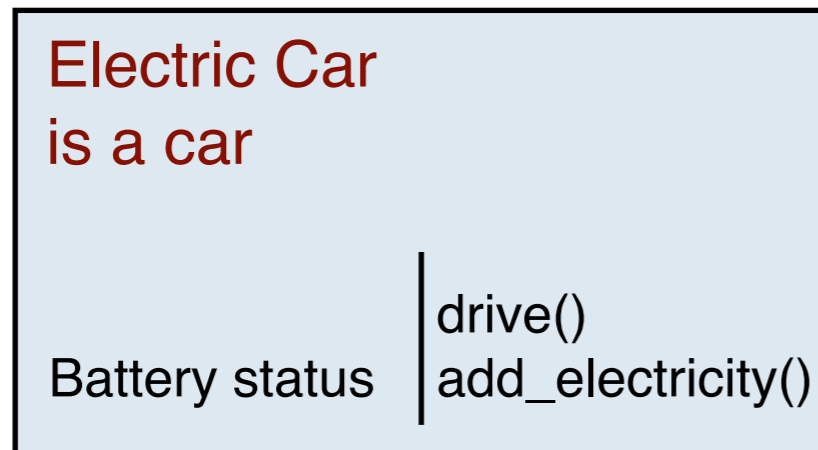
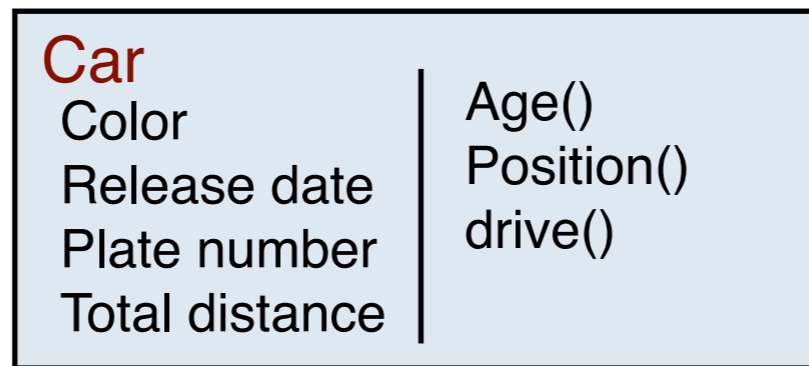
```
int main () {  
    ThreeVector a(1,2,3);  
    ThreeVector b(1,0,0);  
    cout << "norm a " << a.norm() << endl;  
    cout << "norm b " << b.norm() << endl;  
    cout << "a*b= " << a*b << endl;  
    Parralelogram P(a,b);  
    cout << "area of parralelogram " << P.get_area()<<endl;  
}
```

# Inheritance

Electric Car	
Color	Age()
Release date	Position()
Plate number	drive()
Battery status	add_electricity()

Fuel Car	
Color	Age()
Release date	Position()
Plate number	drive()
Fuel	add_fuel()

# Inheritance



- The two class (Electric/fuel car) does not to redefine their structure just what they changed compare to the original class!
- They can change or supersede the behaviour

# Visibility of attribute/function

private

Only accessible from other instance of the same class

Accessible from friends

DEFAULT

```
#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;
};
```

protected

Accessible from other instance of the same class

Accessible from friends

Accessible from instance of the **derived**/child class

public

Accessible from everywhere where the object is visible

READ and WRITE!

```
#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;
};
```



# 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.

<http://cpp.sh/9m2>

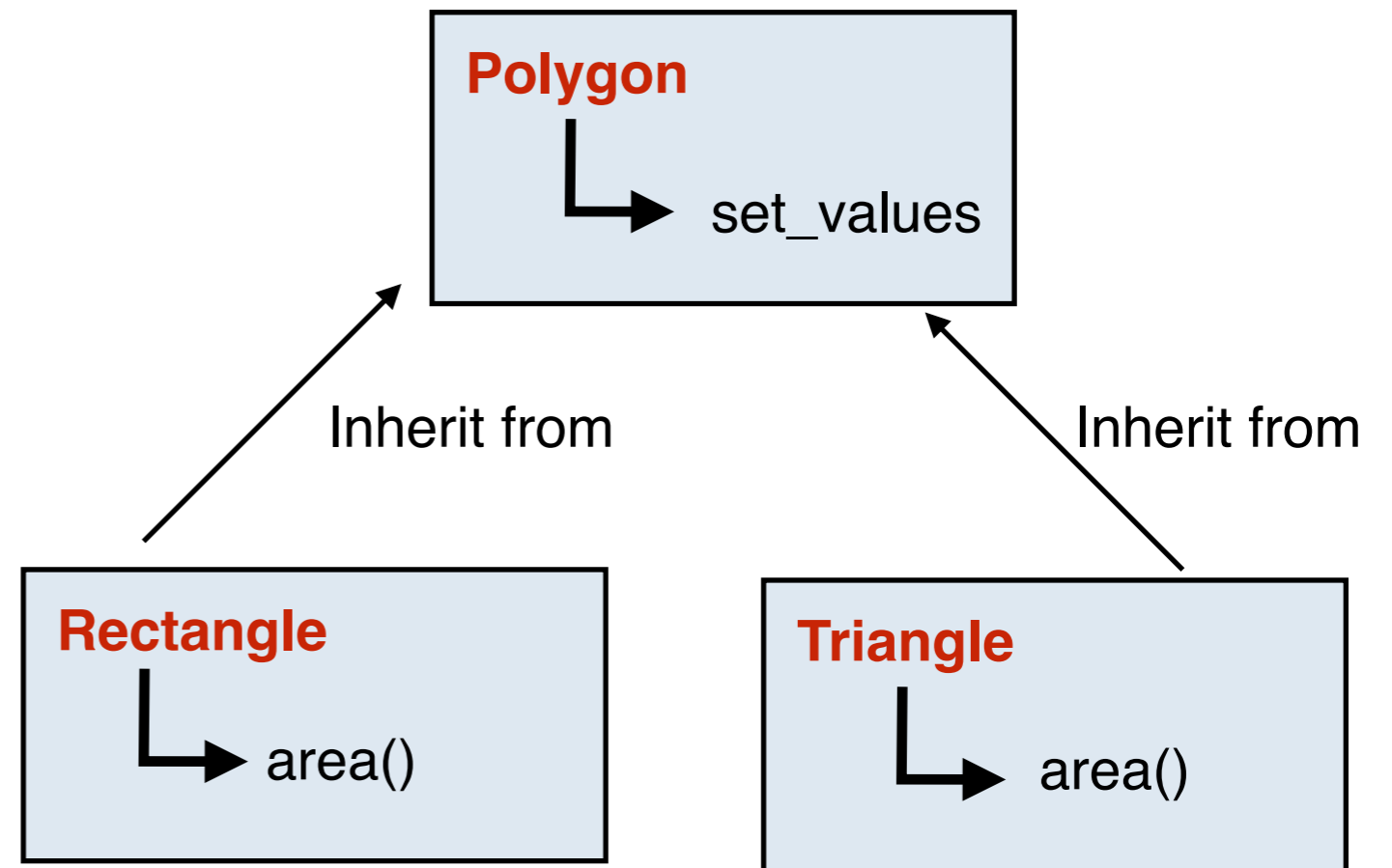
```
// derived classes
#include <iostream>
using namespace std;

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;
    rect.set_values (4,5);
    trgl.set_values (4,5);
    cout << rect.area() << '\n';
    cout << trgl.area() << '\n';
    return 0;
}
```



Both Rectangle and Triangle can call `set_values`

# 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.

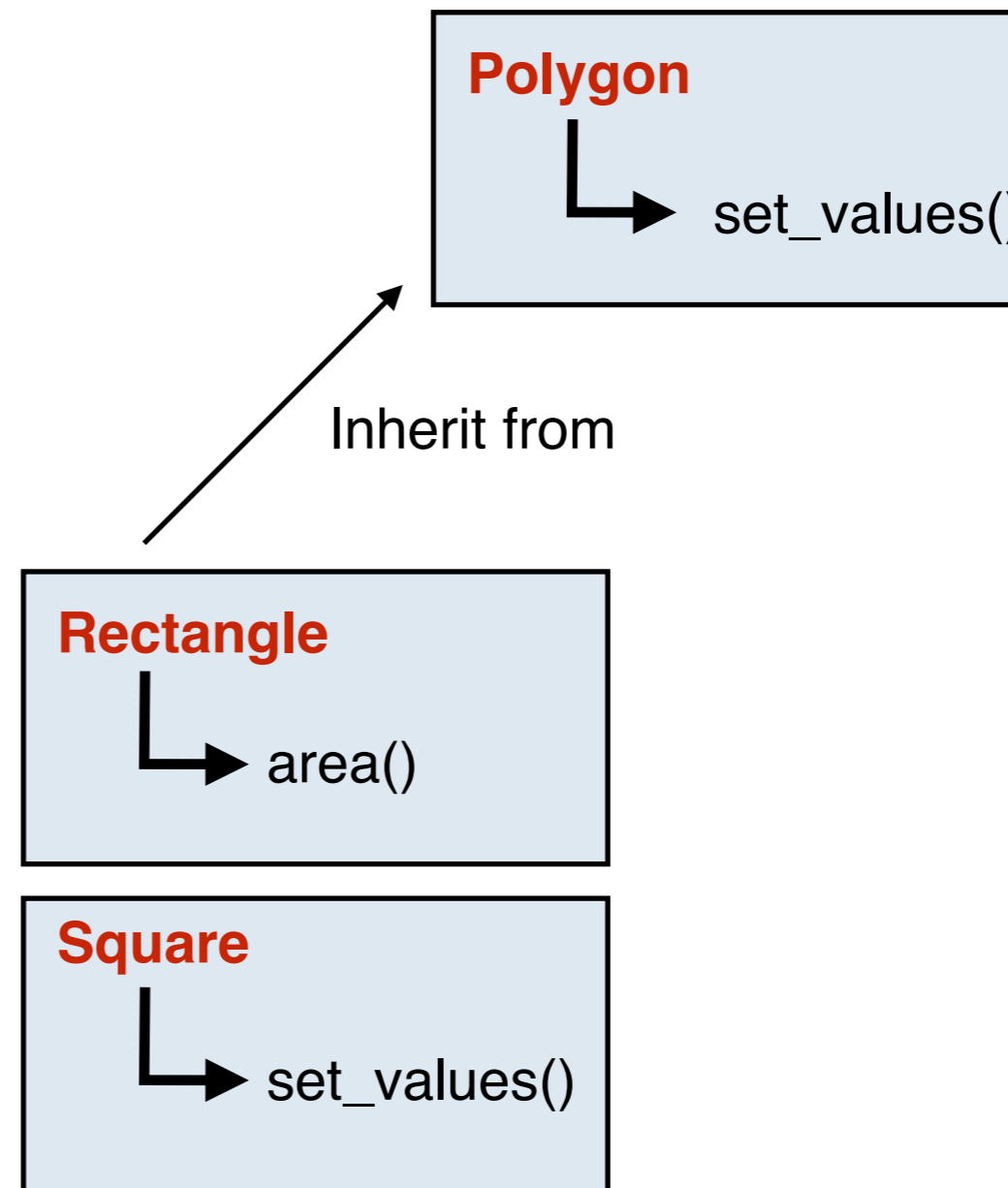
<http://tpcg.io/bKCfmxxQ>

```
// derived classes
#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 Square: public Rectangle {
public:
    void set_values (int a, int b){
        if (a!=b){
            throw "Square need same lenght on both argument";
        }else{
            Polygon::set_values(a,b);
        }
    };
};
```



# 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.sh/72itc](http://cpp.sh/72itc)

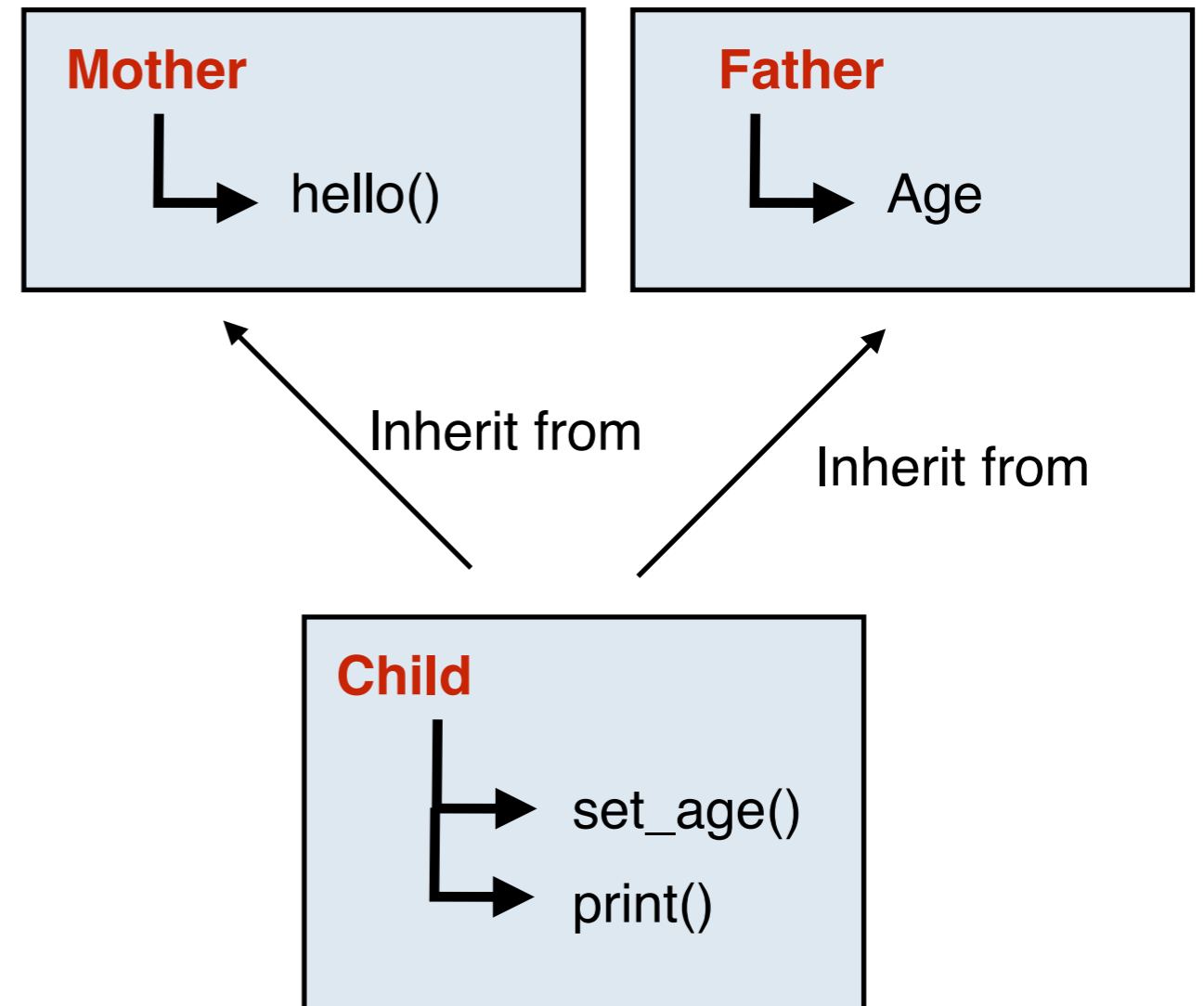
```
1 // example: class constructor
2 #include <iostream>
3 using namespace std;
4
5 class Mother{
6 public:
7     void hello(){
8         cout<< "hello from Mother"<<endl;};
9 };
10
11 class Child1: public Mother{};
12
13 class Child2: public Mother{
14
15 public:
16     void hello() {
17         Mother::hello();
18         cout<< "and from Child2" << endl;};
19 };
20
21 int main () {
22     Child1 test;
23     test.hello();
24
25     Child2 test2;
26     test2.hello();
27 }
```

- “public” tells the maximum level of visibility of the attribute coming from the base class
  - Rare case when not set on public
- Private argument are not passed to the child (but they still exists!)
- Constructor/Destructor are **not** passed to the child
- Assignment operator (operator =) are **not** passed to the child

# Multi-inheritance

[cpp.sh/3nhb](http://cpp.sh/3nhb)

```
1 // example: class constructor
2 #include <iostream>
3 using namespace std;
4
5 class Mother{
6 public:
7     void hello(){
8         cout<< "hello from Mother"<< endl;};
9 };
10
11 class Father{
12 protected:
13     int age;
14 public:
15     Father(){};
16     Father(int x): age(x){};
17 };
18
19
20 class Child: public Mother, public Father{
21
22 public:
23     Child(int x){age=x;};
24
25     void print() {hello(); cout<<"my age is " << age;}
26     void set_age(int x){age=x;};
27
28 };
29
30
31 int main () {
32     Child test(3);
33     test.hello();
34     test.print();
35     test.set_age(4);
36     test.print();
37 }
```



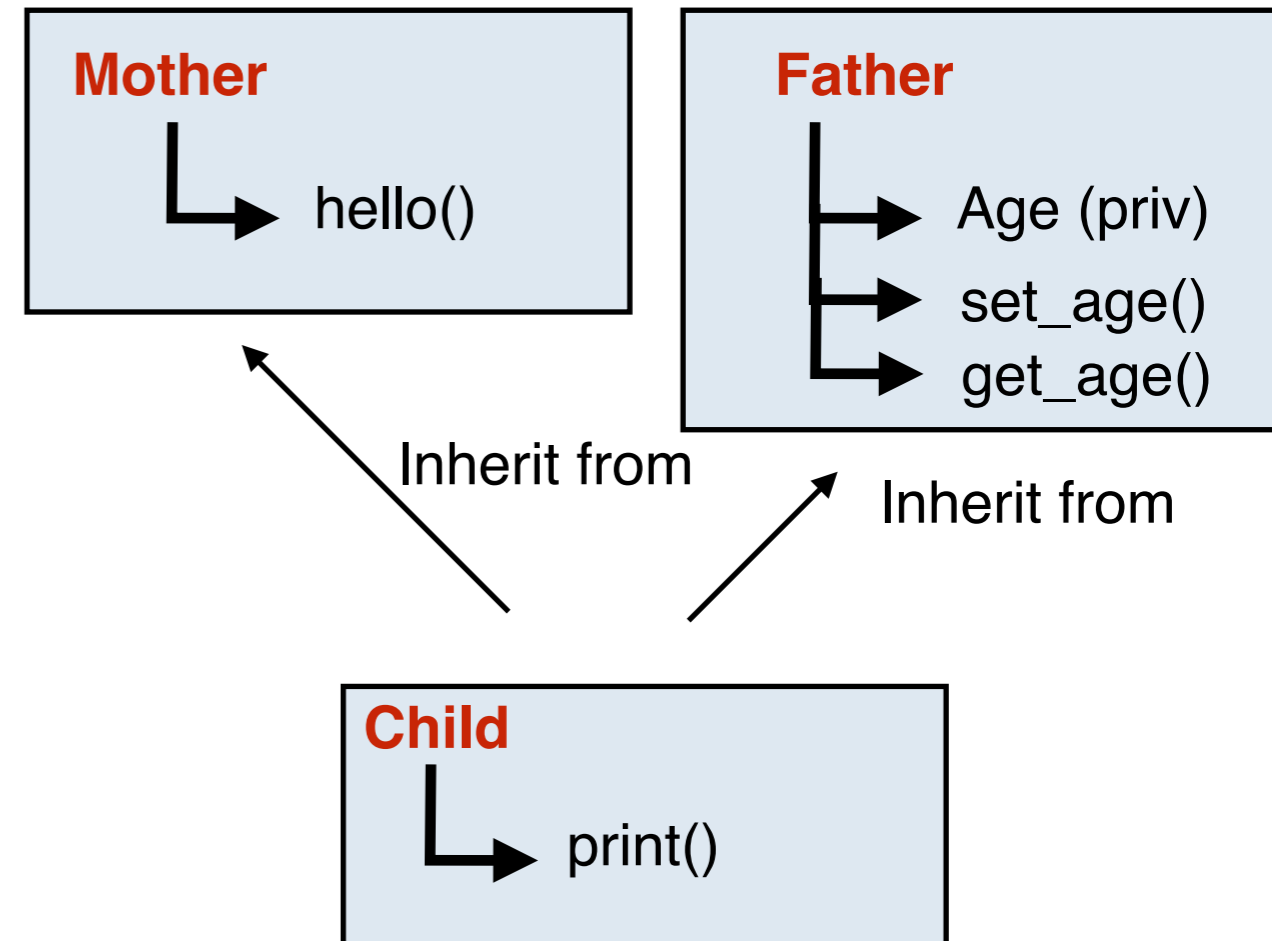
Can still call hello()

Can access to age (protected)

# Multi-inheritance

[cpp.sh/8vev](http://cpp.sh/8vev)

```
1 // example: class constructor
2 #include <iostream>
3 using namespace std;
4
5 class Mother{
6 public:
7     void hello(){
8         cout<< "hello from Mother"<< endl;};
9 };
10
11 class Father{
12     int age;
13 public:
14     Father();
15     Father(int x): age(x){};
16     void set_age(int x){age=x;};
17     int get_age(){return age;};
18 };
19
20
21 class Child: public Mother, public Father{
22
23 public:
24     Child(int x){set_age(x);};
25     void print() {hello(); cout<<"my age is " << get_age();}
26
27
28 };
29
30
31 int main () {
32     Child test(3);
33     test.hello();
34     test.print();
35     test.set_age(4);
36     test.print();
37 }
```



Can call hello()

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



# Exercise II

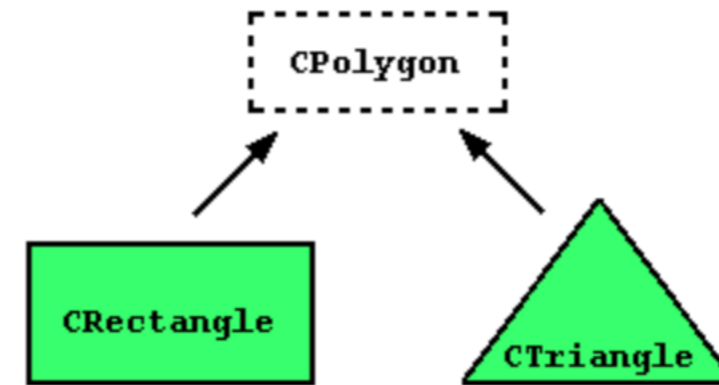
- Update your Rectangle class to have a function returning the smallest Rectangle
- Define a class VectorRectangle
  - Which inherits from your parralelogram class
  - Which inherits from your rectangle class

# Polymorphism

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

[cpp.sh/3tz](#)

```
1 // pointers to base class
2 #include <iostream>
3 using namespace std;
4
5 class Polygon {
6     protected:
7         int width, height;
8     public:
9         void set_values (int a, int b)
10            { width=a; height=b; }
11 };
12
13 class Rectangle: public Polygon {
14     public:
15         int area()
16            { return width*height; }
17 };
18
19 class Triangle: public Polygon {
20     public:
21         int area()
22            { return width*height/2; }
23 };
24
25 int main () {
26     Rectangle rect;
27     Triangle trgl;
28     Polygon * ppoly1 = &rect;
29     Polygon * ppoly2 = &trgl;
30     ppoly1->set_values (4,5);
31     ppoly2->set_values (4,5);
32     cout << rect.area() << '\n';
33     cout << trgl.area() << '\n';
34     return 0;
35 }
```

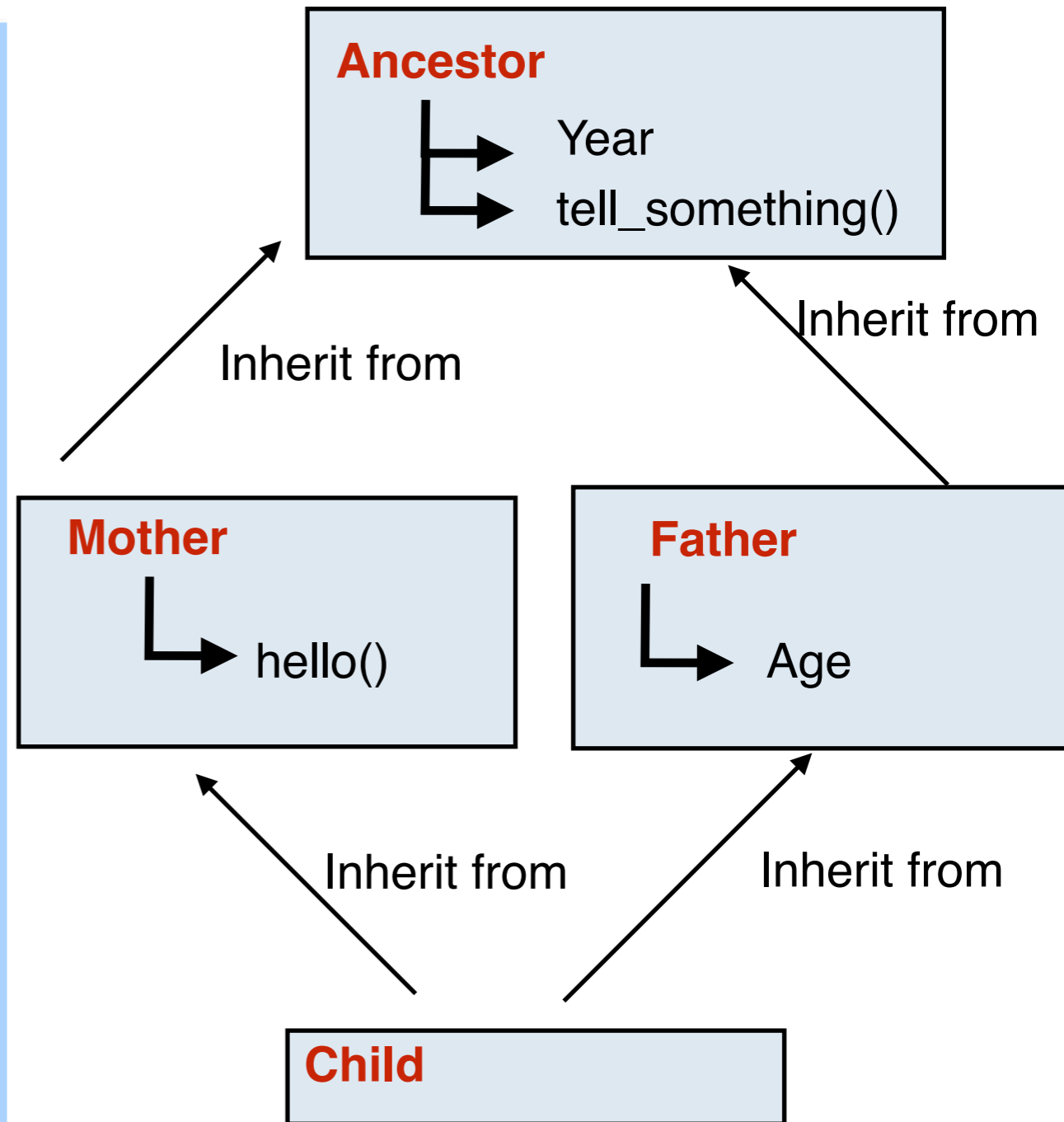


- We can use a pointer of the class CPolygon (CPolygon\*) with object from his derived class
- Note that from pointer you can access attribute/member function with ->
- Carefull which function you access with polymorphism

# Diamond Diagram

[cpp.sh/4inoj](http://cpp.sh/4inoj)

```
1 // example: class constructor
2 #include <iostream>
3 using namespace std;
4
5 class Ancestor{
6 public:
7     int year;
8     void tell_something(){cout<<"In the year " << year <<endl;};
9 };
10
11 class Mother: public Ancestor{
12 public:
13     void hello(){
14         tell_something();
15         cout<< "hello from Mother"<< endl;
16     };
17 };
18
19 class Father:public Ancestor{
20 protected:
21     int age;
22 public:
23     Father(){};
24     Father(int x): age(x){};
25 };
26
27 class Child: public Mother, public Father{
28 };
29
30
31 int main () {
32     Child test;
33     test.Mother::year = 1980;
34     test.Father::year = 1950;
35     test.hello();
36     test.Father::tell_something();
37 }
```



# Diamond Diagram

[cpp.sh/4inoj](http://cpp.sh/4inoj)

```
1 // example: class constructor
2 #include <iostream>
3 using namespace std;
4
5 class Ancestor{
6 public:
7     int year;
8     void tell_something(){cout<<"In the year " << year <<endl;};
9 };
10
11 class Mother: public Ancestor{
12 public:
13     void hello(){
14         tell_something();
15         cout<< "hello from Mother"<< endl;
16     };
17 };
18
19 class Father:public Ancestor{
20 protected:
21     int age;
22 public:
23     Father(){};
24     Father(int x): age(x){};
25 };
26
27 class Child: public Mother, public Father{
28 };
29
30
31 int main () {
32     Child test;
33     test.Mother::year = 1980;
34     test.Father::year = 1950;
35     test.hello();
36     test.Father::tell_something();
37 }
```

- Two copy of the Ancestor class
  - ➔ test.Mother::year
  - ➔ test.Father::year
- You can use virtual inheritance to have a single copy
  - ➔ “public virtual Ancestor”
- Consider as bad design in C++
  - ➔ Because C++ sucks on those!

# 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

```
1 // overloaded functions
2 #include <iostream>
3 using namespace std;
4
5 int sum (int a, int b)
6 {
7     return a+b;
8 }
9
10 double sum (double a, double b)
11 {
12     return a+b;
13 }
14
15 int main ()
16 {
17     cout << sum (10,20) << '\n';
18     cout << sum (1.0,1.5) << '\n';
19     return 0;
20 }
```



[cpp.sh/4jq](#)

```
1 // function template
2 #include <iostream>
3 using namespace std;
4
5 template <class T>
6 T sum (T a, T b)
7 {
8     T result;
9     result = a + b;
10    return result;
11 }
12
13 int main () {
14     int i=5, j=6, k;
15     double f=2.0, g=0.5, h;
16     k=sum<int>(i,j);
17     h=sum<double>(f,g);
18     cout << k << '\n';
19     cout << h << '\n';
20     return 0;
21 }
```



# 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
  - ➔ Coding is learning by exercise/exploration
  - ➔ Read book on coding style
    - ◆ How to present you code (space/comment/indentation)
    - ◆ Type of good structure/...