

# **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

Functional, Generic, structured,  
procedural, object oriented

# Why C++

## Tiobe Ranking

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

- 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

```
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++

Hmem/linux

```
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>

C++11

Hmem/linux

Run Once

```
module load GCC/4.9.3-2.25
```

```
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>

# 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
C++11 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/8y1>

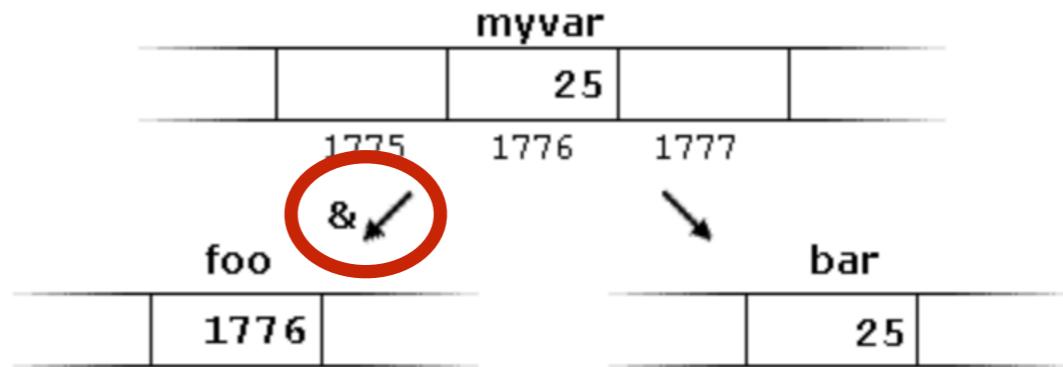
```
1 // my first string
2 #include <iostream>
3 #include <string>
4 using namespace std;
5
6 int main ()
7 {
8     string mystring;
9     mystring = "This is a string";
10    cout << mystring;
11    return 0;
12 }
```

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

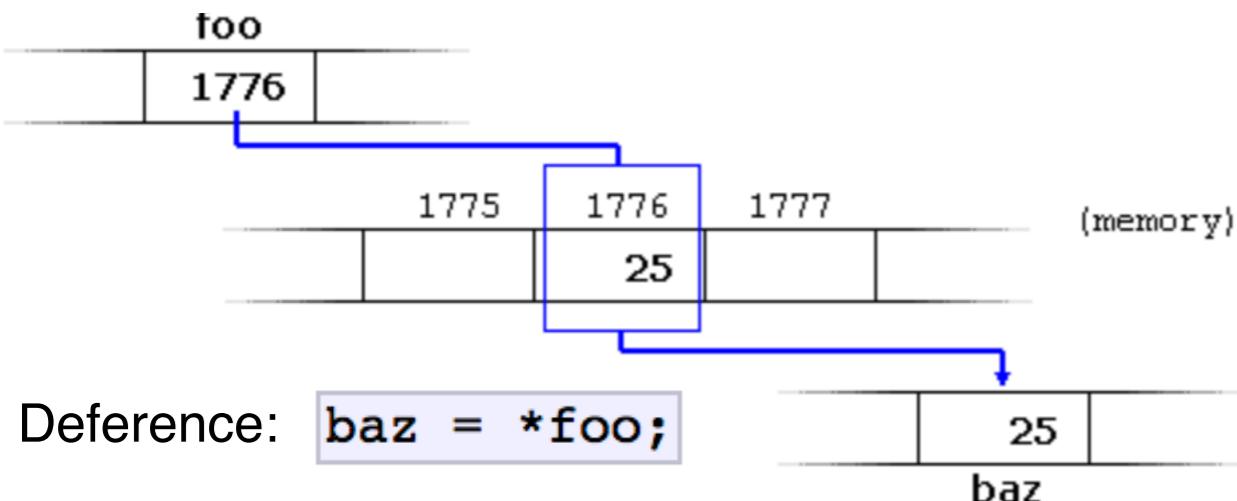
# Basic of C++: pointer



Pointer = position in memory of the variable



```
foo = &myvar;
```



- Due to deference 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 is given a name,
- which can be called from some point of the program

## Passing Parameters by Variable

[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 }
```

## Passing Parameters by reference

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

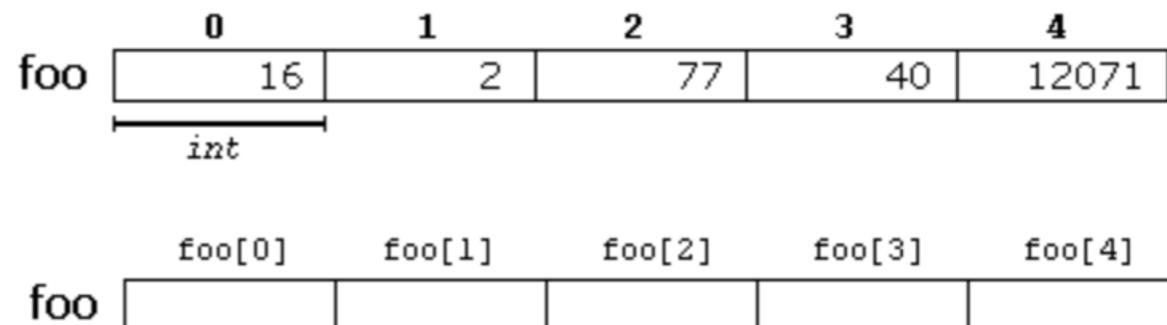
```
1 // passing parameters by reference
2 #include <iostream>
3 using namespace std;
4
5 void duplicate (int& a, int& b, int& c)
6 {
7     a*=2;
8     b*=2;
9     c*=2;
10}
11
12 int main ()
13 {
14     int x=1, y=3, z=7;
15     duplicate (x, y, z);
16     cout << "x=" << x << ", y=" << y << ", z="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

# Basic of C++: Array



Array = sequential memory space of the same type

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



```
1 int foo[ ] = { 10, 20, 30 };
```

C++11 2 int foo[ ] { 10, 20, 30 };

[cpp.sh/6fzb](#)

```
1 // arrays as parameters
2 #include <iostream>
3 using namespace std;
4
5 void printarray (int arg[], int length) {
6     for (int n=0; n<length; ++n)
7         cout << arg[n] << ' ';
8     cout << '\n';
9 }
10
11 int main ()
12 {
13     int firstarray[] = {5, 10, 15};
14     int secondarray[] = {2, 4, 6, 8, 10};
15     printarray (firstarray,3);
16     printarray (secondarray,5);
17 }
```

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

[cpp.sh/7aot](#)

# Exercise I

- 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

[part I : cpp.sh/6ar2x](#)

[part II: cpp.sh/3wr4](#)

```
1 // function example
2 #include <iostream>
3 using namespace std;
4
5 int* cut_before_val ( int sequence[], int val)
6 {
7     int i =0;
8     while(true){
9         if(sequence[i] == val){
10             return &sequence[i];
11         }
12         i++;
13     }
14 };
15
16 int main (){
17     int a[] ={1,2,3,4,5};
18     int* z;
19     z = cut_before_val(a,3);
20     cout << "The result is " << z[1] << "next" << z[2]<<endl;
21     cout << "The result is " << *(++z) << "next" << *(++z)<<endl;
22 }
23
```

# Classes



classes = data structure with functions

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

- 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

```
1 class Rectangle {  
2     int width, height;  
3 public:  
4     void set_values (int,int);  
5     int area (void);  
6 } rect;
```

# Visibility of attribute/function

private	protected	public
<p>Only accessible from other instance of the same class</p> <p>Accessible from friends</p> <p><b>DEFAULT</b></p>	<p>Accessible from other instance of the same class</p> <p>Accessible from friends</p> <p>Accessible from instance of the <b>derived</b>/child class</p>	<p>Accessible from everywhere where the object is visible</p> <p><b>READ and WRITE!</b></p>
<pre>#include &lt;iostream&gt; using namespace std;  class Rectangle{ private:     int width, height; };  int main(){     Rectangle A;     A.width =3;     A.height=2;     cout &lt;&lt; "width=" &lt;&lt; A.width&lt;&lt;endl; };</pre>		<pre>#include &lt;iostream&gt; using namespace std;  class Rectangle{ public:     int width, height; };  int main(){     Rectangle A;     A.width =3;     A.height=2;     cout &lt;&lt; "width=" &lt;&lt; A.width&lt;&lt;endl };</pre>
<pre>simple.cpp:11:5: error: 'width' is a private member of 'Rectangle'     A.width =3;</pre>		

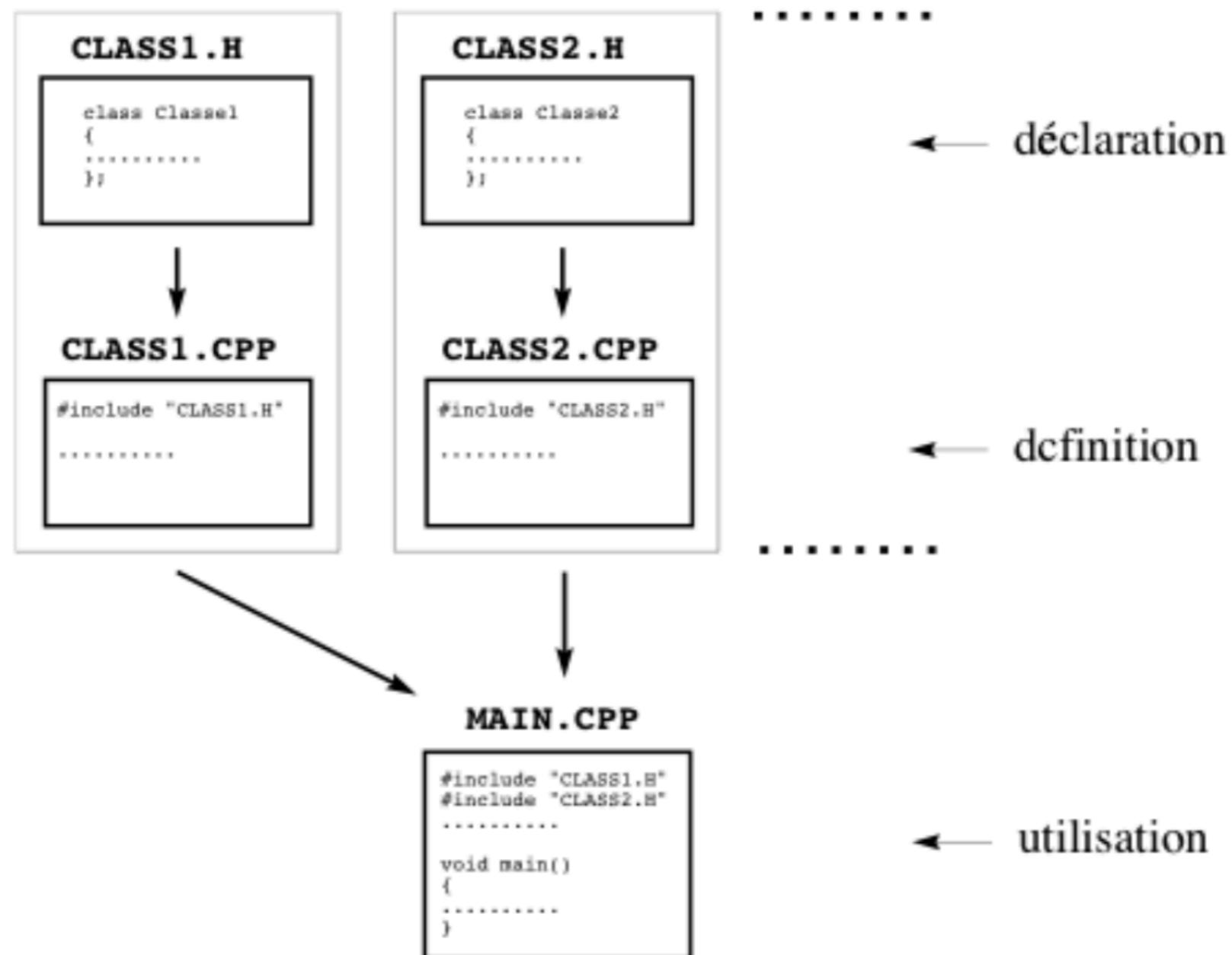
# First Example

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

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

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

# Code Structure



# Constructor

constructor = function called after the object is created

[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](#)

```
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	C:::C();
Destructor	C::~C();
Copy constructor	C:::C (const C&);
Copy assignment	C& operator= (const C&);
Move constructor	C:::C (C&&);
Move assignment	C& operator= (C&&);

- 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
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 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.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

[cpp.sh/7dpvg](#)

```
26 class Parralelogram{  
27     ThreeVector first;  
28     ThreeVector second;  
29 public:  
30     Parralelogram(ThreeVector f, ThreeVector second): first(f), second(second){};  
31     float get_area() {return first*second;}  
32 };  
33  
34  
35 int main () {  
36     ThreeVector a(1,2,3);  
37     ThreeVector b(1,0,0);  
38     cout << "norm a " << a.norm() << endl;  
39     cout << "norm b " << b.norm() << endl;  
40     cout << "a*b= " << a*b << endl;  
41     Parralelogram P(a,b);  
42     cout << "area of parralelogram " << P.get_area()<<endl;  
43 }
```

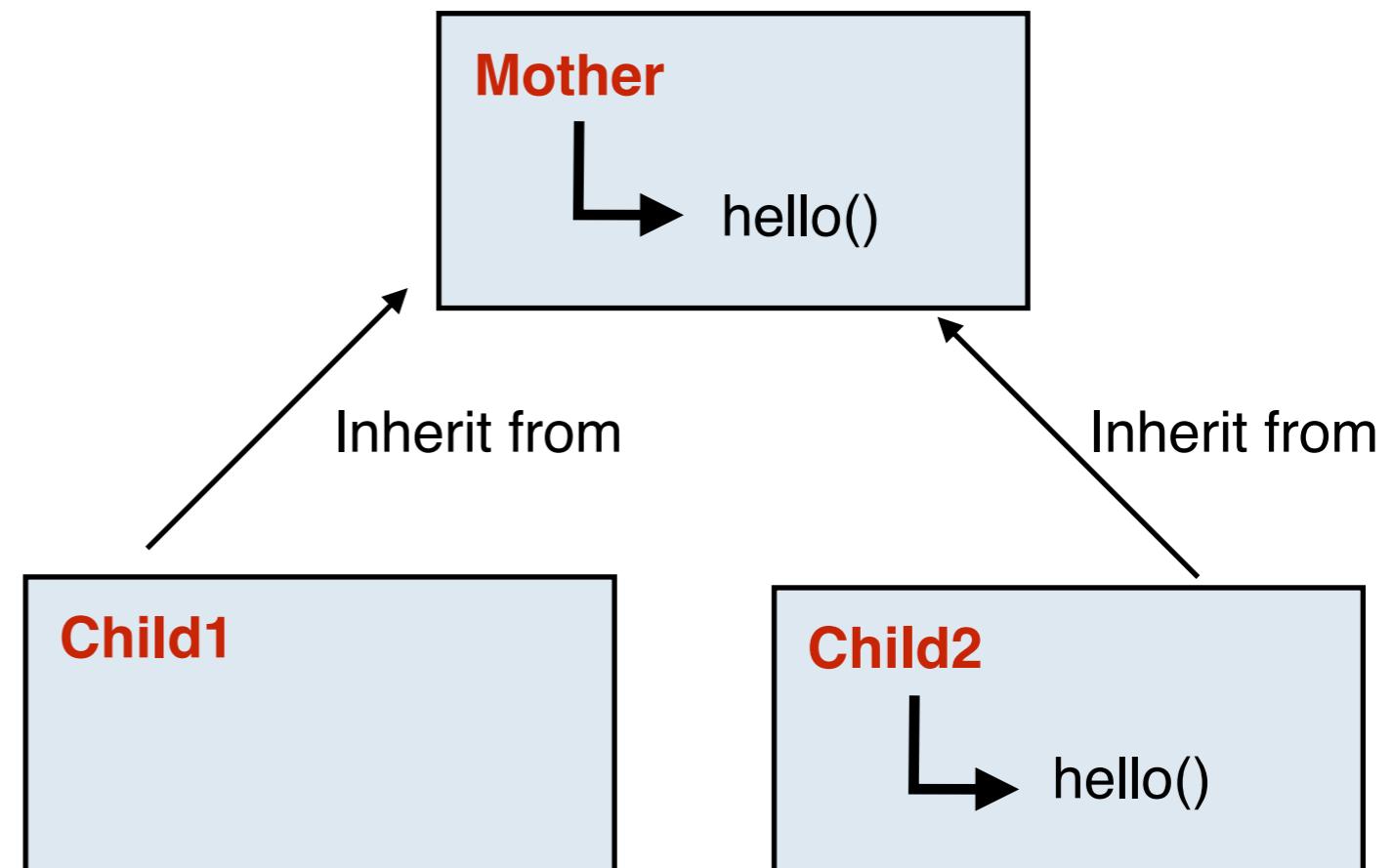
# Inheritance

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](#)

```
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 public:
15     void hello() {
16         Mother::hello();
17         cout<< "and from Child2" << endl;};
18     };
19
20 int main () {
21     Child1 test;
22     test.hello();
23
24     Child2 test2;
25     test2.hello();
26 }
27 }
```



# 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](#)

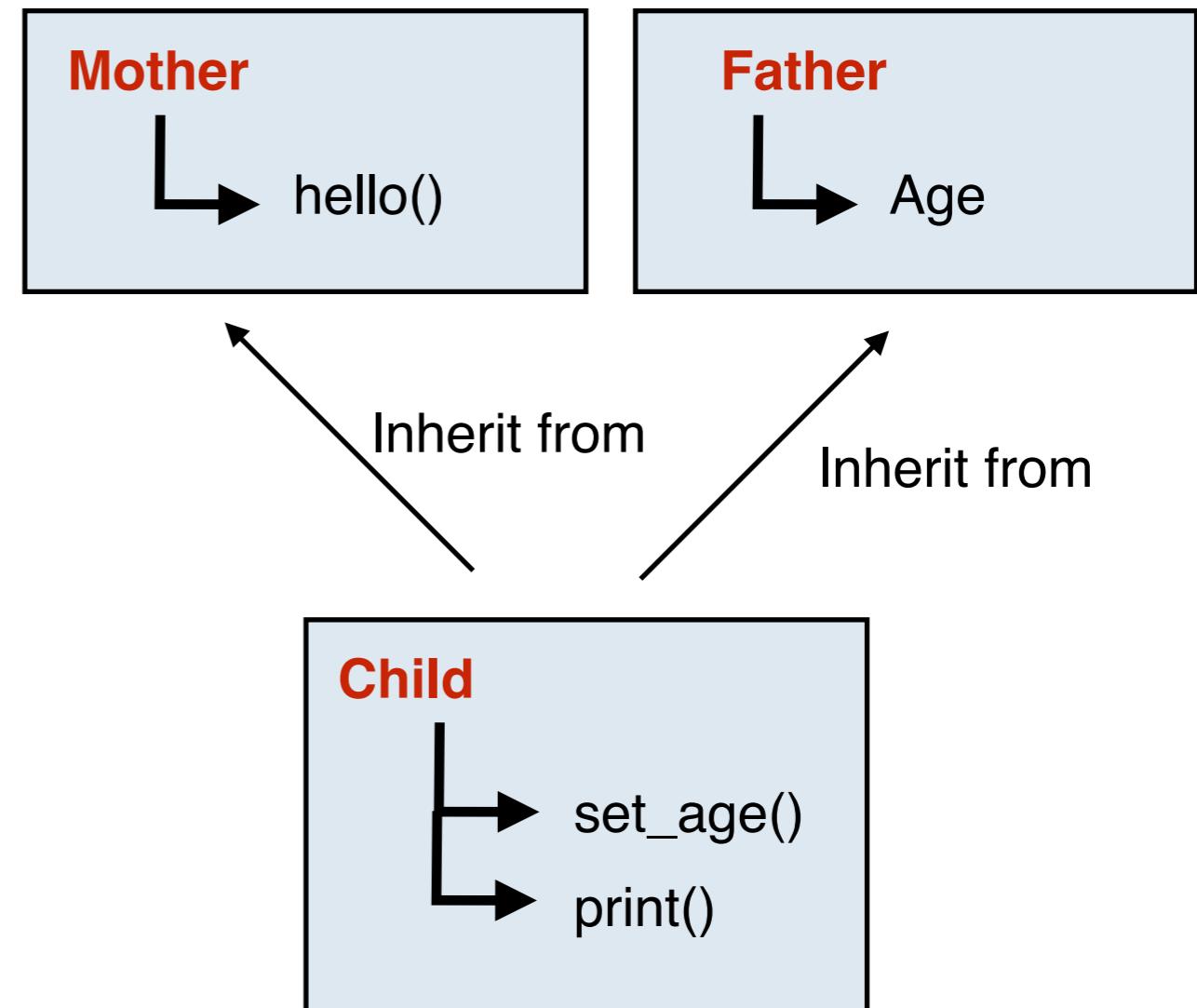
```
1 // example: class constructor
2 #include <iostream>
3 using namespace std;
4
5 class Mother{
6 public:
7     void hello(){
8         cout<< "hello from Mother" << endl;};
9 }
10
11 class Child1: public Mother{};
12
13 class Child2: public Mother{
14 public:
15     void hello() {
16         Mother::hello();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

- “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 exists!)
- Constructor/Destructor are **not** passed to the child
- Assignment operator (operator =) are **not** passed to the child

# Multi-inheritance

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 public:
22     Child(int x){age=x;};
23
24     void print() {hello(); cout<<"my age is " << age;};
25     void set_age(int x){age=x;};
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 }
```



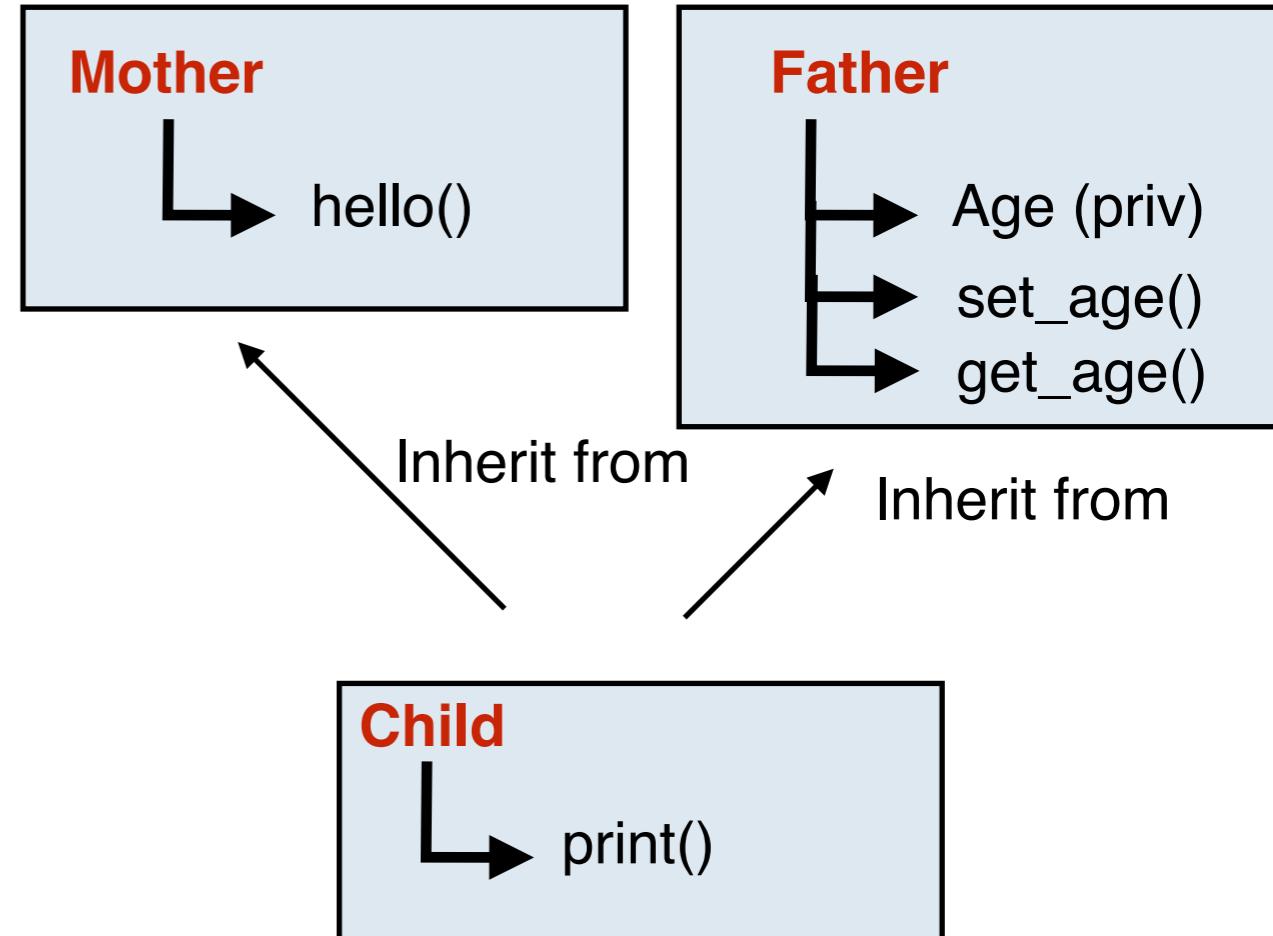
Can still call hello()

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 public:
23     Child(int x){set_age(x);};
24     void print() {hello(); cout<<"my age is " << get_age();};
25
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 III

- Define a class Four-Vector which inherit from your class 3 vector
  - Define the norm like in special relativity
    - ◆  $x^*x = x[0]x[0] - x[1]x[1] - x[2]x[2] - x[3]x[3]$
- 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.

# Solution

[cpp.sh/2jen](#)

```
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;};

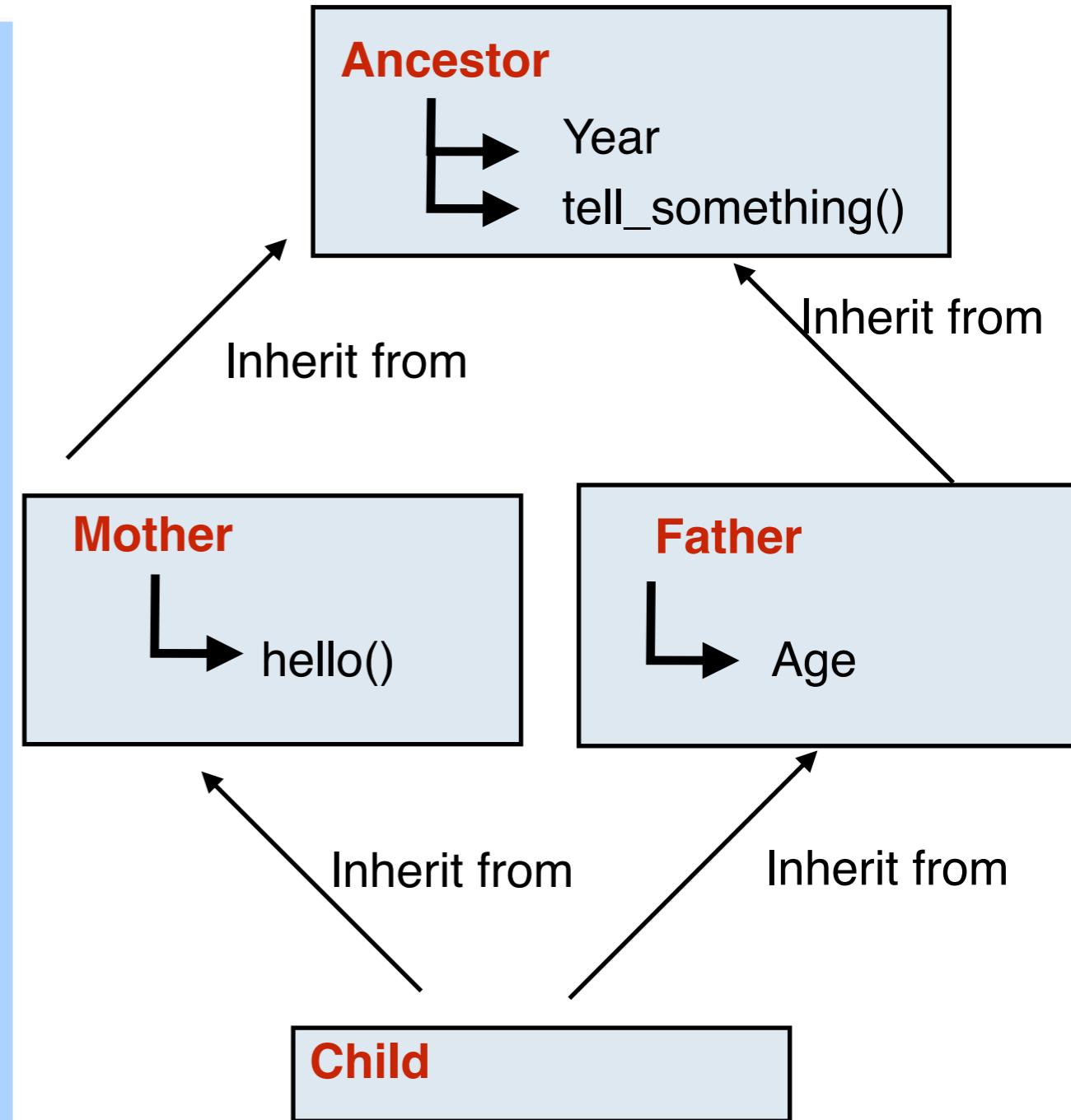
    float mass_gap(){return norm2()-mass*mass;};
};

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;
};
```

# Diamond Diagram

[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](#)

```
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
- 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

```
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](#)

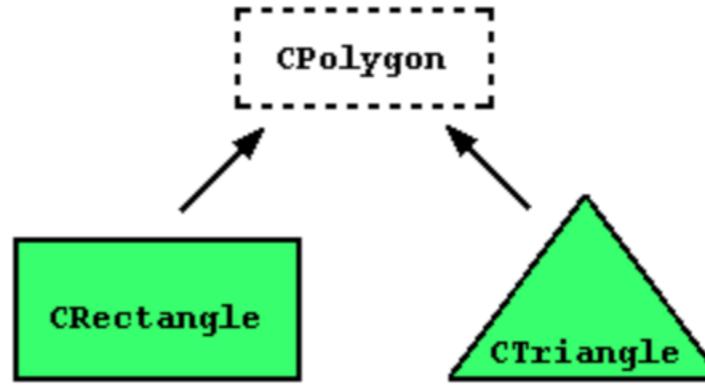
```
1 // function template
2 #include <iostream>
3 using namespace std;
4
5 template <class T>
6 T sum (T a, T b)
7 {
8     T result;
9     result = a + b;
10    return result;
11 }
12
13 int main () {
14     int i=5, j=6, k;
15     double f=2.0, g=0.5, h;
16     k=sum<int>(i,j);
17     h=sum<double>(f,g);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

# 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 `->`
- 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