Introduction to Object -Oriented Programming with C++

Olivier Mattelaer
UCLouvain
CP3 & CISM

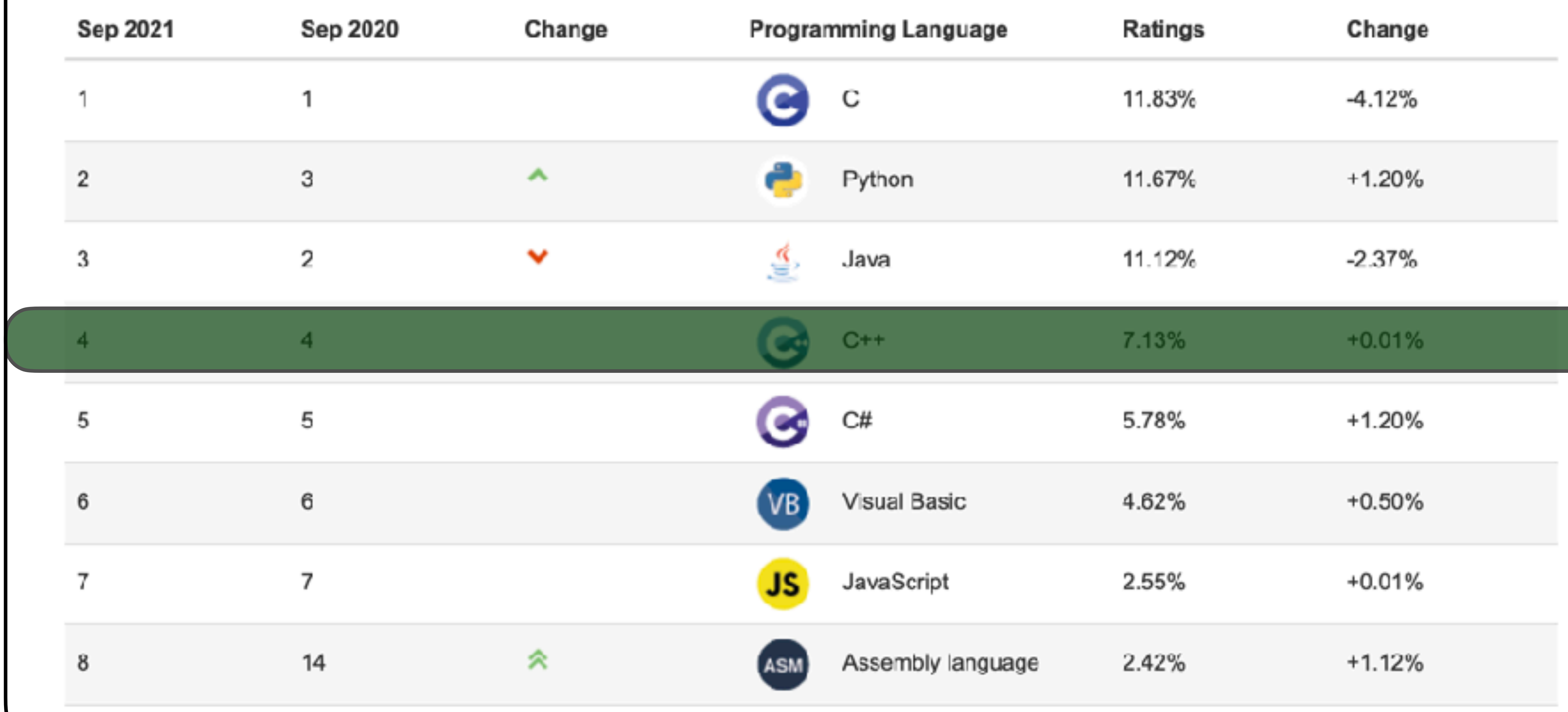
Programming paradigm

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 2021	Sep 2020	Change	Programming Language	Ratings	Change
1	1		C	11.83%	-4.12%
2	3	▲	Python	11.67%	+1.20%
3	2	▼	Java	11.12%	-2.37%
4	4		C++	7.13%	+0.01%
5	5		C#	5.78%	+1.20%
6	6		VB Visual Basic	4.62%	+0.50%
7	7		JS JavaScript	2.55%	+0.01%
8	14	▲▲	ASM Assembly language	2.42%	+1.12%

- 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) Inheritance
    - ➡ Presentation of concept
    - ➡ Code presentation
    - ➡ Exercise

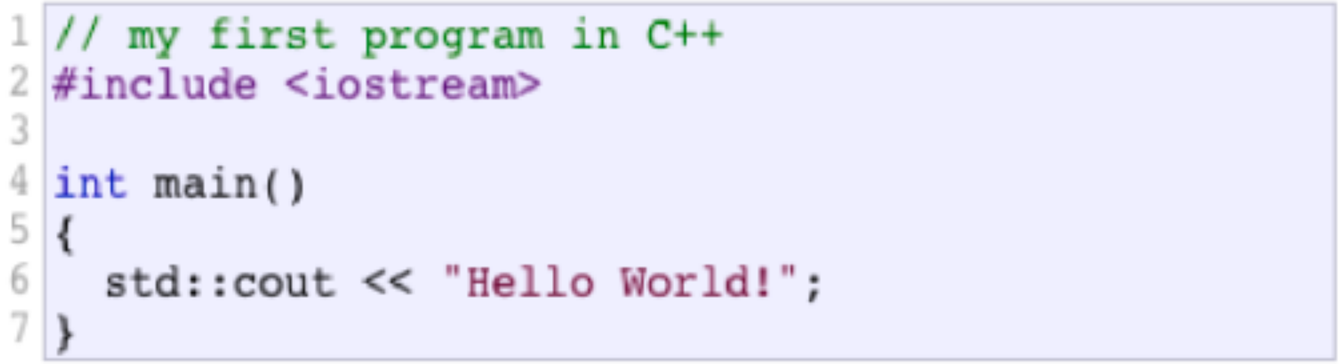
Program of today

- Basic of C++
  - ➡ Presentation of concept
  - ➡ Code presentation
- Introduction to Class/object in C++
  - ➡ Presentation of concept
  - ➡ Code presentation
  - ➡ Exercise
- (Mu[illegible]nce
  - [illegible]entation of concept
  - ➡ Code presentation
  - ➡ Exercise

Slides and examples/solutions are on indico

Tarball: hello_world.cpp

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

- line1: 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

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)

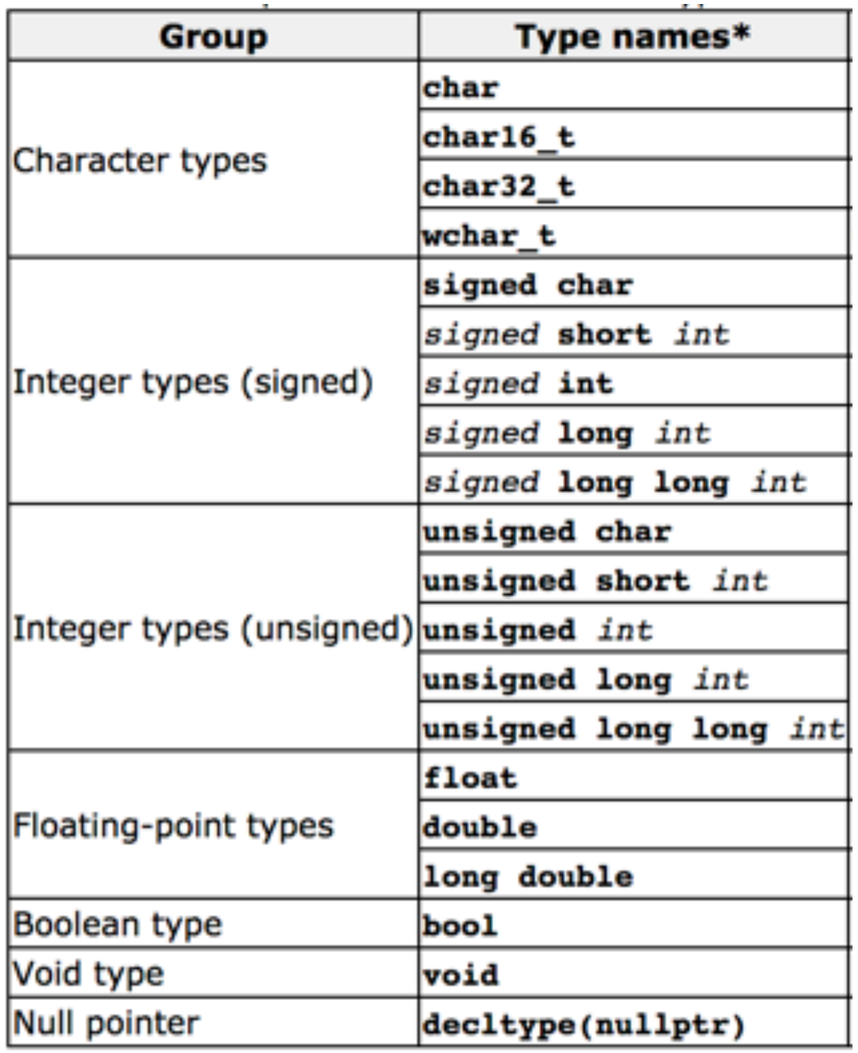
http://www.cpp.sh/2dd

https://www.tutorialspoint.com/compile_cpp_online.php

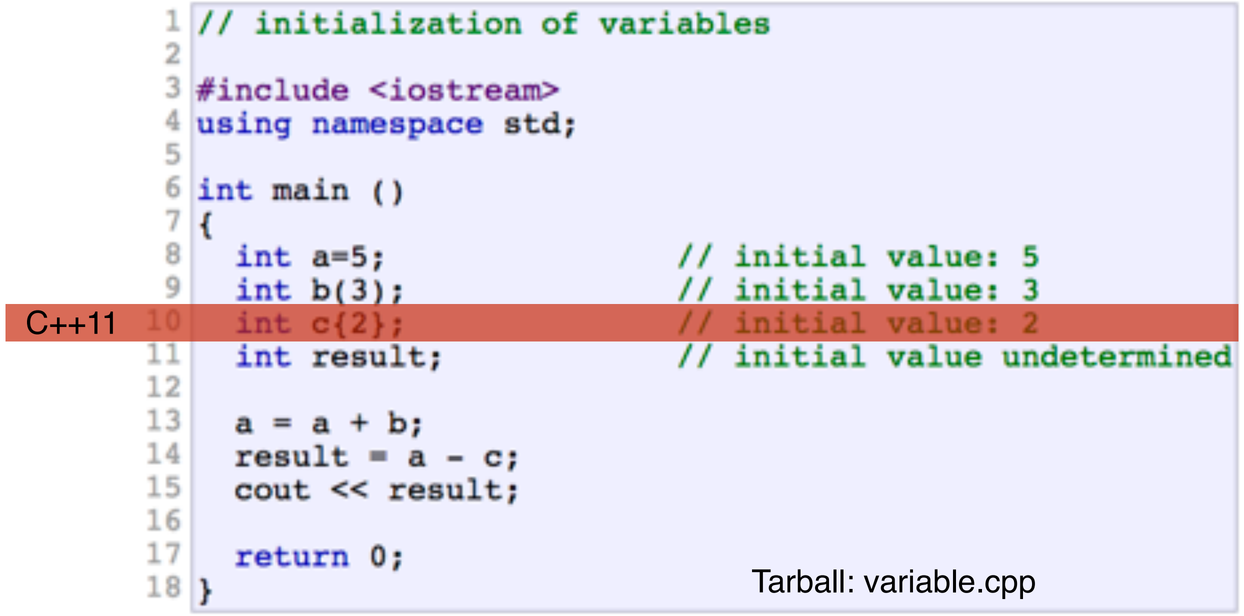
Basic of C++ : variables

Variable = portion of memory storing a value

- C++ is strongly typed
  - ➡ Need to know the type of variable
  - ➡ Optimize memory



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

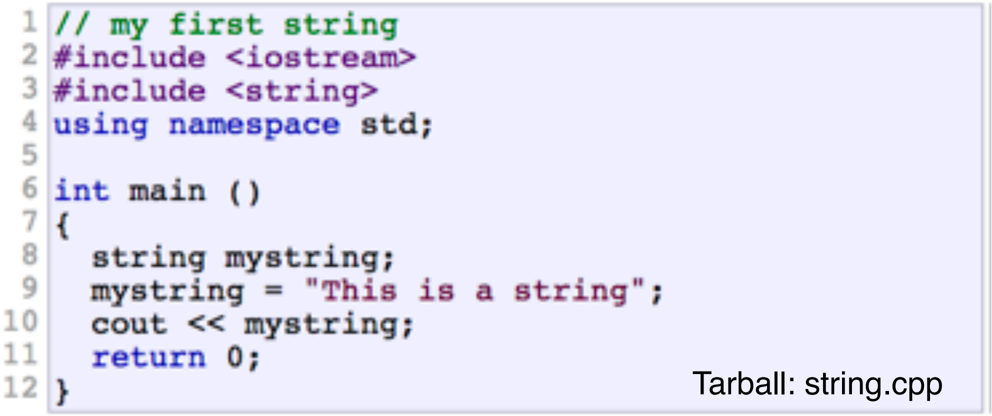


```
// initialization of variables

#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;
}
```

C++11

Tarball: variable.cpp

http://cpp.sh/8yl



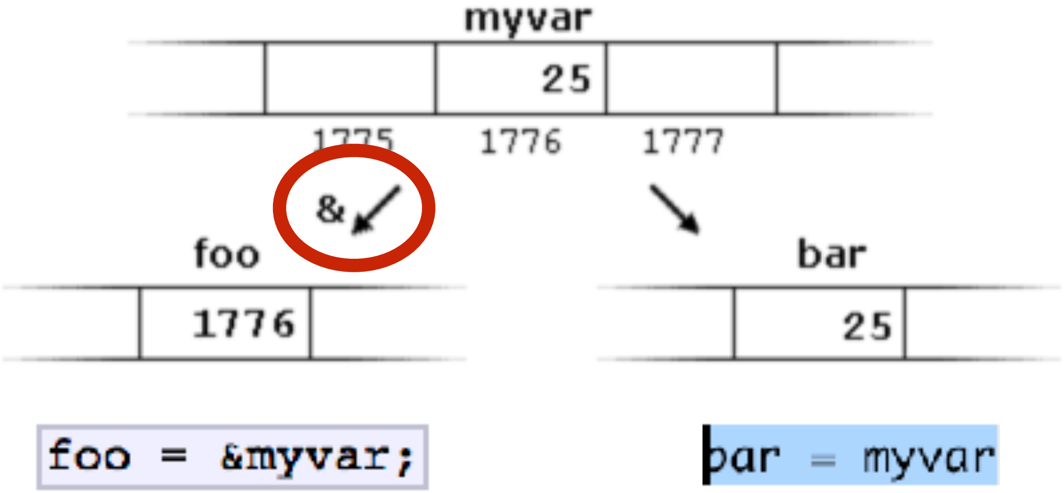
```
// my first string
#include <iostream>
#include <string>
using namespace std;

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

Tarball: string.cpp

http://cpp.sh/7d4

Basic of C++: pointer

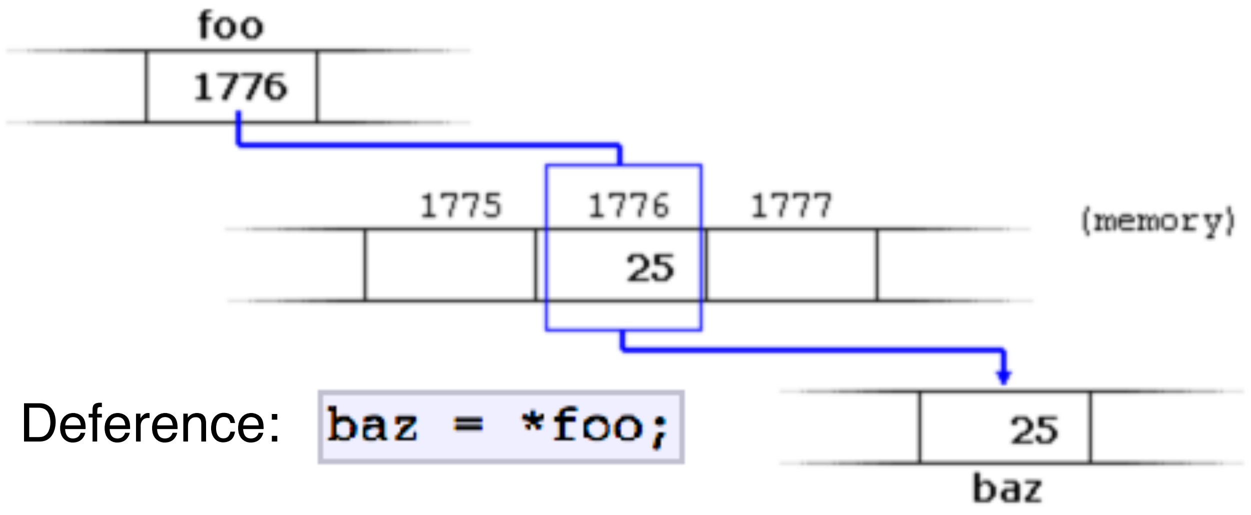
Pointer = variable containing the adress of another variable



```
foo = &myvar;
```

```
bar = myvar
```

```
foo = &myvar;
```



Deference: `baz = *foo;`

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

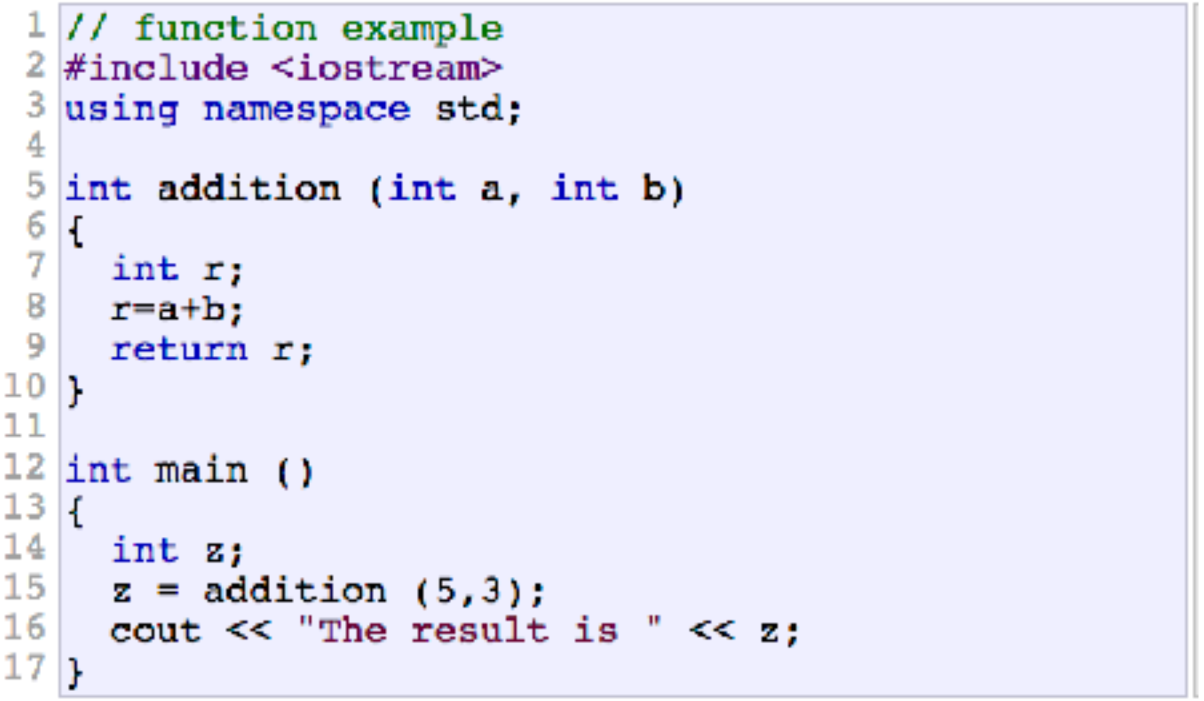
```
int * number;
char * character;
double * decimals;
```

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

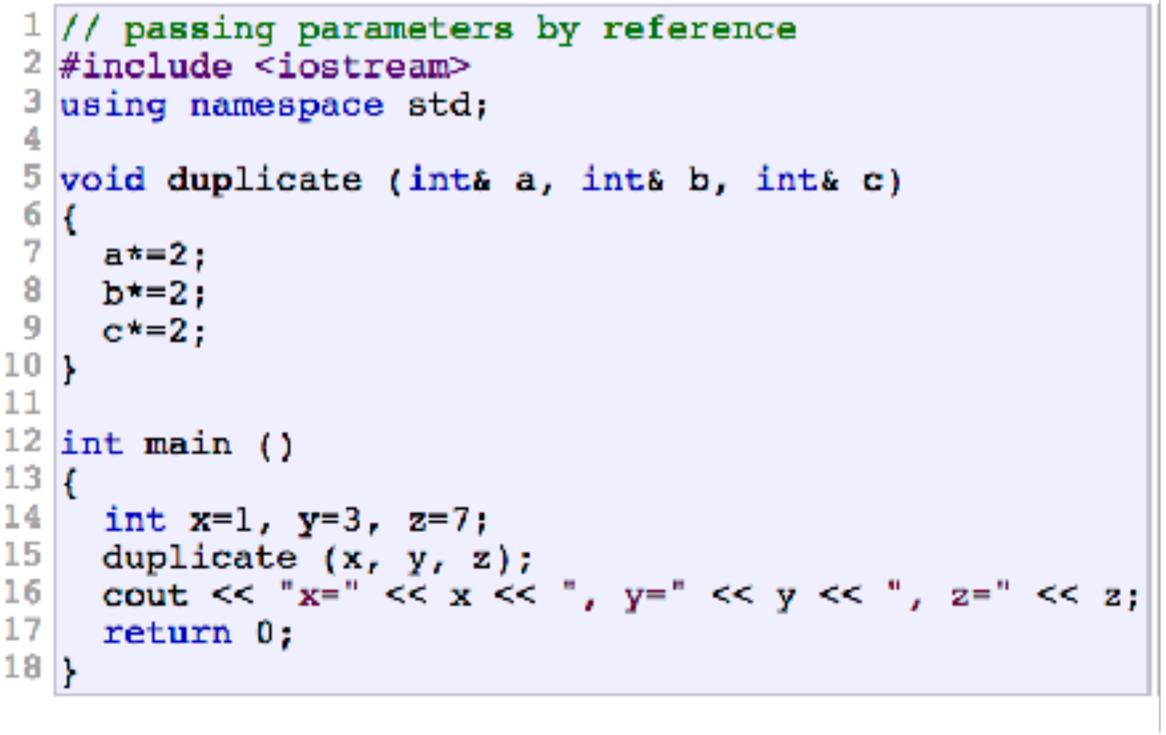


```cpp
// function example
#include <iostream>
using namespace std;

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;
}
```

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

Passing Parameters by reference

http://cpp.sh/9b2



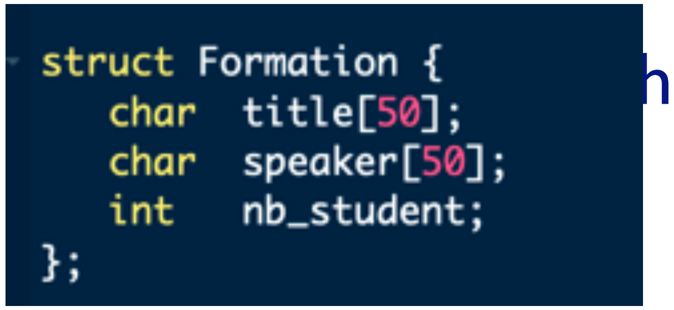
```cpp
// passing parameters by reference
#include <iostream>
using namespace std;

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;
}
```

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

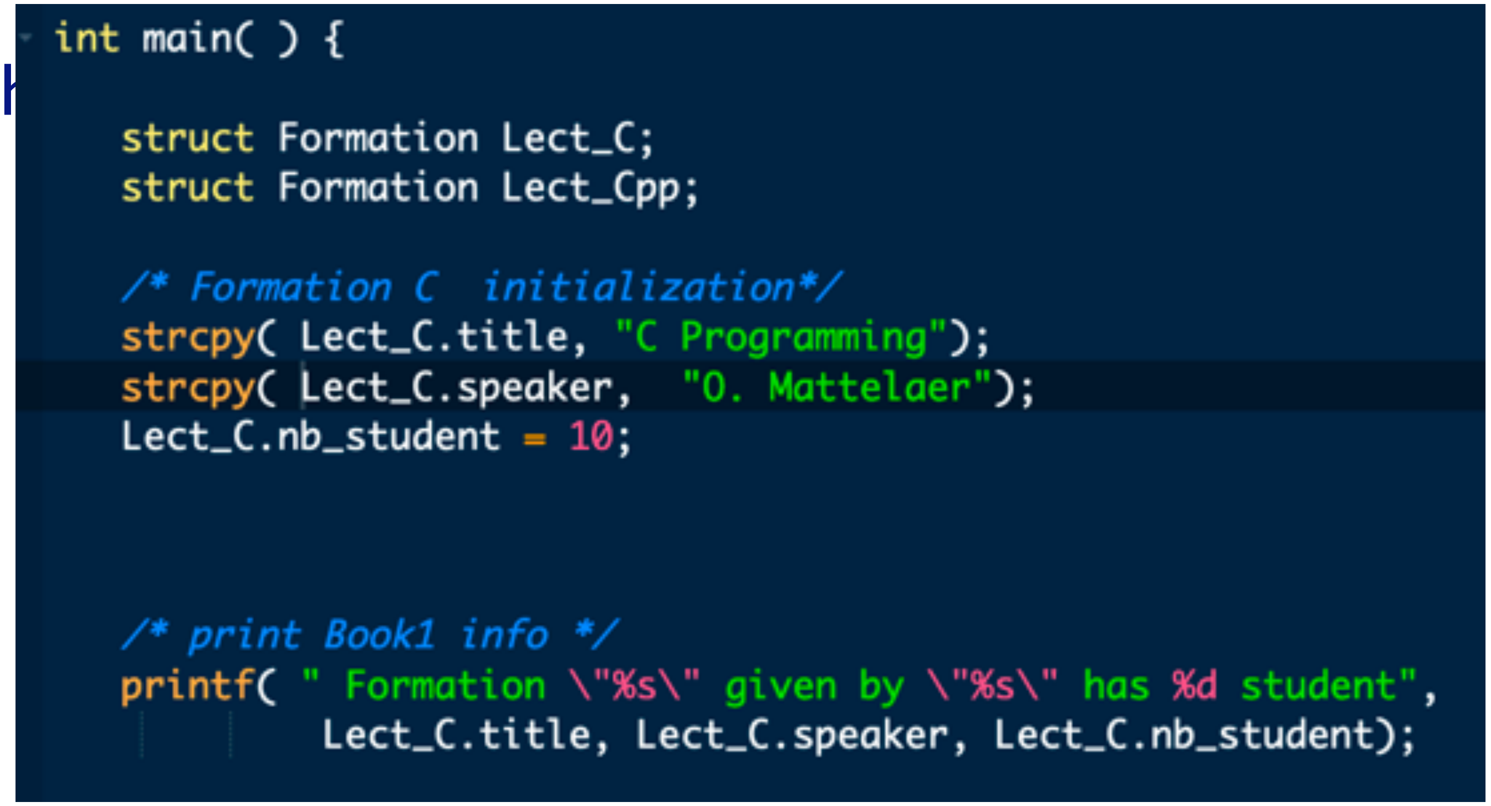
Tarball; fct_by_ref.cpp

Data structure

- Can we have a special data-type with metadata
  - ➡ Like a “formation”
    - ✦ With the number of student
    - ✦ The name of the formation
    - he name of th



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

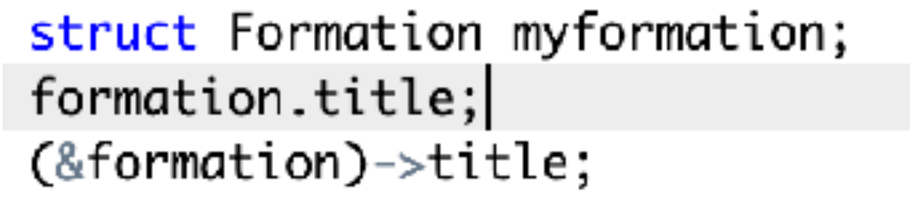


```
int main( ) {

   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);
```

http://tpcg.io/umjalDnr

More on Data structure

- Access data:
  - ➡ From variable use the “.”
  - ➡ From pointer use the “->”



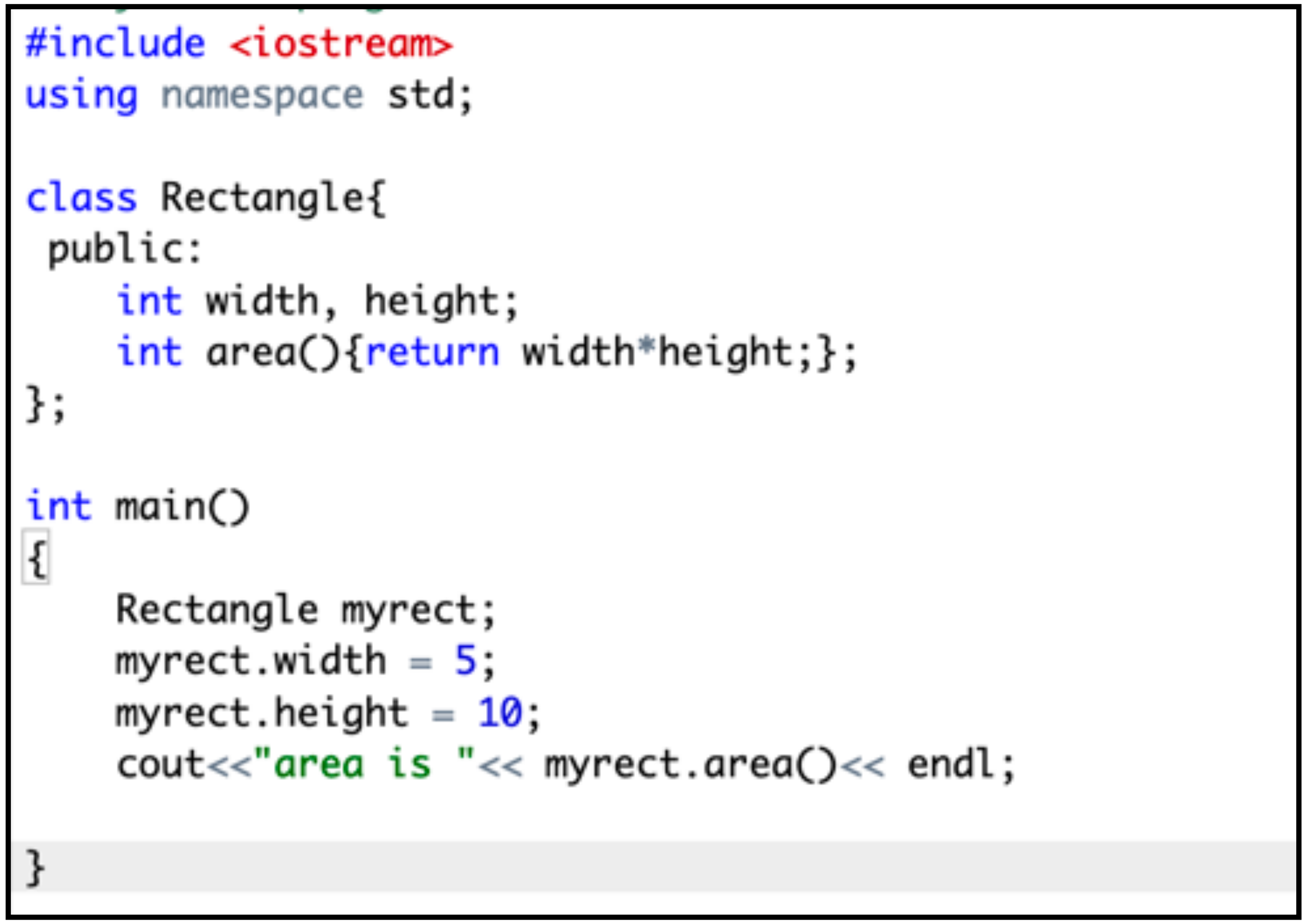
```
struct Formation myformation;
formation.title;
(&formation)->title;
```

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;

}
```

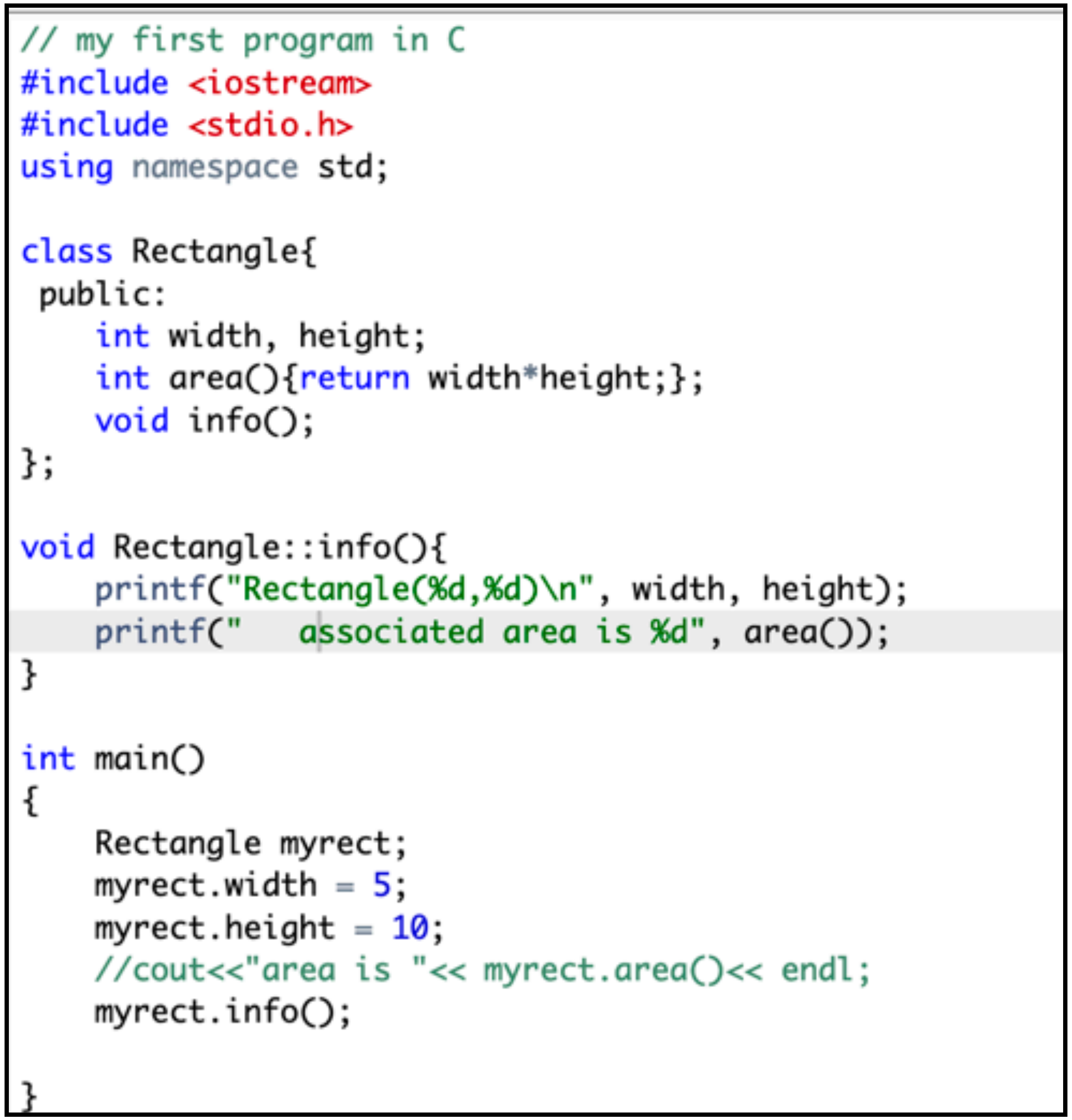
Tarball: class_public_1.cpp

- “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();

}
```

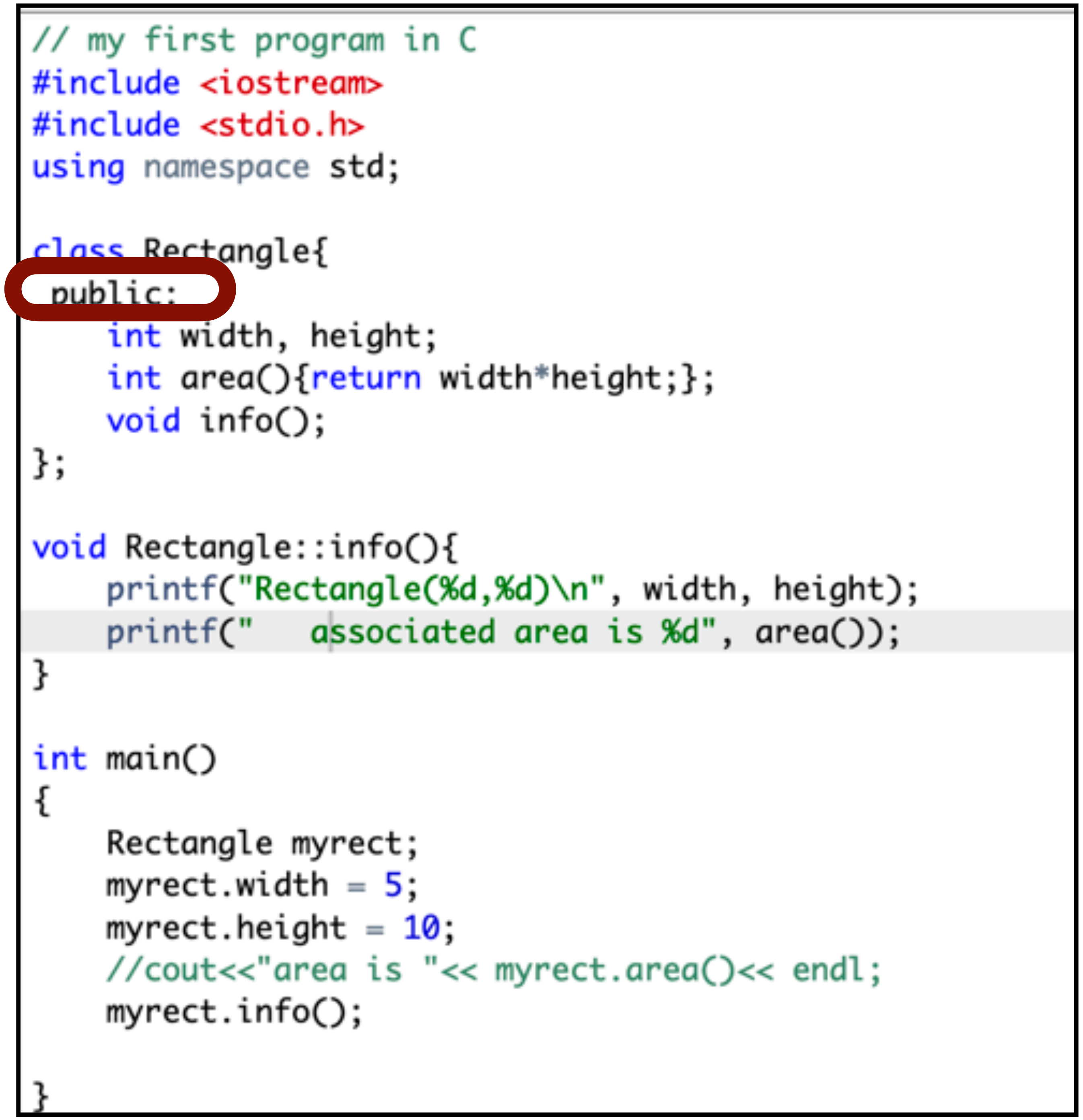
- As for normal function, you do not have to define the full function in the class definition you can postpone it.
- Note that we do not define width/height inside the function

Tarball: class_public_2.cpp

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();

}
```

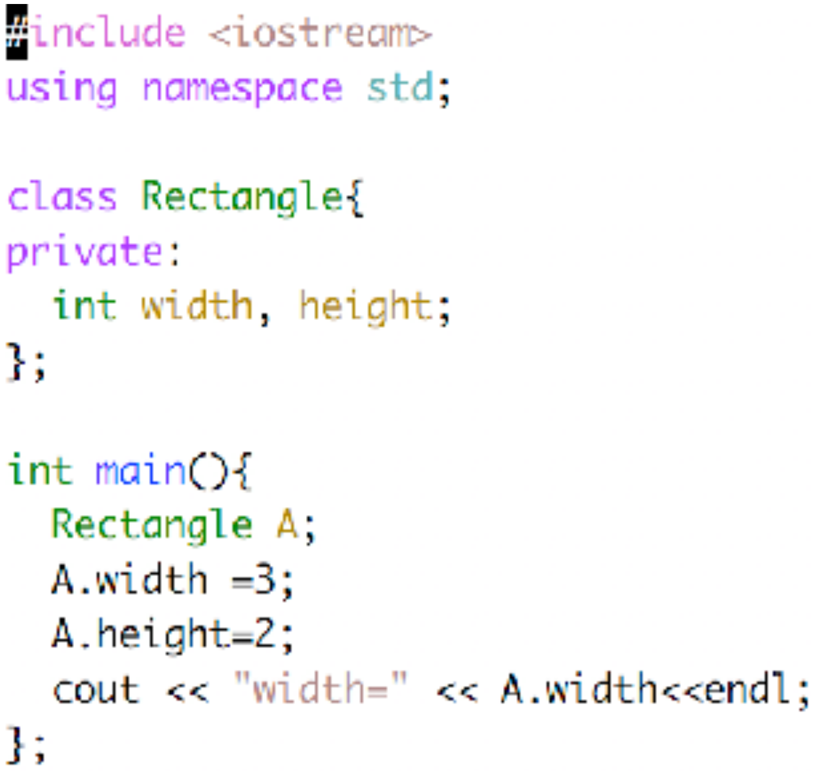
- Allows distinction between
  - Visible information
  - Internal mechanism

cpp.sh/34lna

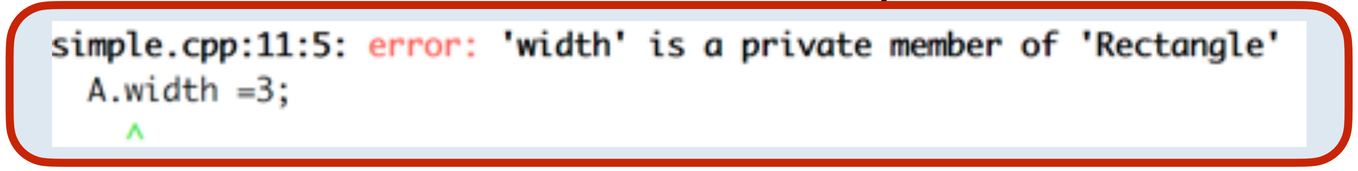
Tarball: class_public_1.cpp

Visibility of attribute/function

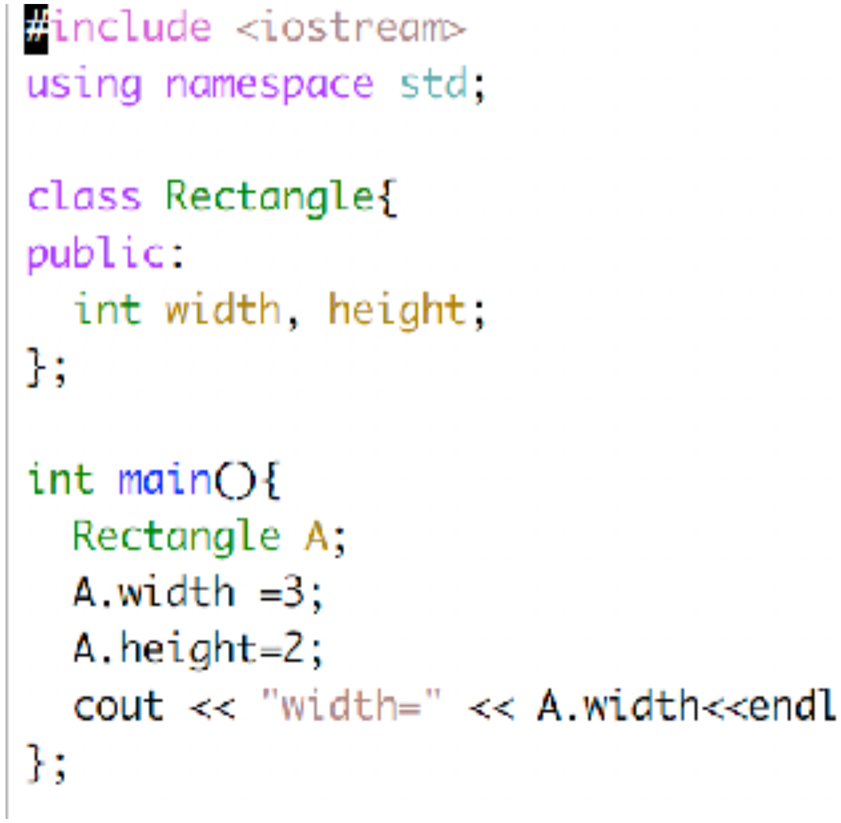
private	protected	public
Only accessible from other instance of the same class Accessible from friends DEFAULT	Accessible from other instance of the same class Accessible from friends Accessible from instance of the derived/child class	Accessible from everywhere where the object is visible READ and WRITE!



```
#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;
};
```

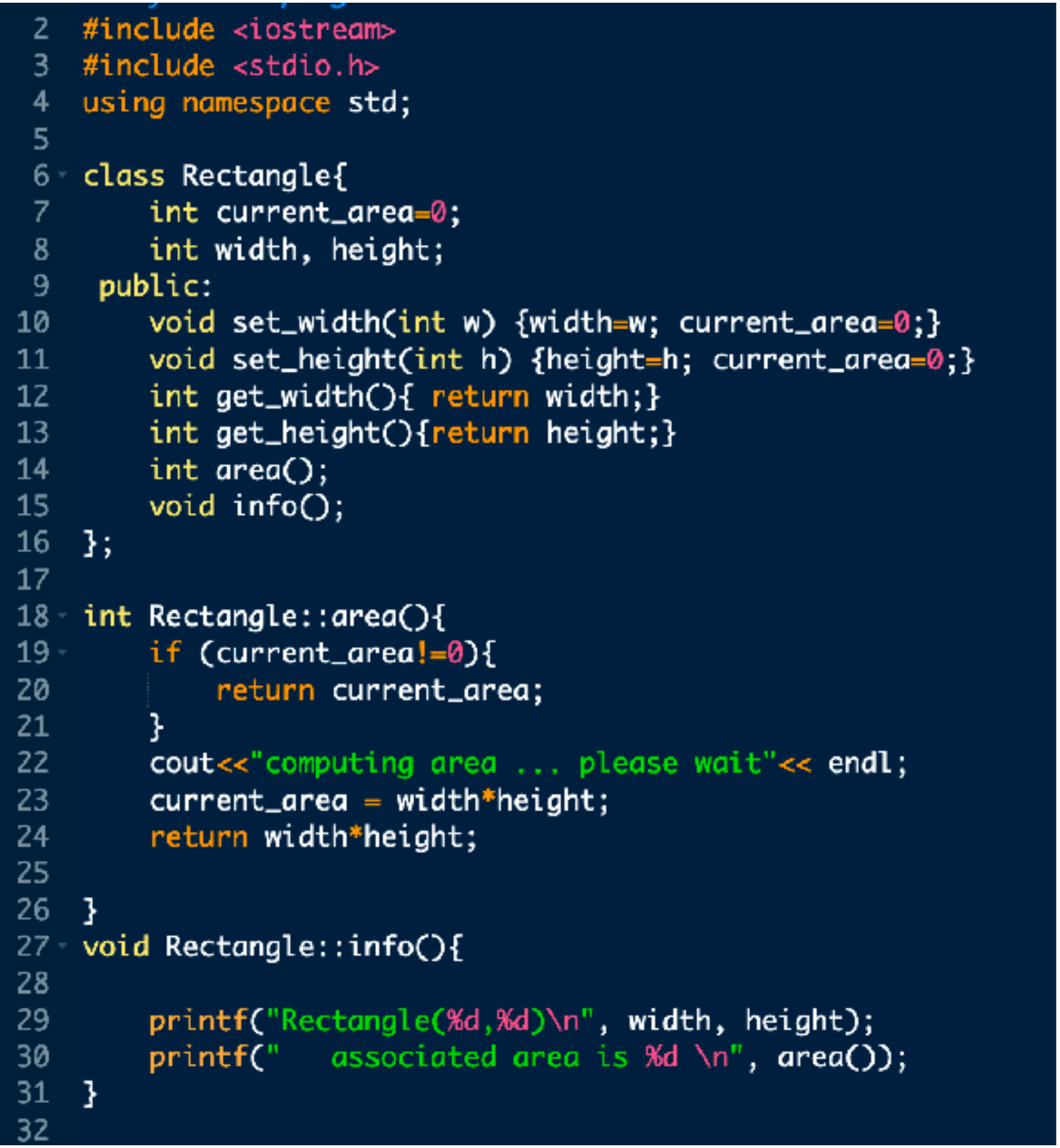


```
simple.cpp:11:5: error: 'width' is a private member of 'Rectangle'
  A.width =3;
    ^
```



```
#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



```
#include <iostream>
#include <stdio.h>
using namespace std;

class Rectangle{
    int current_area=0;
    int width, height;
 public:
    void set_width(int w) {width=w; current_area=0;}
    void set_height(int h) {height=h; current_area=0;}
    int get_width(){ return width;}
    int get_height(){return height;}
    int area();
    void info();
};

int Rectangle::area(){
    if (current_area!=0){
        return current_area;
    }
    cout<<"computing area ... please wait"<< endl;
    current_area = width*height;
    return width*height;

}
void Rectangle::info(){

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

```

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

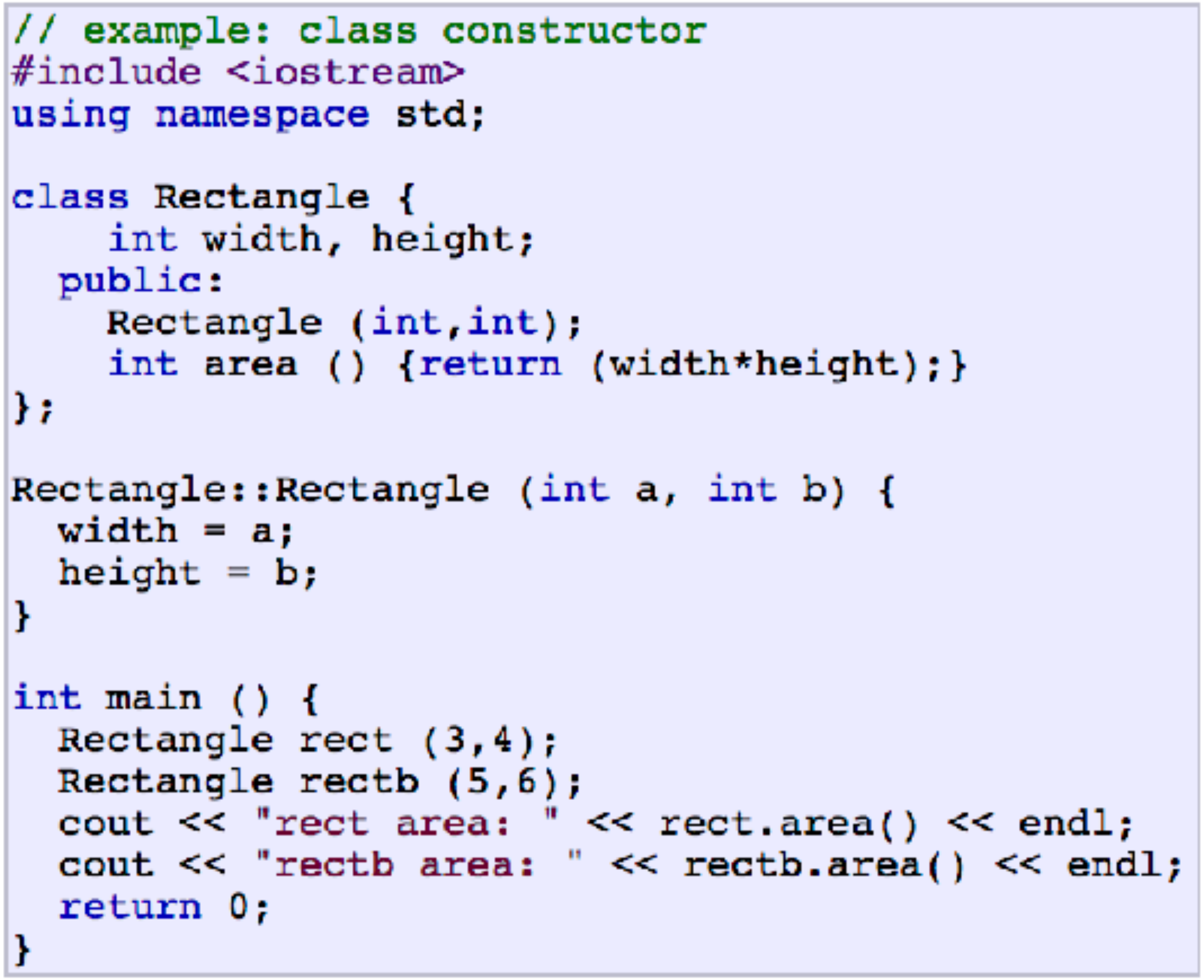
Tarball: class_private.cpp

http://tpcg.io/bKCfmxxQ

Constructor

constructor = function called after the object is created

cpp.sh/8lr

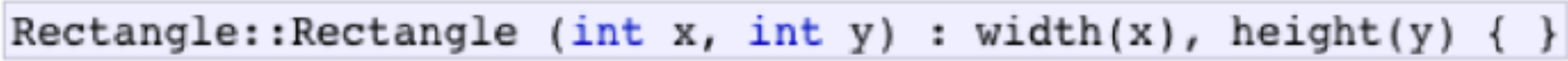


```
// example: class constructor
#include <iostream>
using namespace std;

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;
}
```

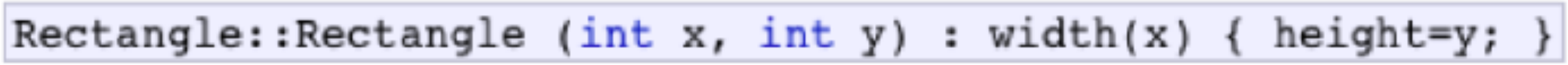
- The name of the constructor is the name of the function itself!

Tarball: constructor.cpp

- Shortcut for setting attribute



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

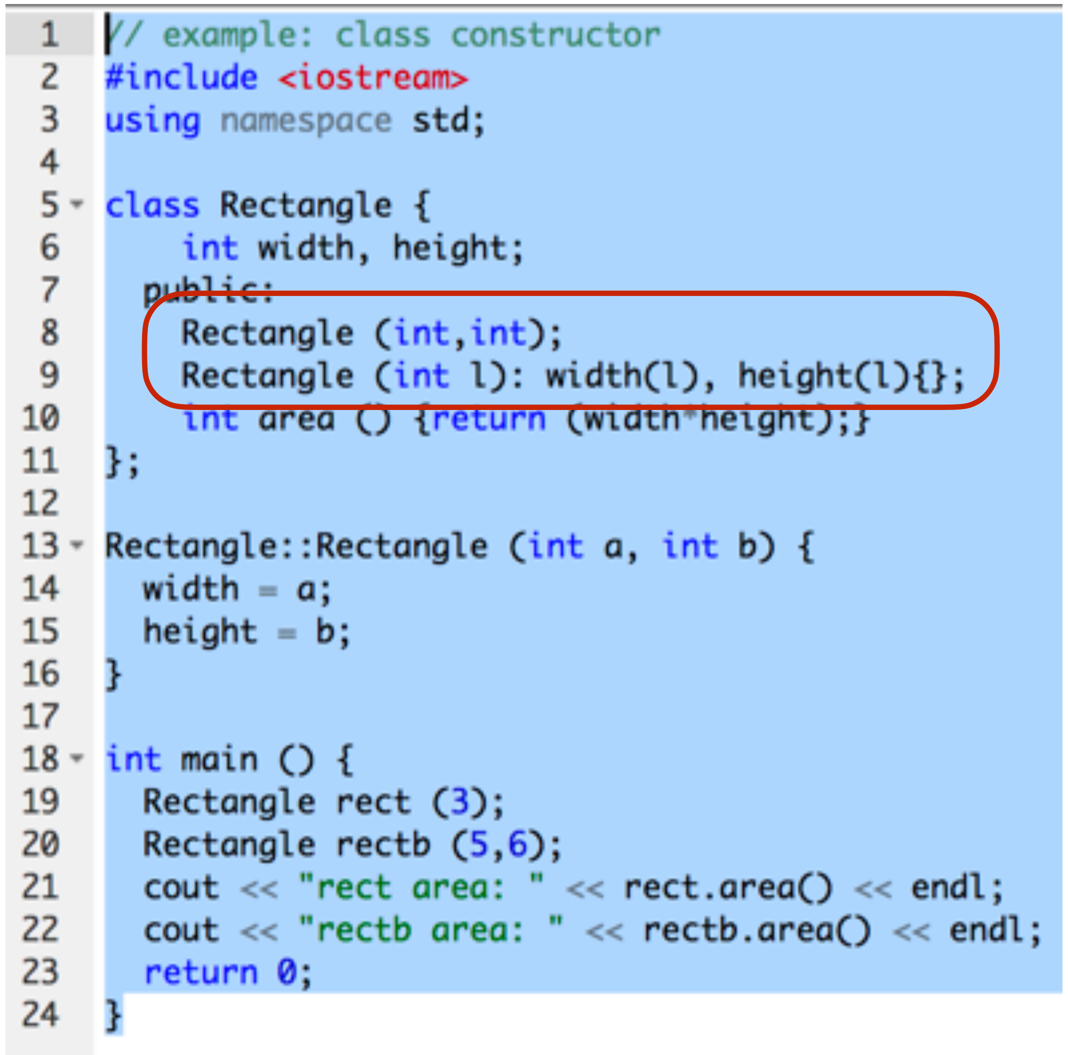


```
Rectangle::Rectangle (int x, int y) : width(x) { height=y; }
```

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.

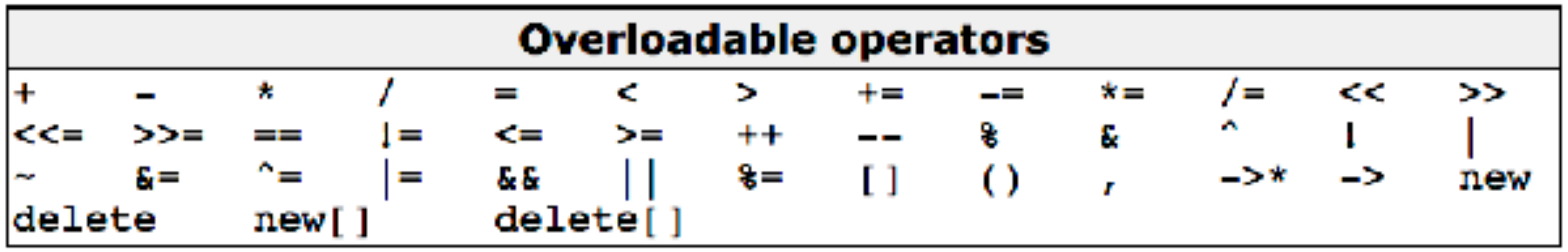


```
// example: class constructor
#include <iostream>
using namespace std;

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 +

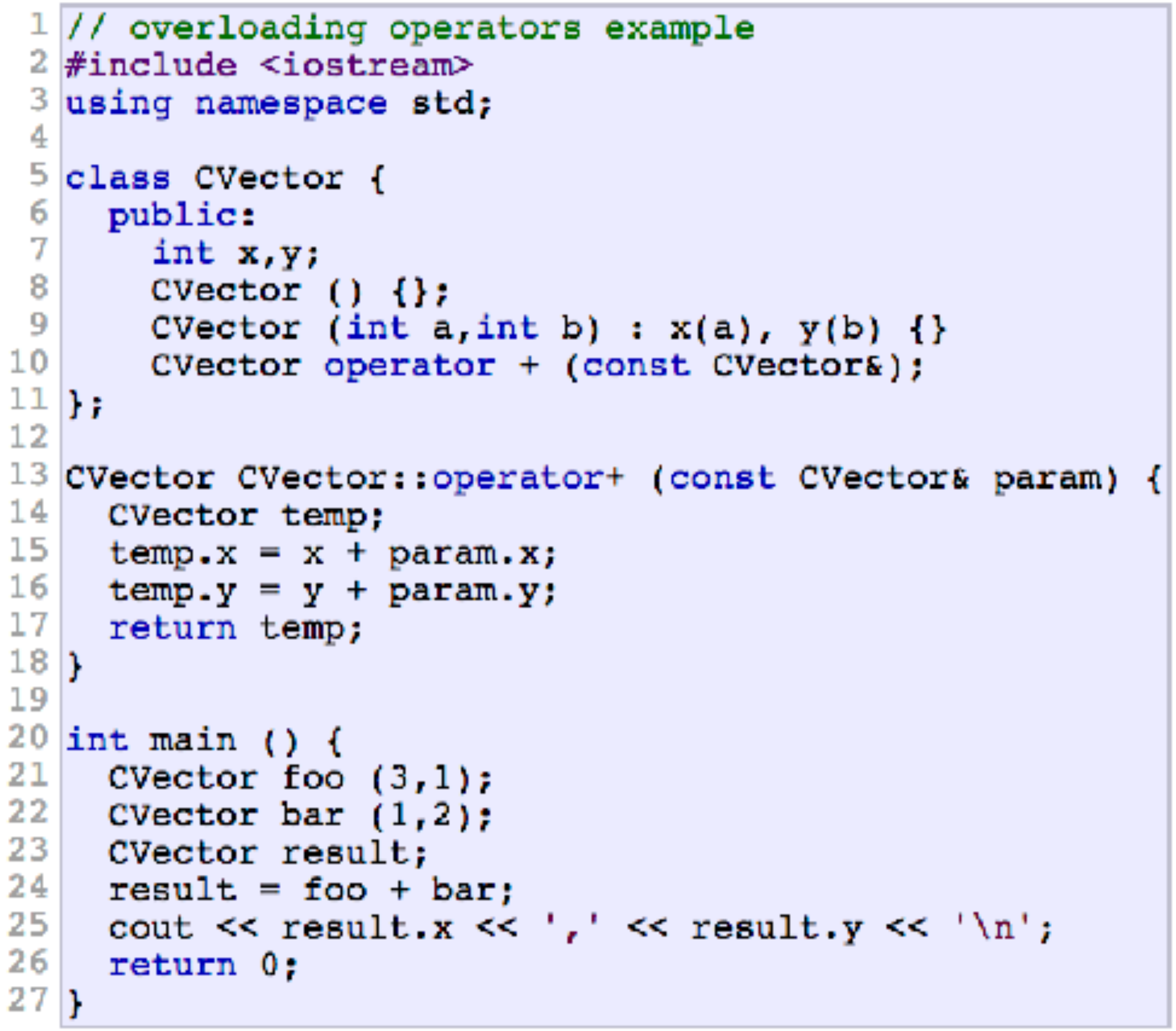
Overloading

Overloading = more than one function with the same name



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

cpp.sh/27l



```
// 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;
}
```

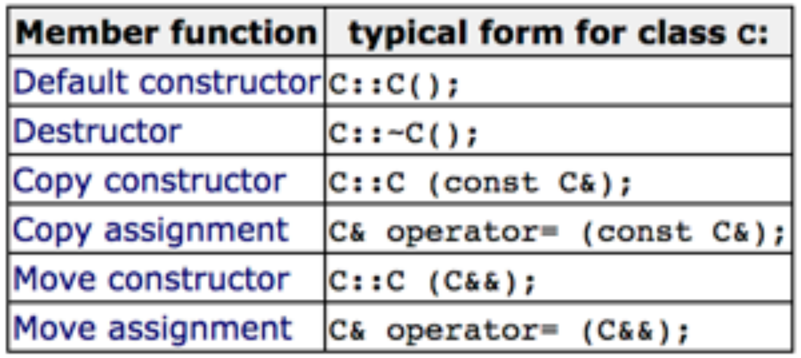
More details for the syntax on

https://www.tutorialspoint.com/cplusplus/cpp_overloading.htm

Tarball: overloading.cpp

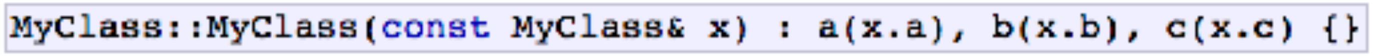
Special members

Special members = member functions implicitly defined

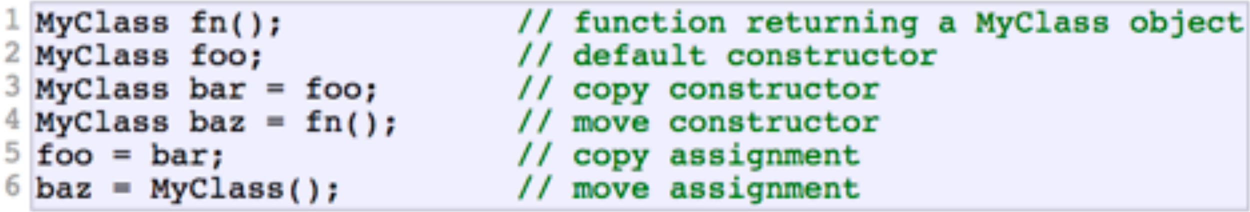


Member function	typical form for class C:
Default constructor	`C::C();`
Destructor	`C::~C();`
Copy constructor	`C::C (const C&);`
Copy assignment	`C& operator= (const C&);`
Move constructor	`C::C (C&&);`
Move assignment	`C& operator= (C&&);`

- 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



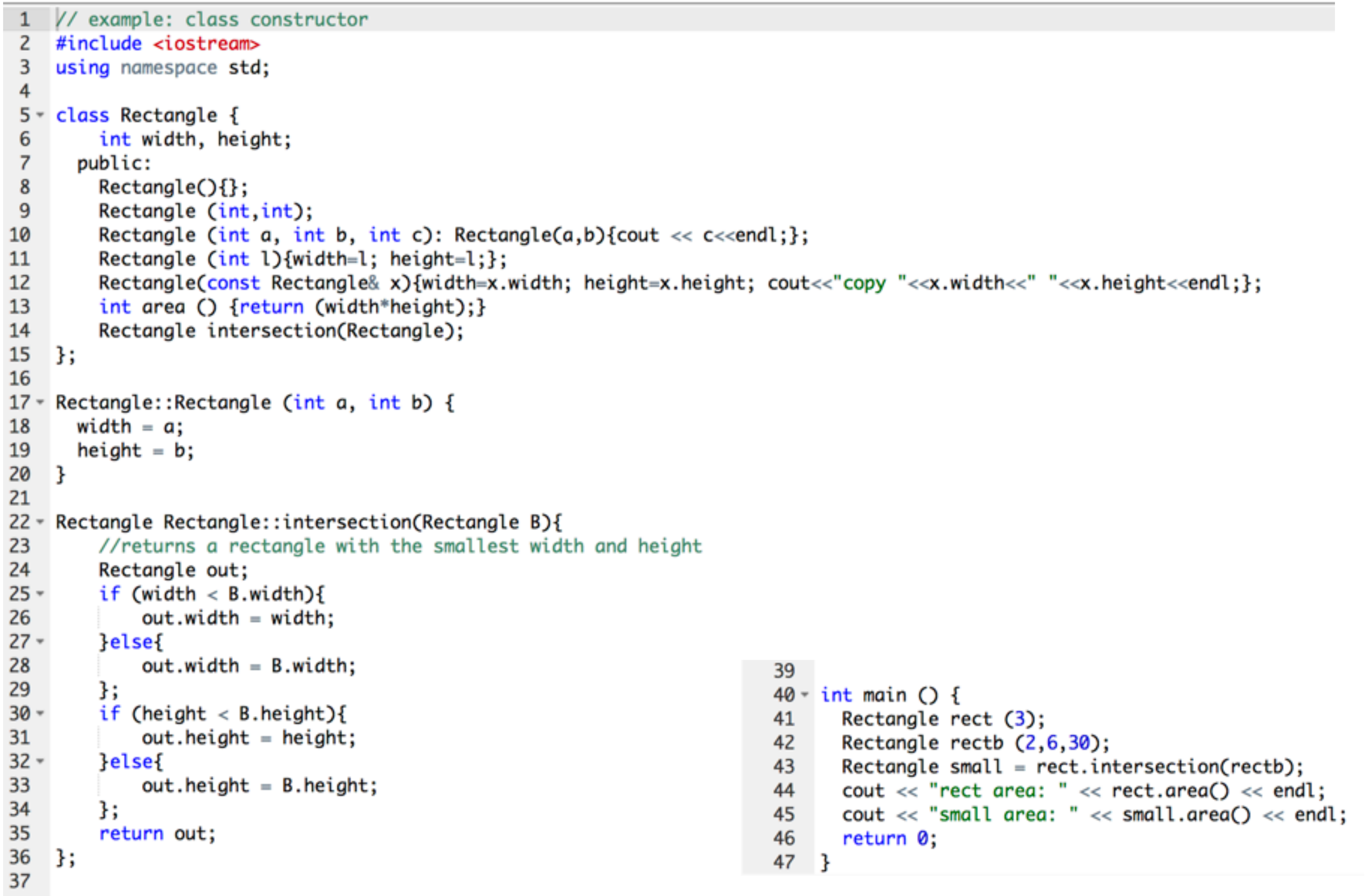
```
MyClass::MyClass(const MyClass& x) : a(x.a), b(x.b), c(x.c) {}
```



```
MyClass fn();            // function returning a MyClass object
MyClass foo;             // default constructor
MyClass bar = foo;       // copy constructor
MyClass baz = fn();      // move constructor
foo = bar;               // copy assignment
baz = MyClass();         // move assignment
```

cpp.sh/3zgct

Example



```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;
};

```

```cpp

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;
}
```

Tarball: copy_constructor.cpp

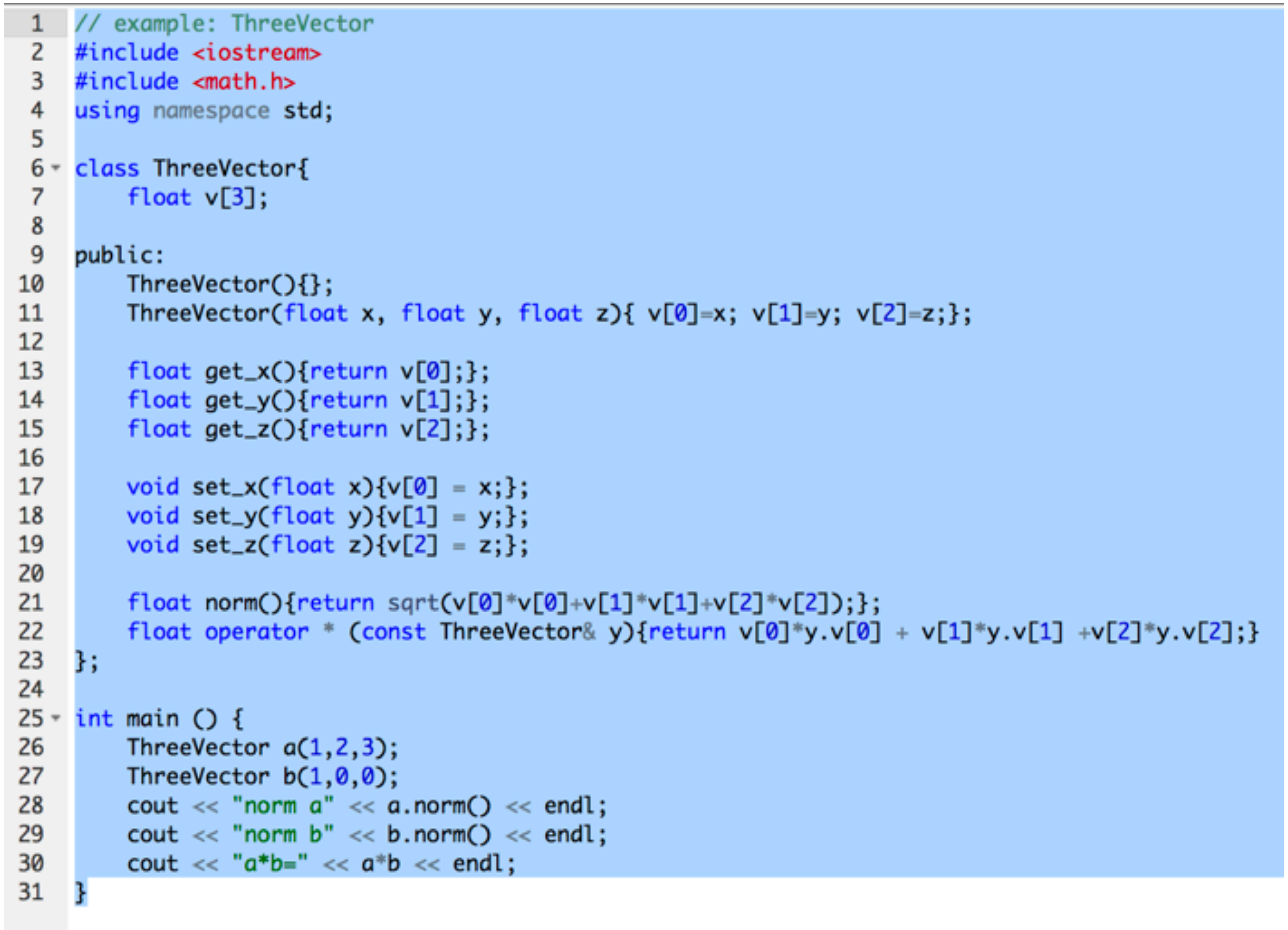
Exercise 1

- 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

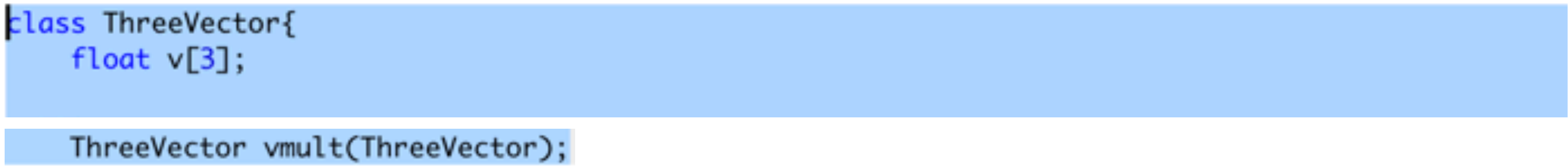
cpp.sh/6vgu2c



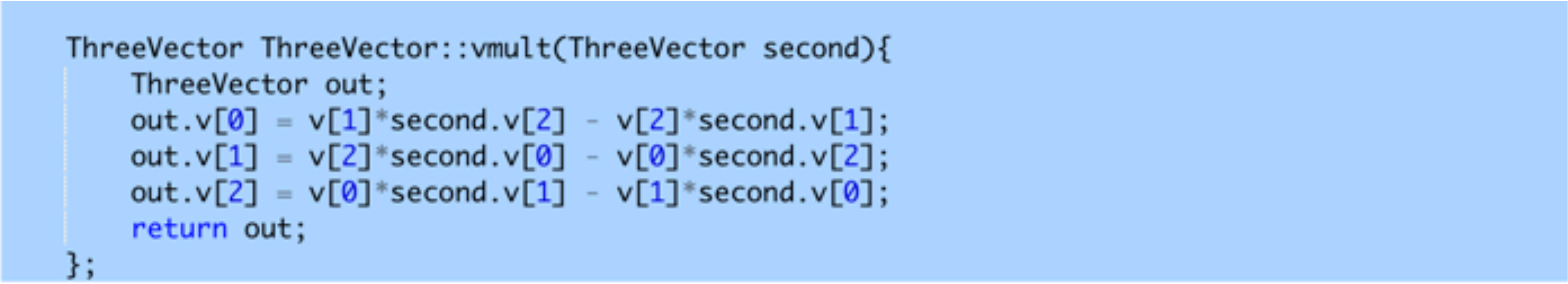
```
// example: ThreeVector
#include <iostream>
#include <math.h>
using namespace std;

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;
}
```

Tarball: solution_threevector.cpp

Solution

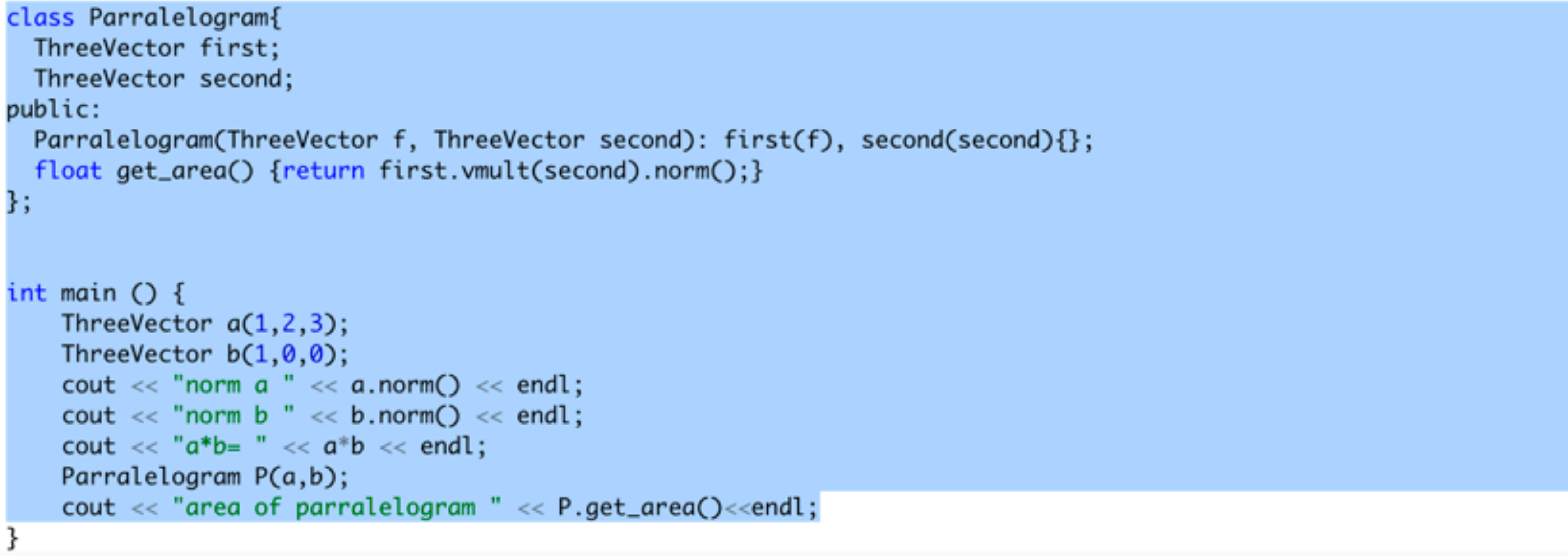


```
class ThreeVector{
    float v[3];

    ThreeVector vmult(ThreeVector);
```



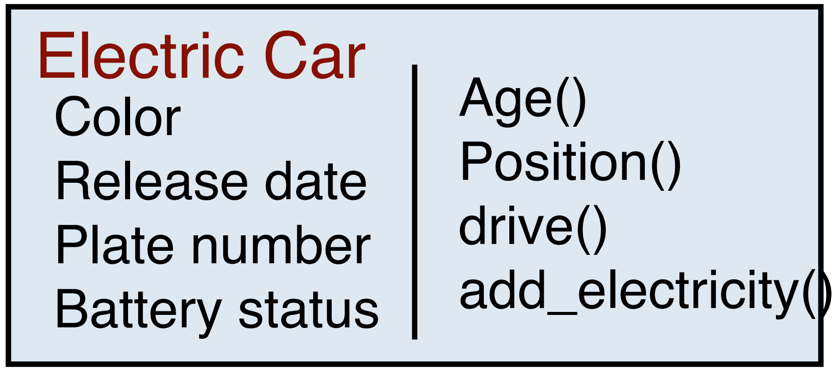
```
    ThreeVector ThreeVector::vmult(ThreeVector second){
        ThreeVector out;
        out.v[0] = v[1]*second.v[2] - v[2]*second.v[1];
        out.v[1] = v[2]*second.v[0] - v[0]*second.v[2];
        out.v[2] = v[0]*second.v[1] - v[1]*second.v[0];
        return out;
    };
```

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;
}
```

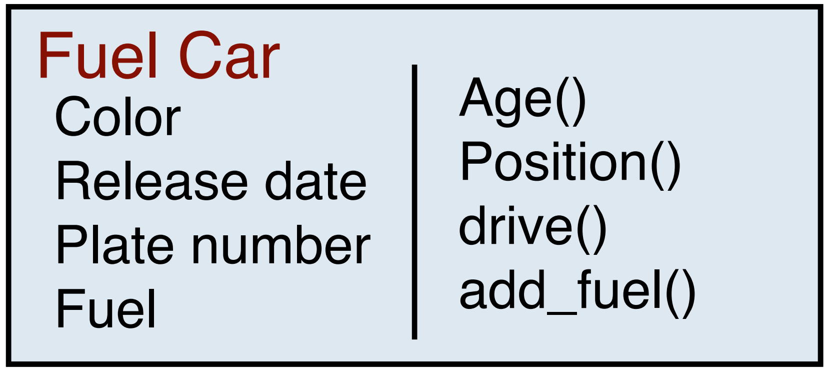
Inheritance



Electric Car

Color
Release date
Plate number
Battery status

Age()
Position()
drive()
add_electricity()

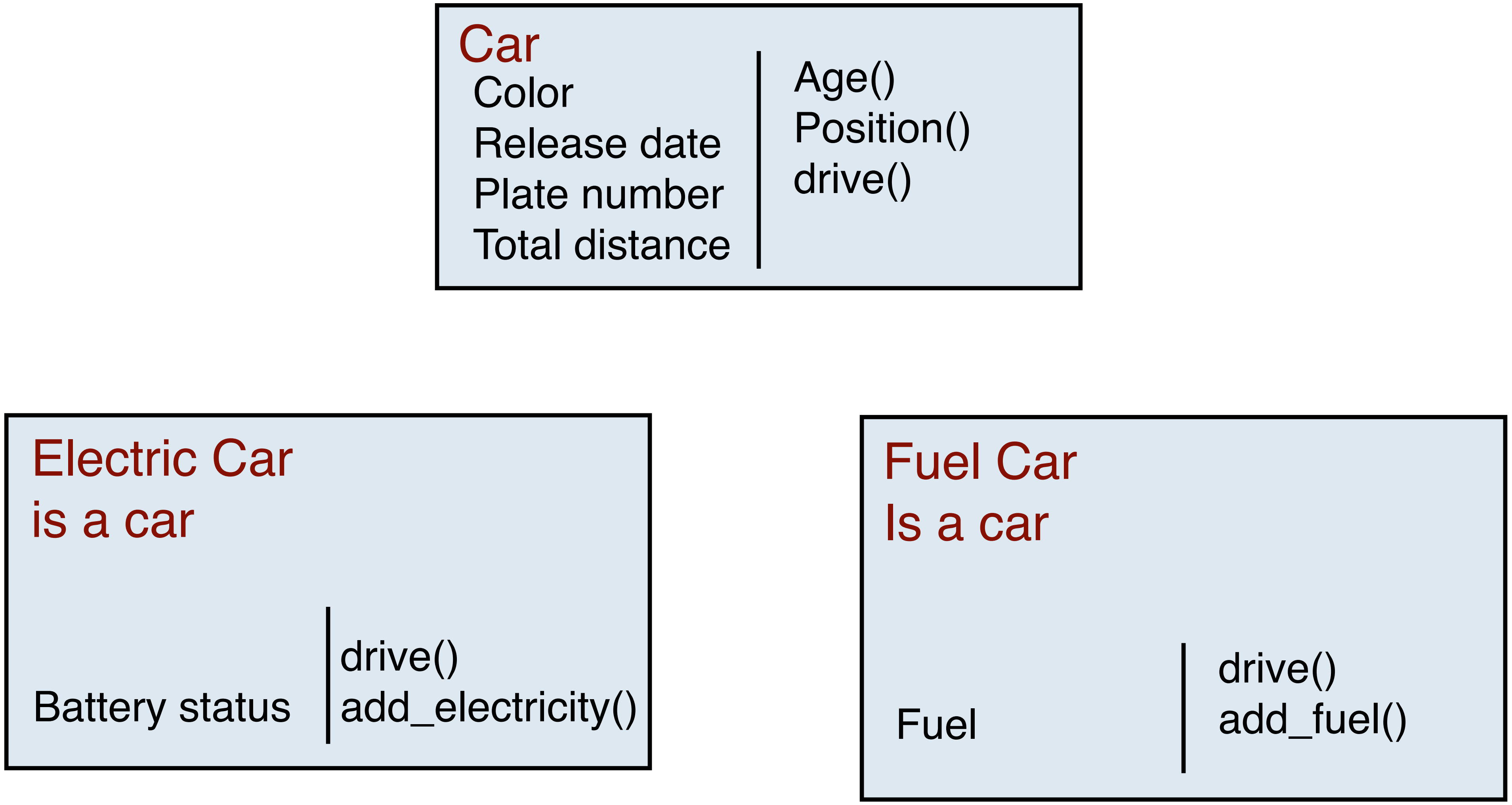


Fuel Car

Color
Release date
Plate number
Fuel

Age()
Position()
drive()
add_fuel()

Inheritance



Car

Color
Release date
Plate number
Total distance

Age()
Position()
drive()

Electric Car
is a car

Battery status

drive()
add_electricity()

Fuel Car
Is a car

Fuel

drive()
add_fuel()

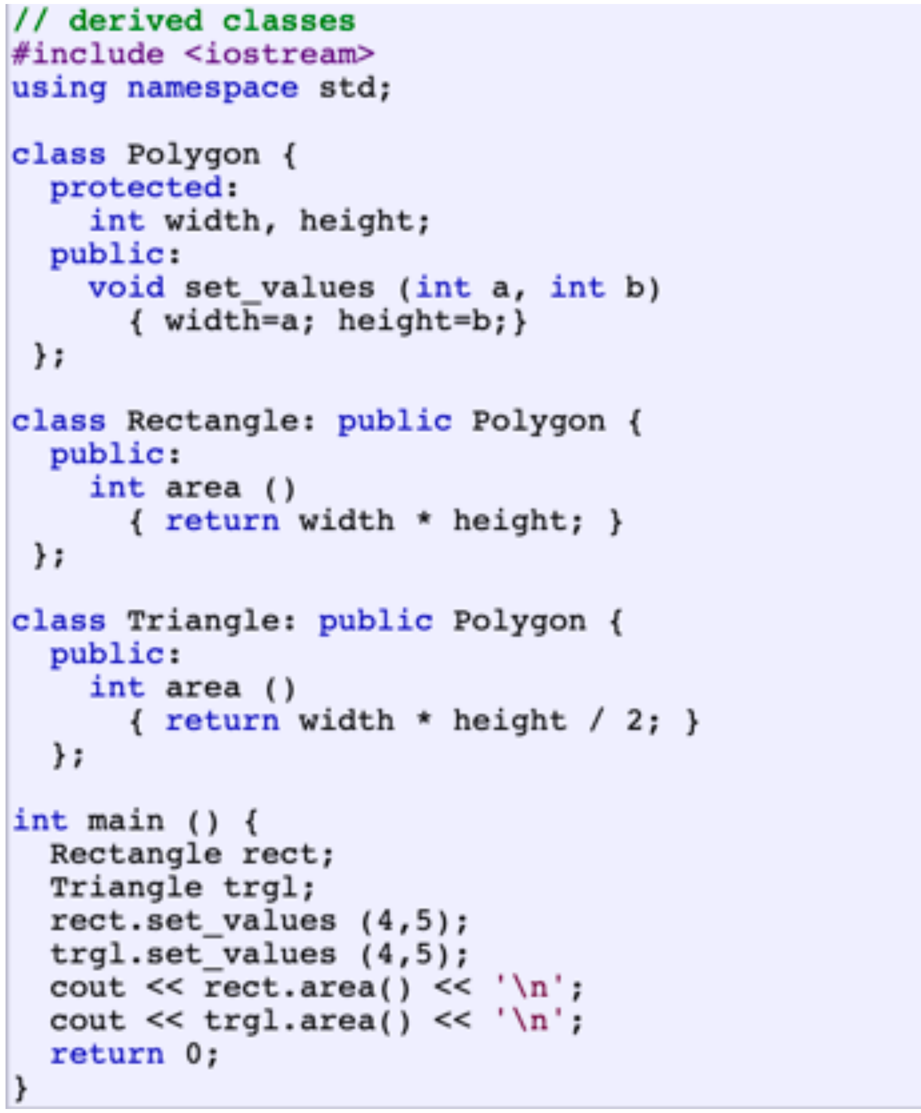
- The two class (Electric/fuel car) does not to redefine their structure just what they changed compare to the original class!
- They can change or superseed the behaviour

Inheritance

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

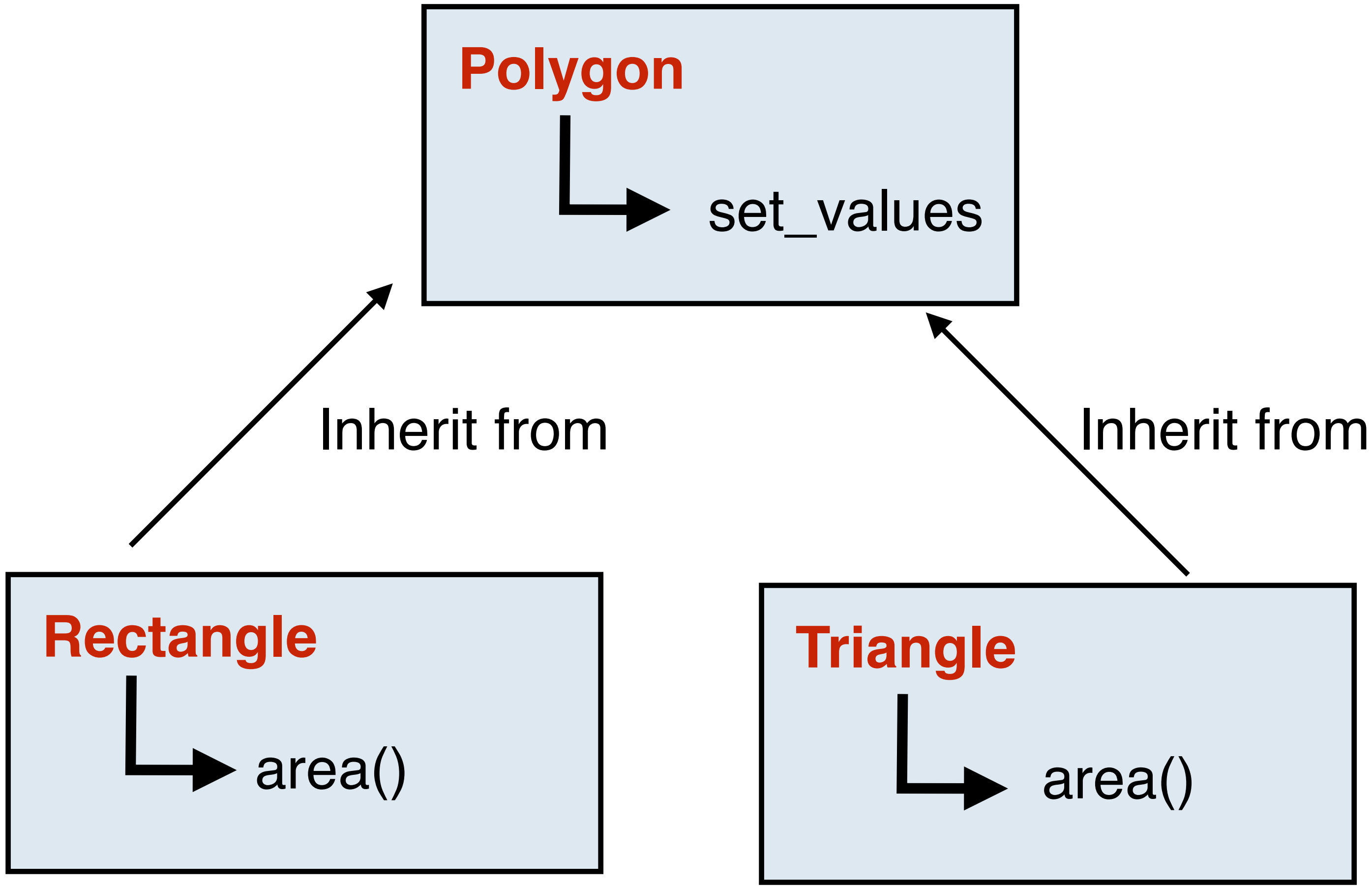
- The idea is the heritage. What a parent can do, their child can do it too.

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;
}
```

Tarball: inheritance.cpp



Polygon

set_values

Inherit from

Inherit from

Rectangle

area()

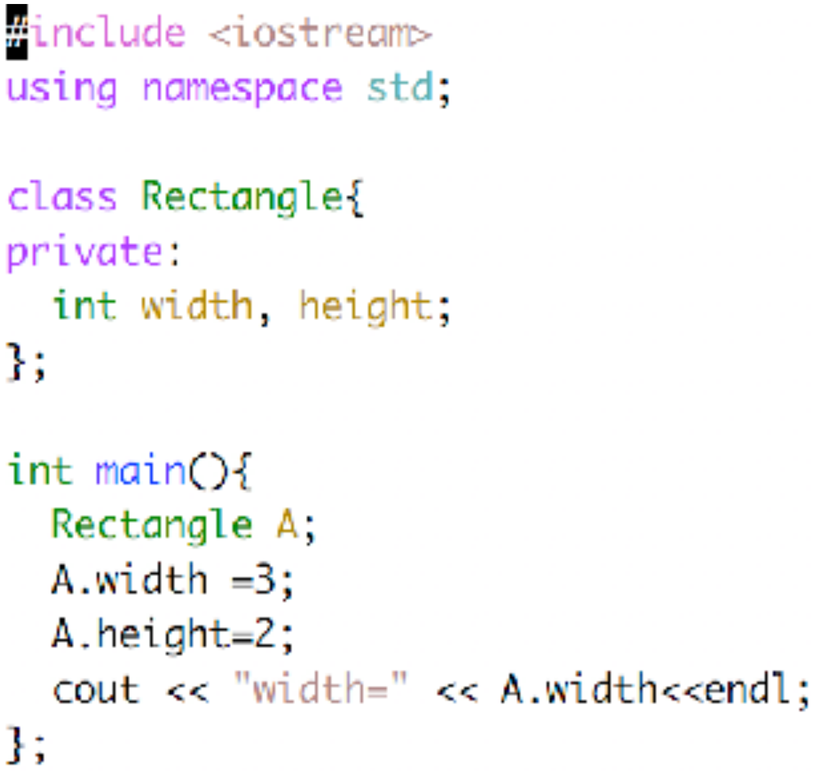
Triangle

area()

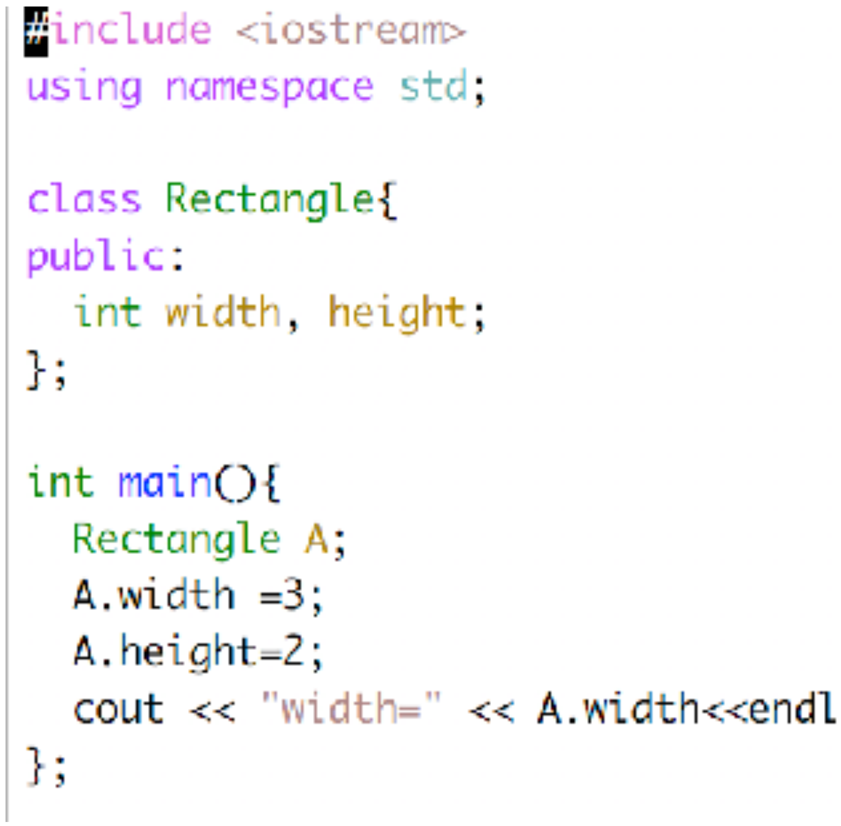
Both Rectangle and Triangle can call set_values

Visibility of attribute/function

private	protected	public
Only accessible from other instance of the same class Accessible from friends DEFAULT	Accessible from other instance of the same class Accessible from friends Accessible from instance of the derived/child class	Accessible from everywhere where the object is visible READ and WRITE!



```
#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;
};
```



```
#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
};
```

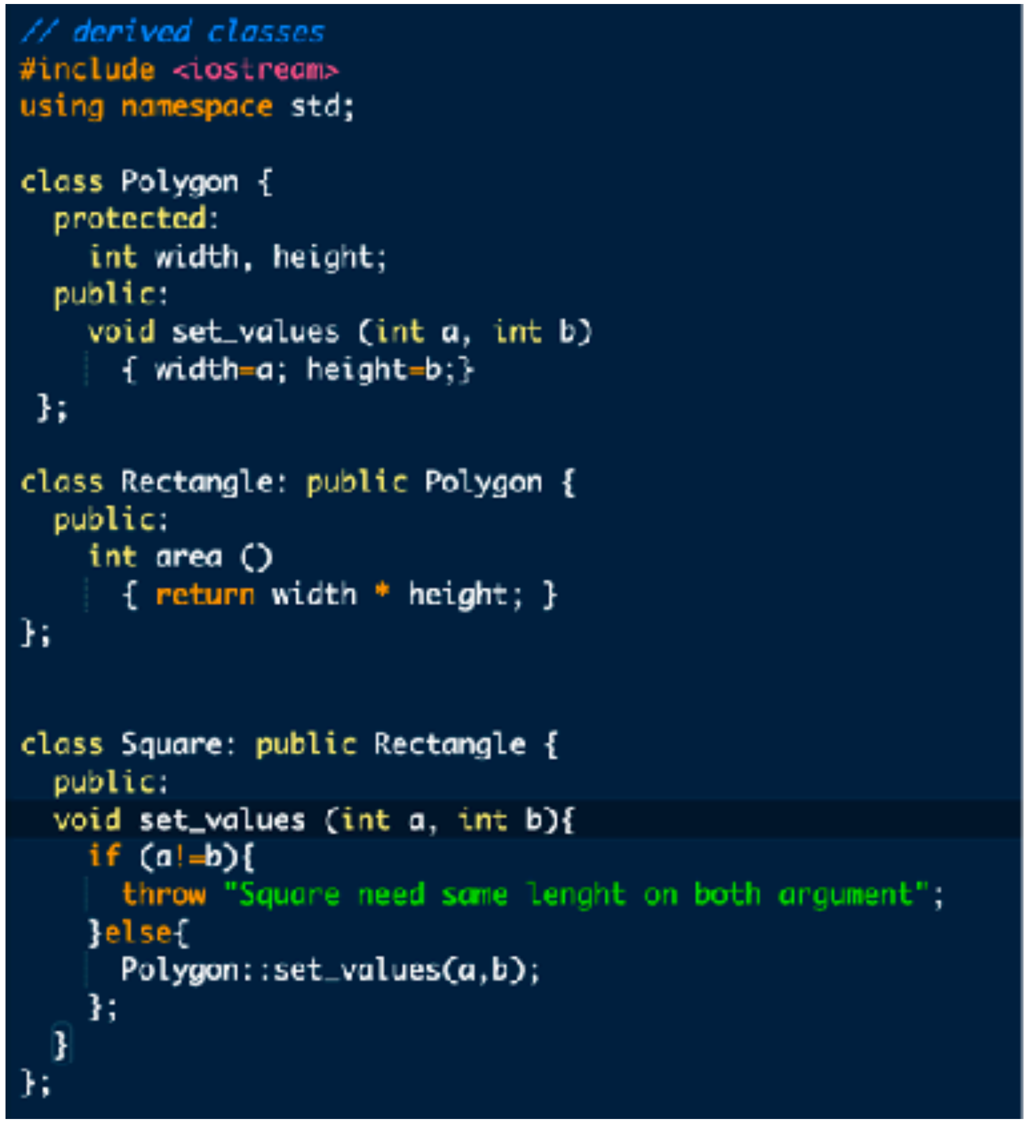
Inheritance

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

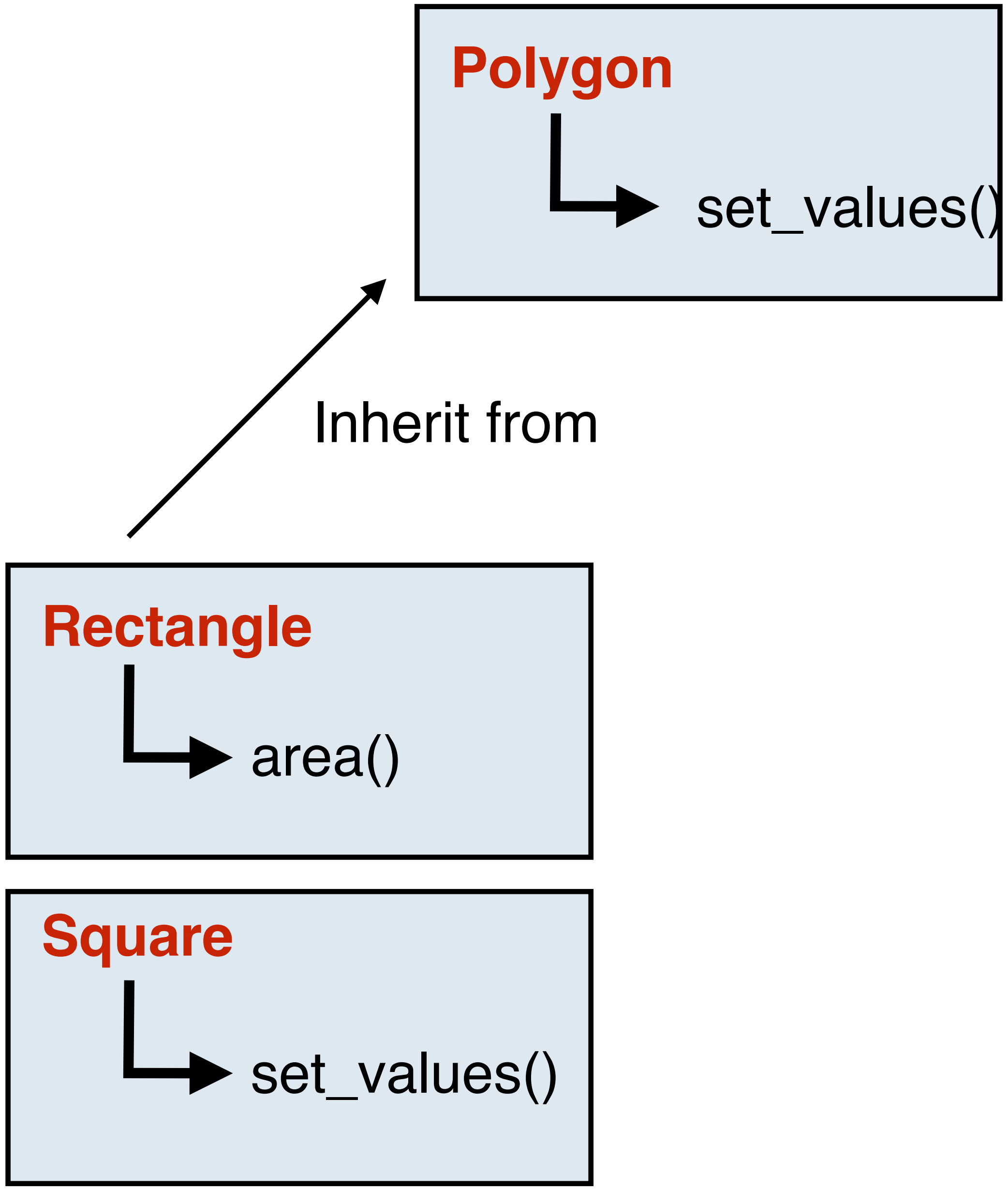
- The idea is the heritage. What a parent can do, their child can do it too.

http://tpcg.io/bKCfmxxQ

Tarball: inheritance2.cpp



```
// 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);
    };
  }
};
```



Polygon

set_values()

Inherit from

Rectangle

area()

Square

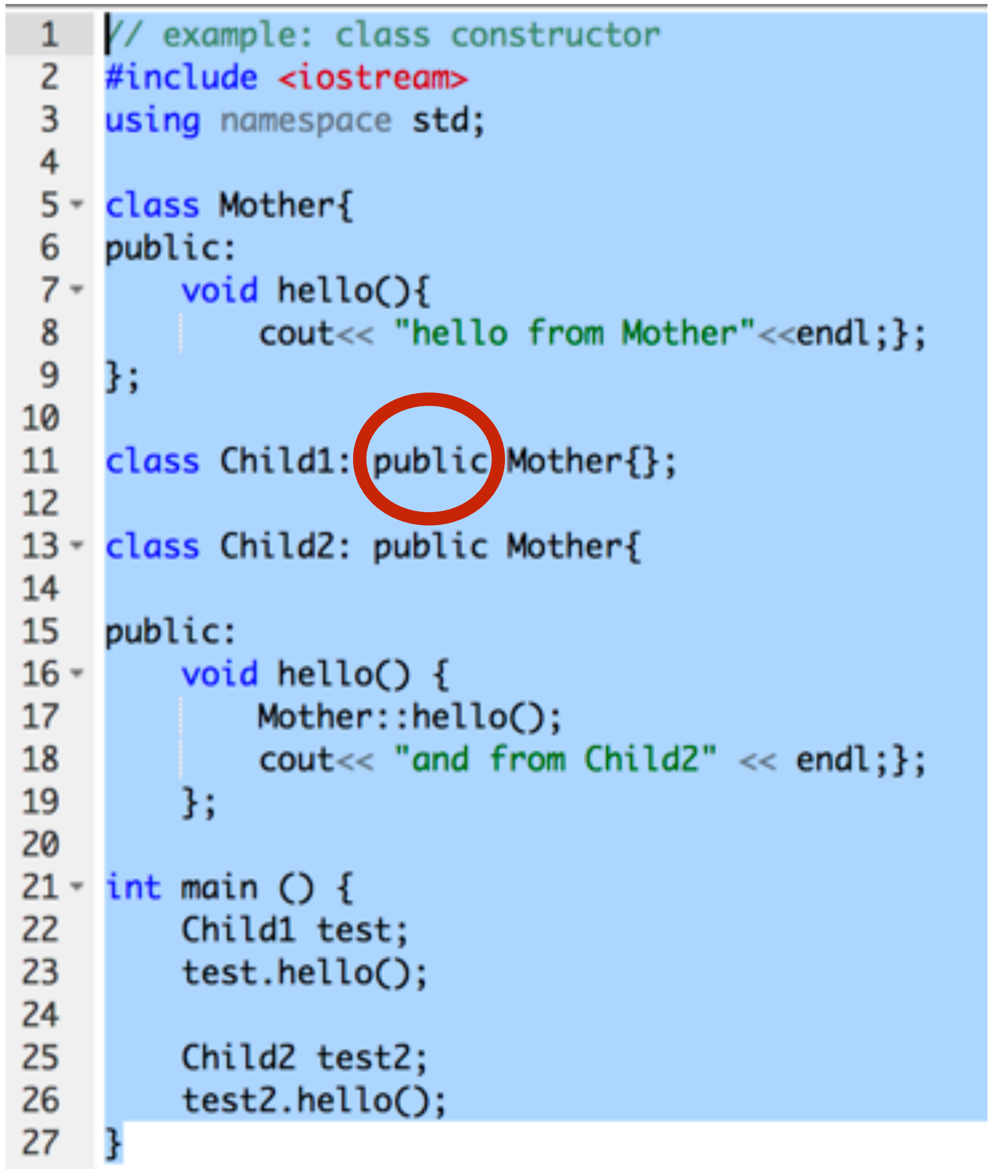
set_values()

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.

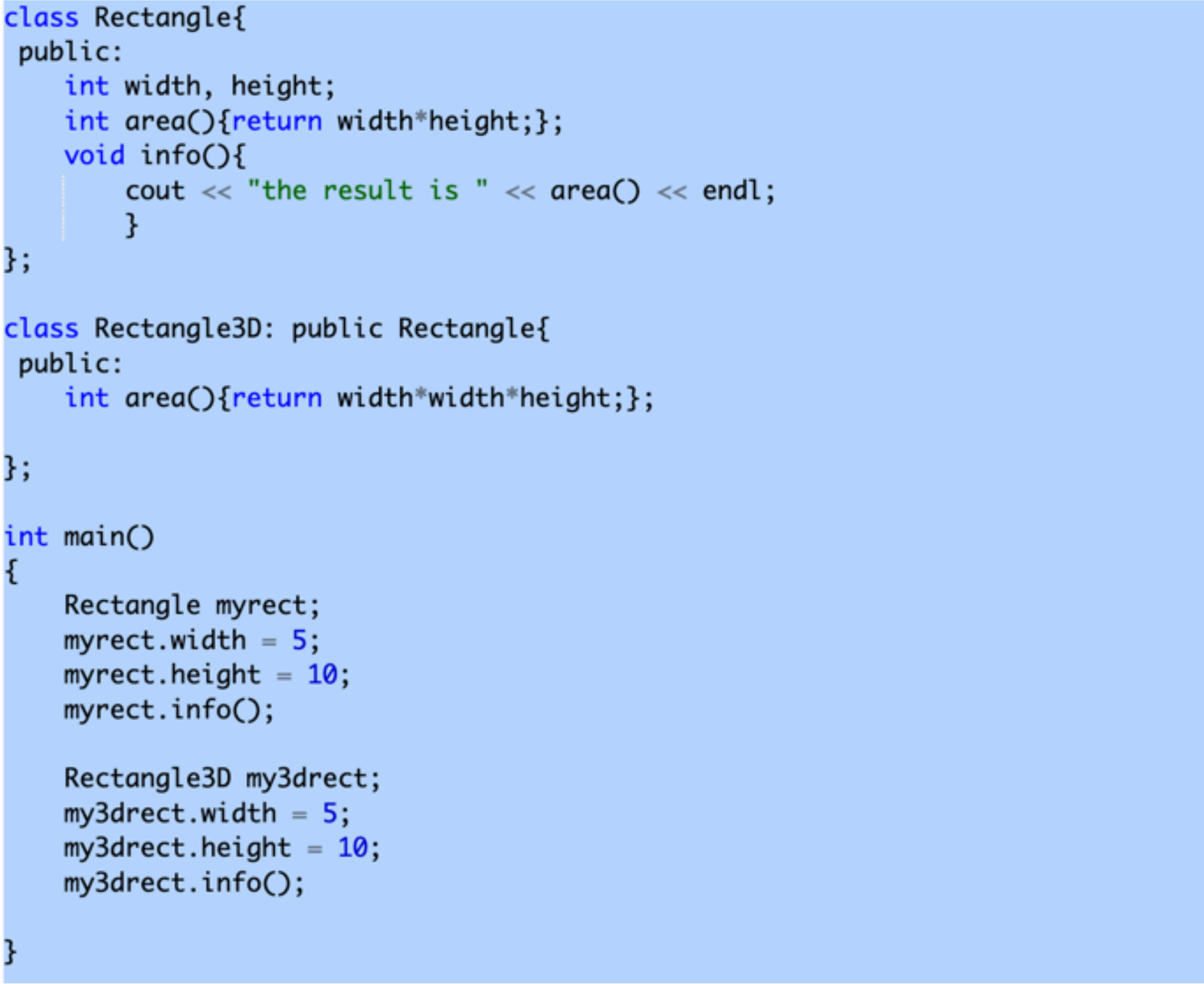
cpp.sh/72itc



```
// 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();
}
```

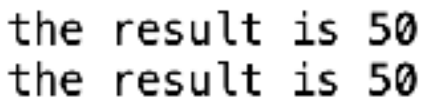
- “public” tells the maximum level of visibility of the attribute coming from the base class
  - Rare case when not set on public
- Private argument are not passed to the child (but they still exits!)
- Constructor/Destructor are **not** passed to the child
- Assignment operator (operator =) are **not**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

Virtual function



```
class Rectangle{
 public:
    int width, height;
    int area(){return width*height;};
    void info(){
        cout << "the result is " << area() << endl;
        }
};

class Rectangle3D: public Rectangle{
 public:
    int area(){return width*width*height;};

};

int main()
{
    Rectangle myrect;
    myrect.width = 5;
    myrect.height = 10;
    myrect.info();

    Rectangle3D my3drect;
    my3drect.width = 5;
    my3drect.height = 10;
    my3drect.info();

}
```

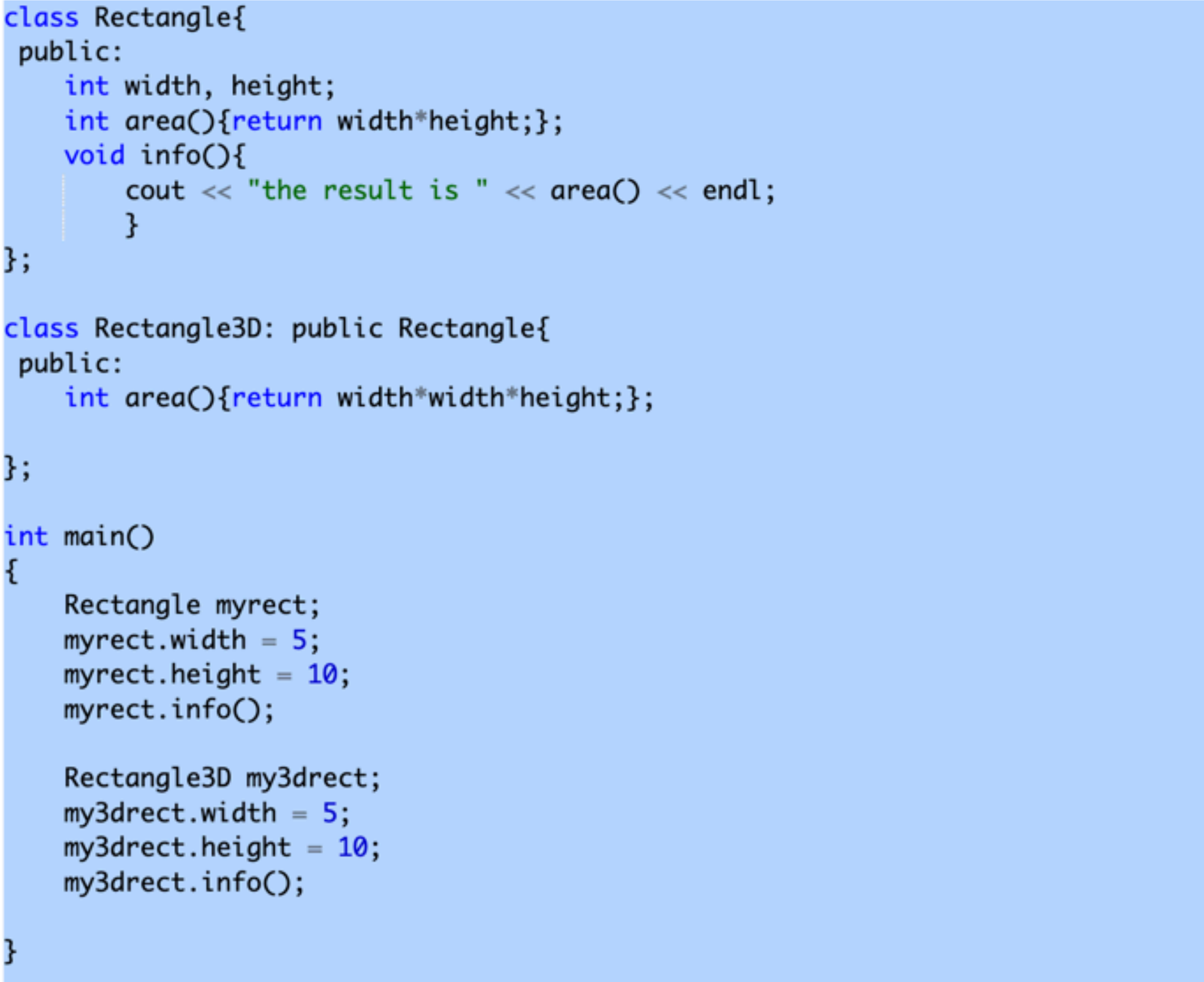
- Returns



```
the result is 50
the result is 50
```

cpp.sh/5wjno Tarball: virtual_issue.cpp

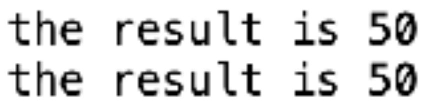
Virtual function



```
class Rectangle{
 public:
    int width, height;
    int area(){return width*height;};
    void info(){
        cout << "the result is " << area() << endl;
        }
};

class Rectangle3D: public Rectangle{
 public:
    int area(){return width*width*height;};

};

int main()
{
    Rectangle myrect;
    myrect.width = 5;
    myrect.height = 10;
    myrect.info();

    Rectangle3D my3drect;
    my3drect.width = 5;
    my3drect.height = 10;
    my3drect.info();

}
```

cpp.sh/5wjno Tarball: virtual_issue.cpp

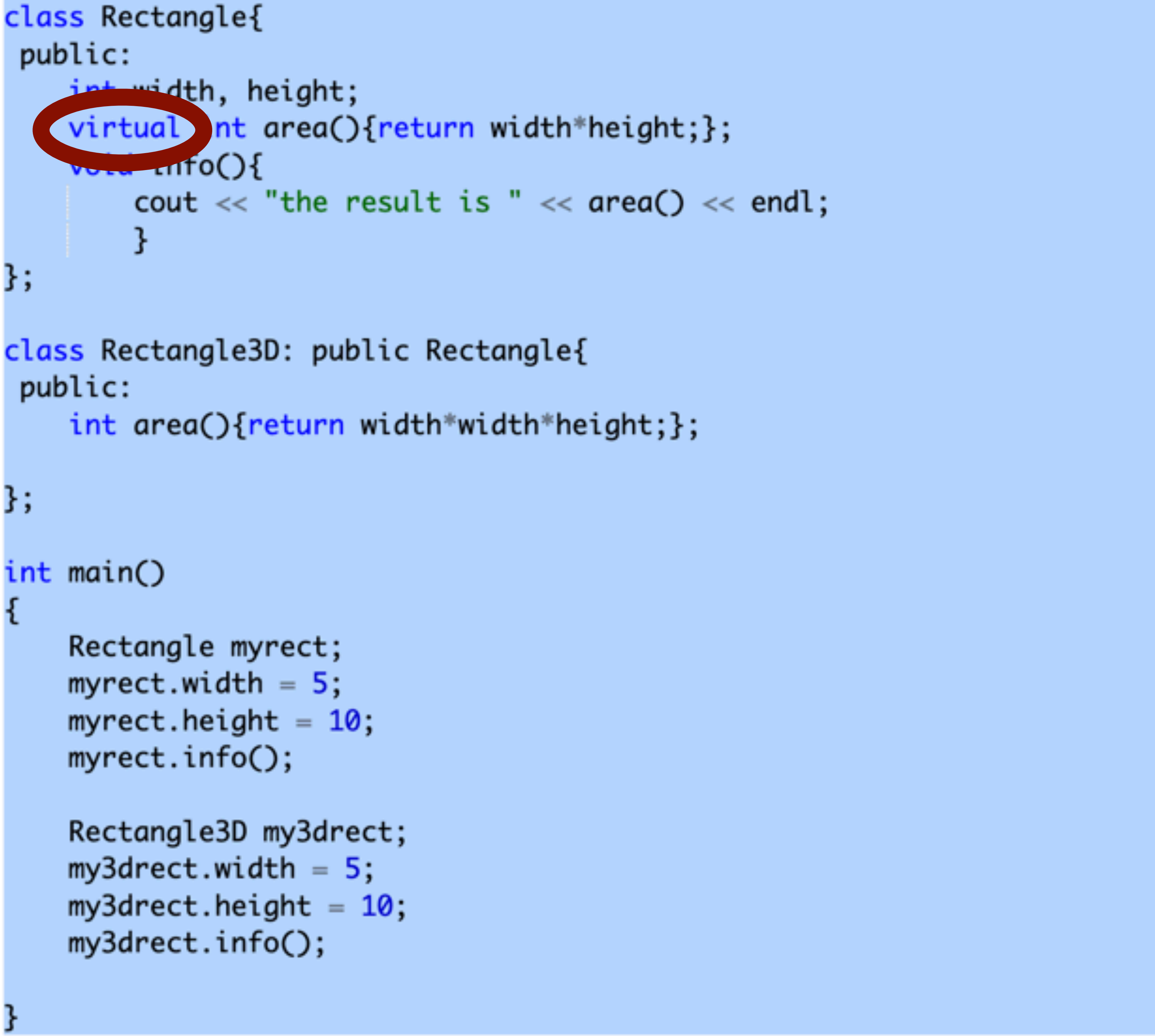
- Returns



```
the result is 50
the result is 50
```

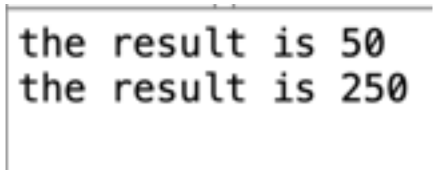
- The area() call in info link at compile time to the one of Rectangle

Virtual function



```
class Rectangle{
 public:
    int width, height;
    virtual int area(){return width*height;};
    void info(){
        cout << "the result is " << area() << endl;
        }
};

class Rectangle3D: public Rectangle{
 public:
    int area(){return width*width*height;};

};

int main()
{
    Rectangle myrect;
    myrect.width = 5;
    myrect.height = 10;
    myrect.info();

    Rectangle3D my3drect;
    my3drect.width = 5;
    my3drect.height = 10;
    my3drect.info();

}
```

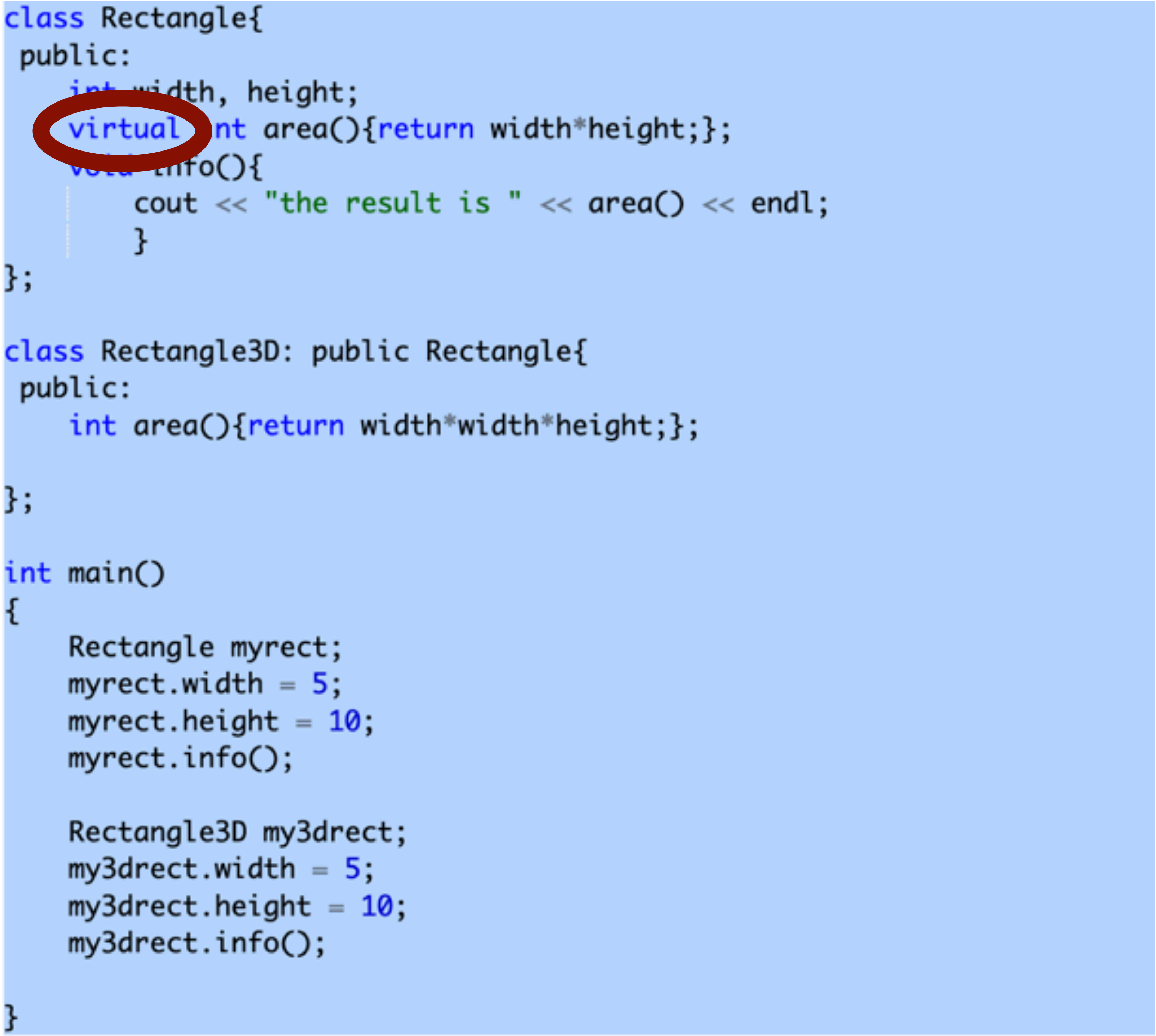
- Returns



```
the result is 50
the result is 250
```

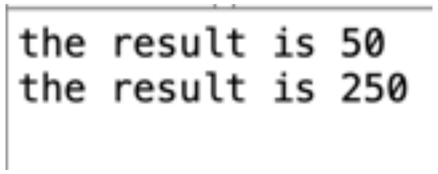
cpp.sh/8haqwr   Tarball: virtual.cpp

Virtual function



```
class Rectangle{
 public:
    int width, height;
    virtual int area(){return width*height;};
    void info(){
        cout << "the result is " << area() << endl;
        }
};

class Rectangle3D: public Rectangle{
 public:
    int area(){return width*width*height;};

};

int main()
{
    Rectangle myrect;
    myrect.width = 5;
    myrect.height = 10;
    myrect.info();

    Rectangle3D my3drect;
    my3drect.width = 5;
    my3drect.height = 10;
    my3drect.info();

}
```

- Returns



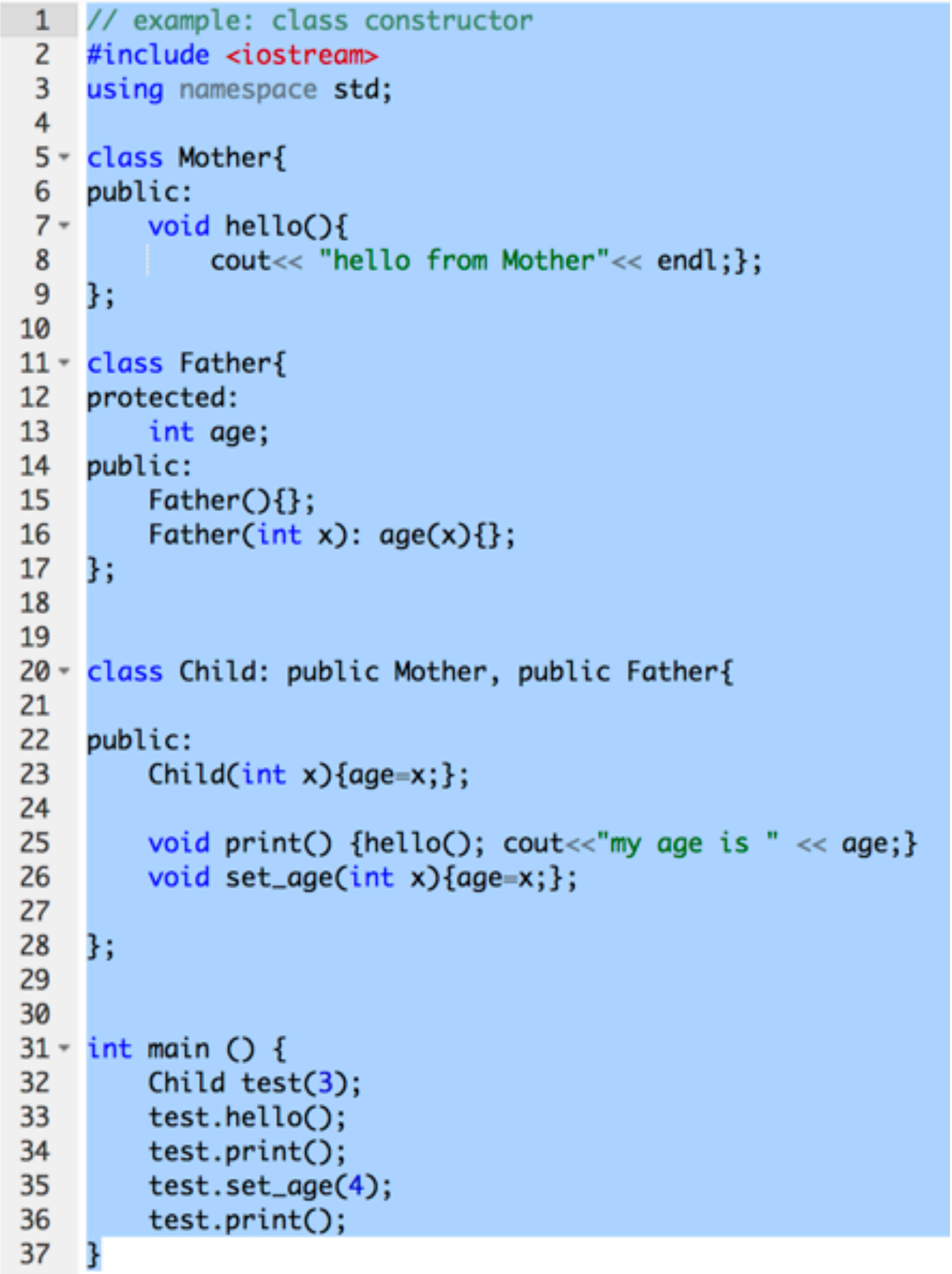
```
the result is 50
the result is 250
```

- The area() call in info link at **RUN** time to the one of Rectangle3D

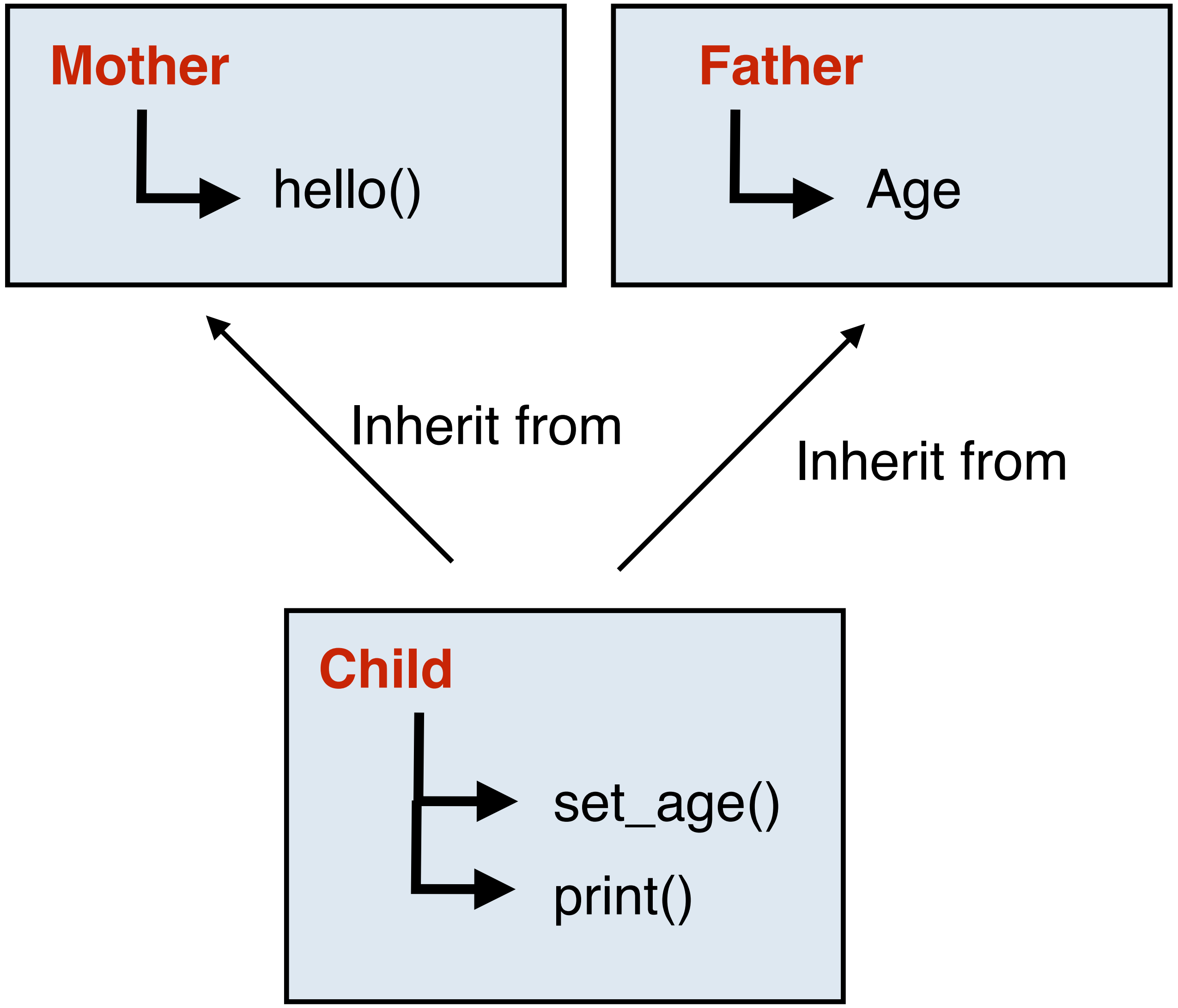
cpp.sh/8haqwr   Tarball: virtual.cpp

Multi-inheritance

cpp.sh/3nhb



```cpp
// example: class constructor
#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();
}
```



Mother

hello()

Father

Age

Inherit from

Inherit from

Child

set_age()

print()

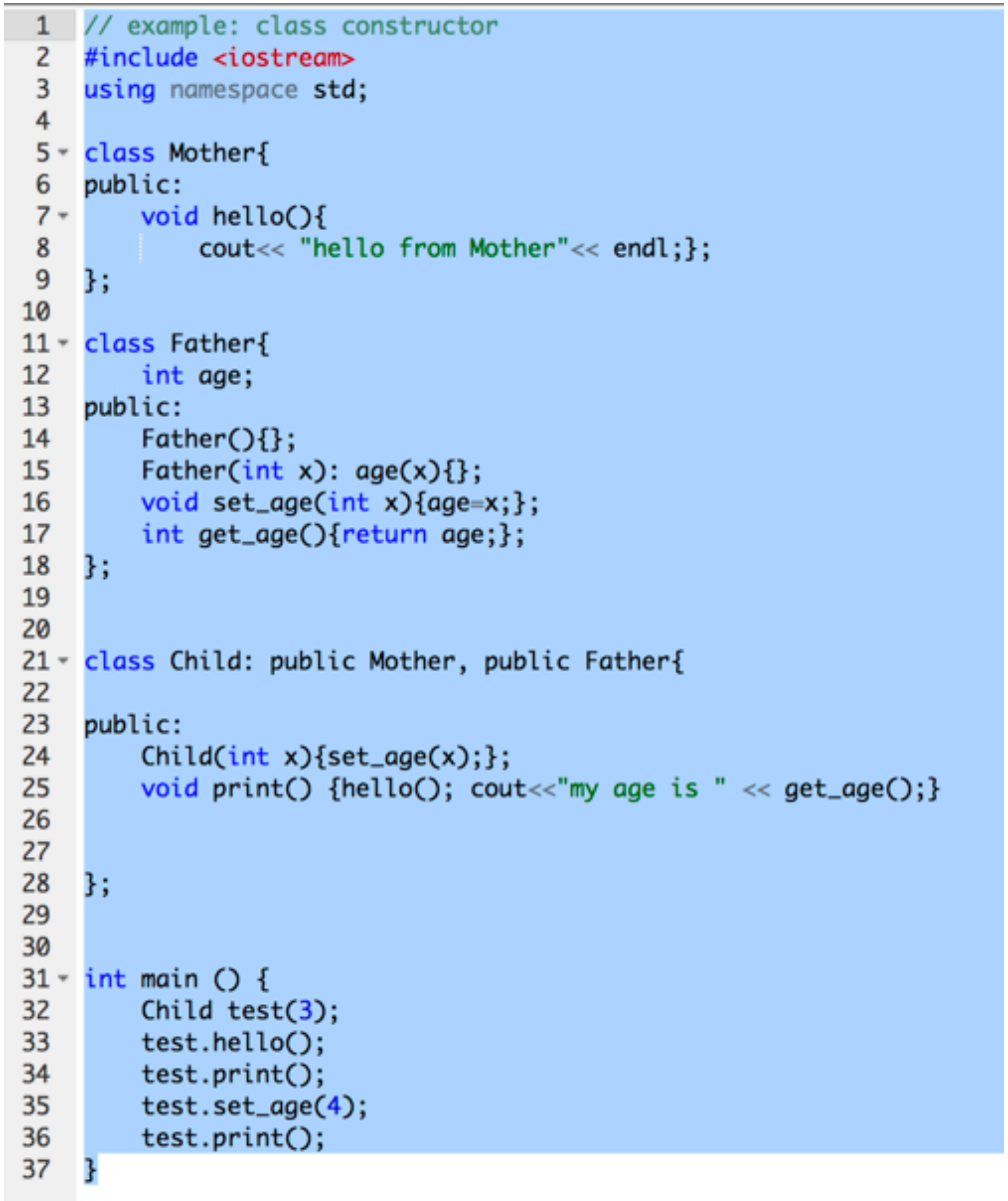
Can still call hello()

Can access to age (protected)

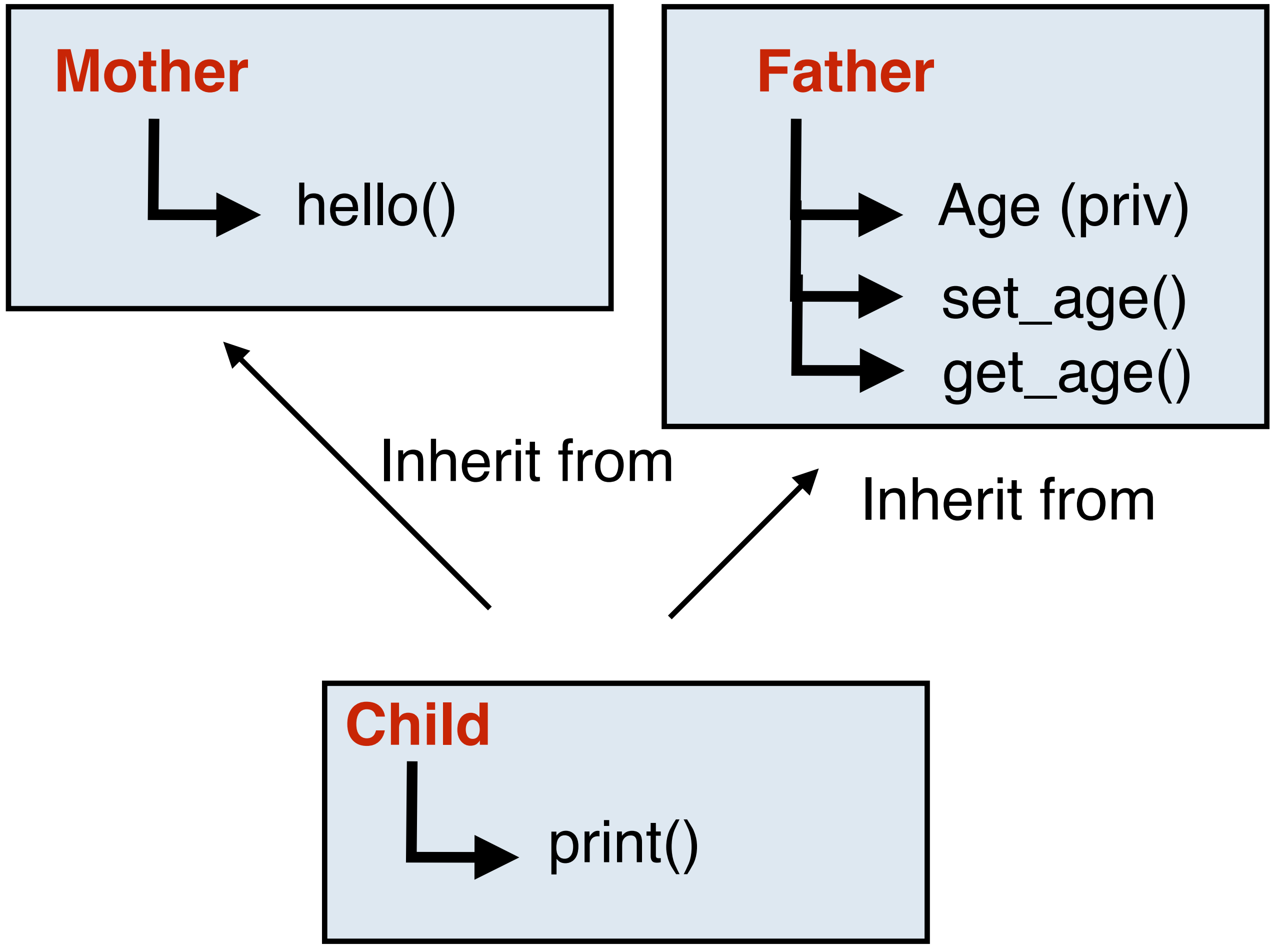
Tarball: multi_inheritence.cpp

Multi-inheritance

cpp.sh/8vev Tarball: multi_inheritance2.cpp



```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();
}
```



Can call hello()

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

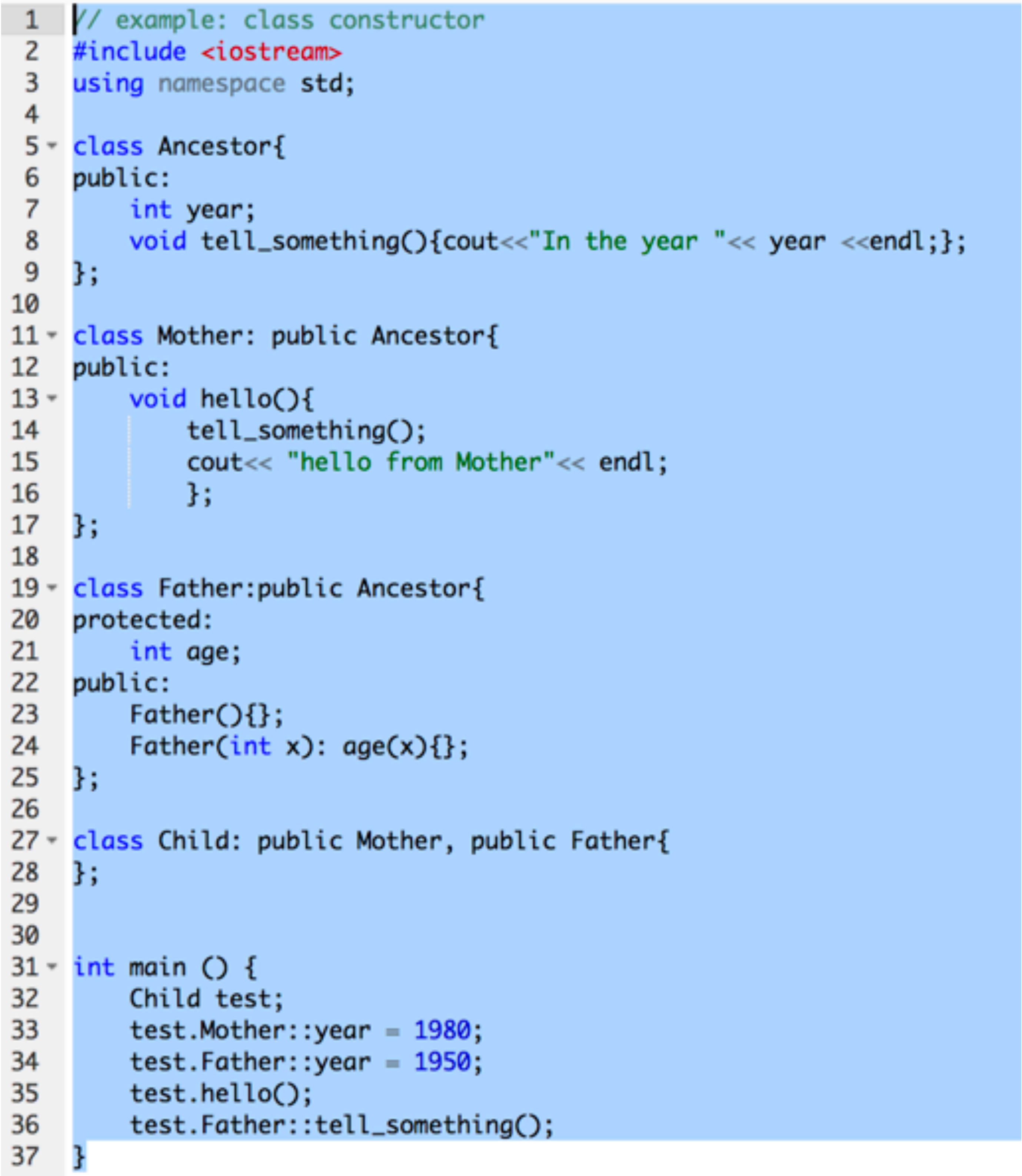
Exercise II

- Let's code a simple game
  - Let's code a class projectile
    - With parameter
      - Speed (vx, vy)
    - Two function
      - Impact_position (2*vx*vy/g)
      - Impact (return a number for damage)
- Define two subclass
  - That defines two type of projectile where
    - One that has reduced gravity
    - One that makes damage proportional to the distance

cpp.sh/2ul6x4

Diamond Diagram

cpp.sh/4inoj

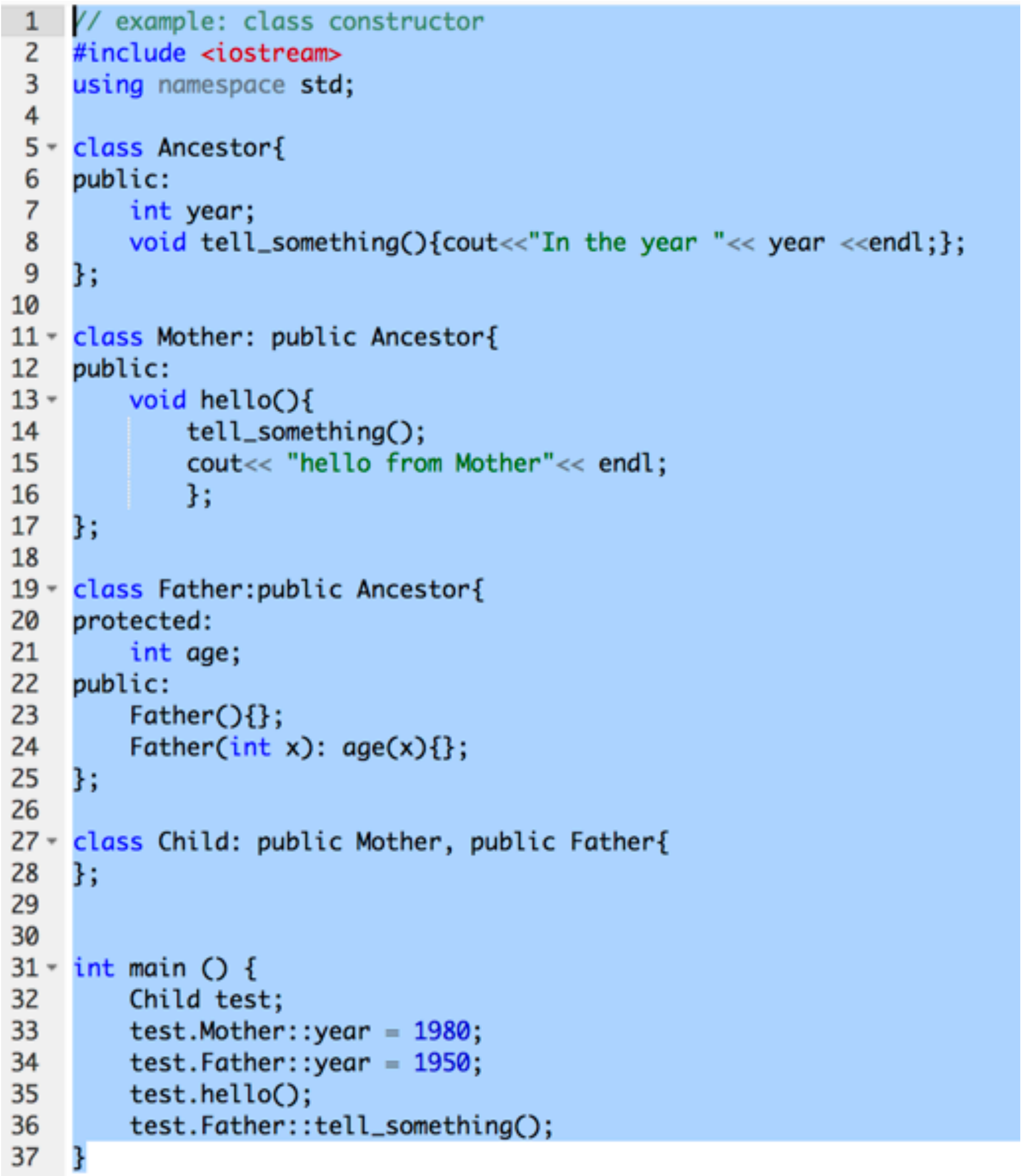


```
// example: class constructor
#include <iostream>
using namespace std;

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();
}
```

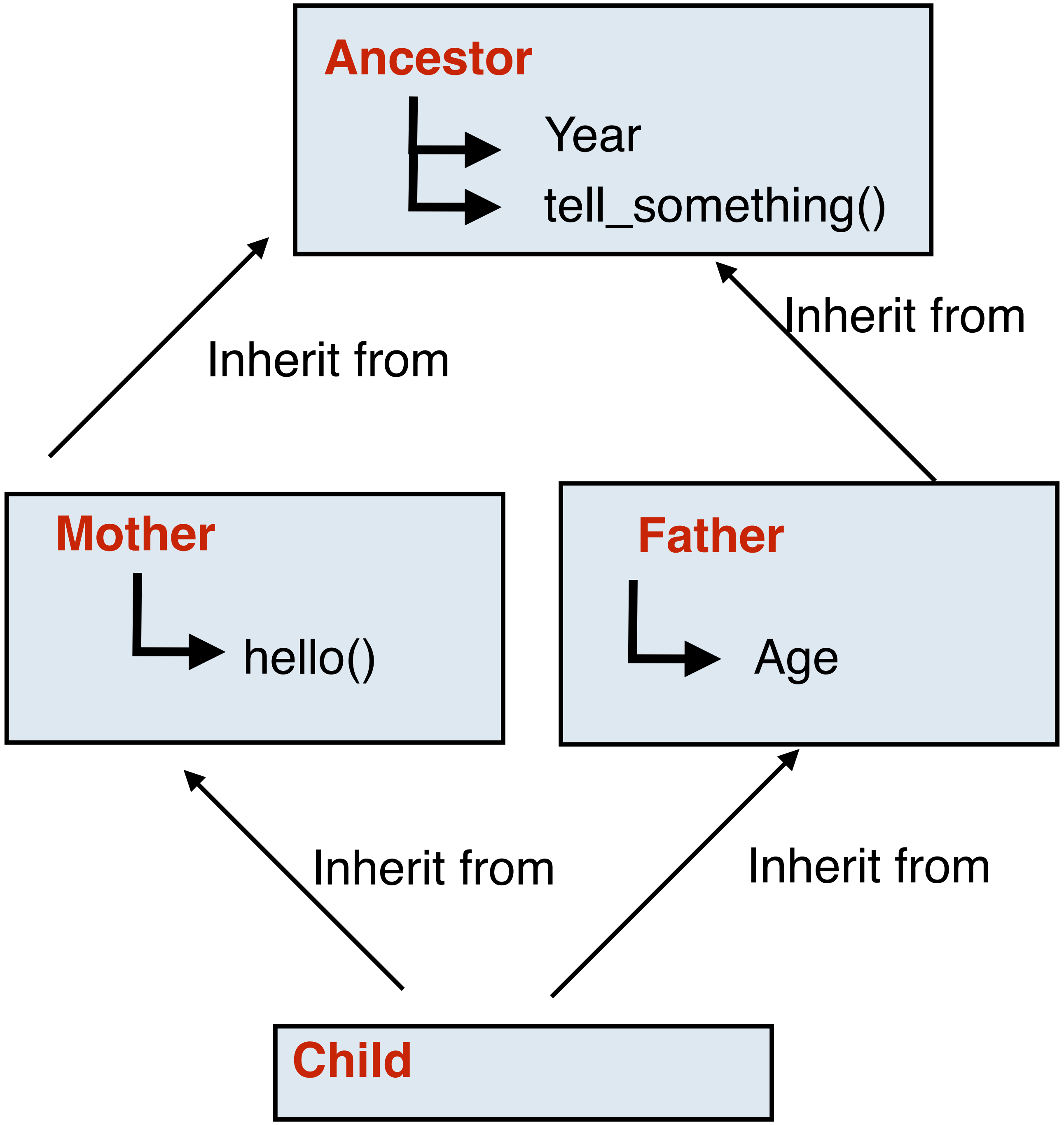
- Two copy of the Ancestor class
  - ➡ test.Mother::year
  - ➡ test.Father::year
- You can use virtual inheritance to have a single copy
  - ➡ “public virtual Ancestor”
- Consider as bad design in C++
  - ➡ Because C++ sucks on those!

Diamond Diagram

cpp.sh/4inoj   Tarball: diamond.cpp



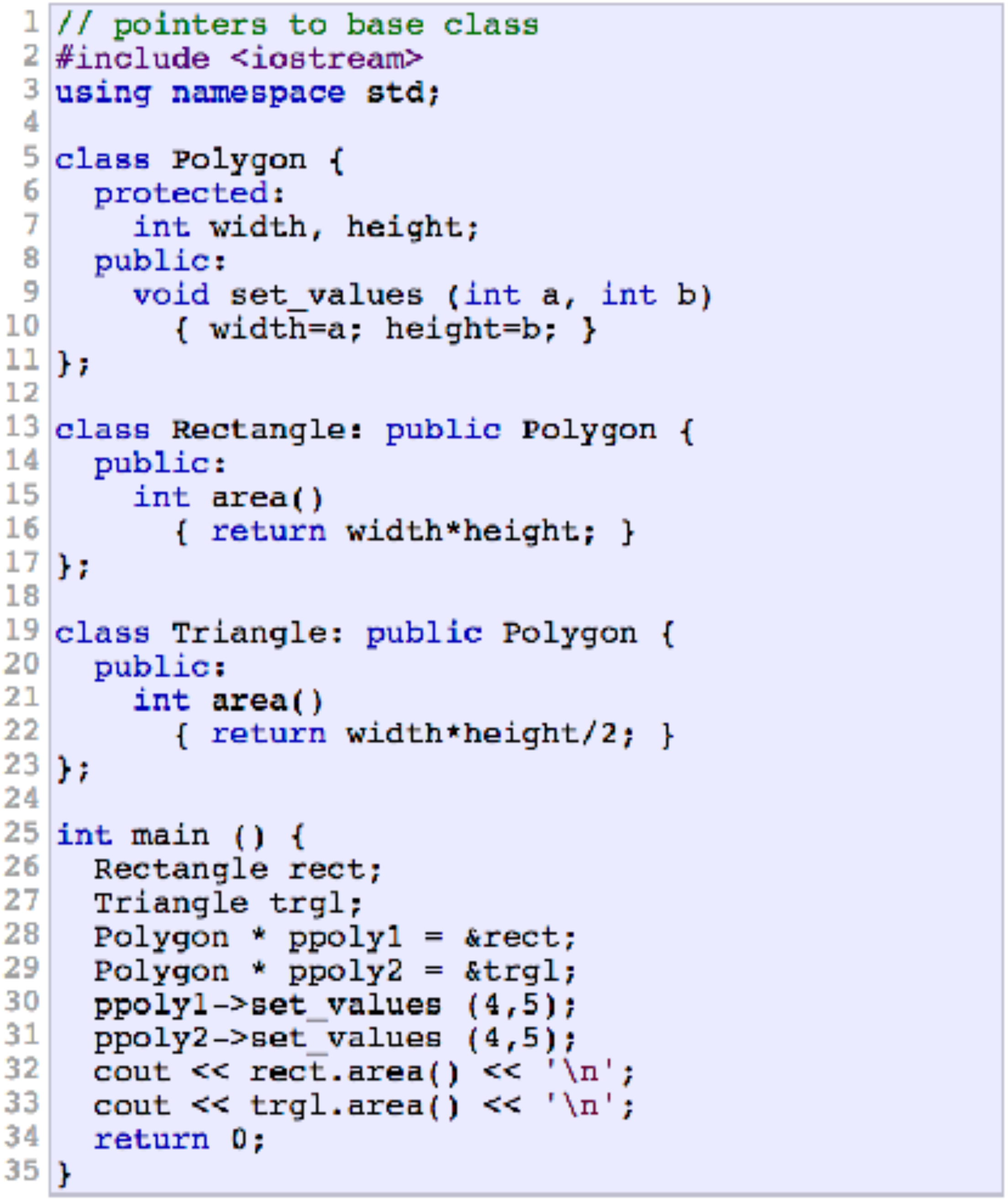
```cpp
// example: class constructor
#include <iostream>
using namespace std;

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();
}
```



Polymorphism

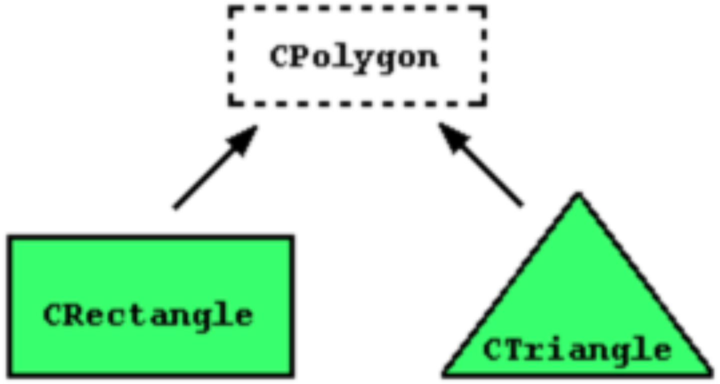
a pointer to a derived class is type-compatible with a pointer to its base class

cpp.sh/3tz



```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;
}
```

Tarball: polymorphic.cpp

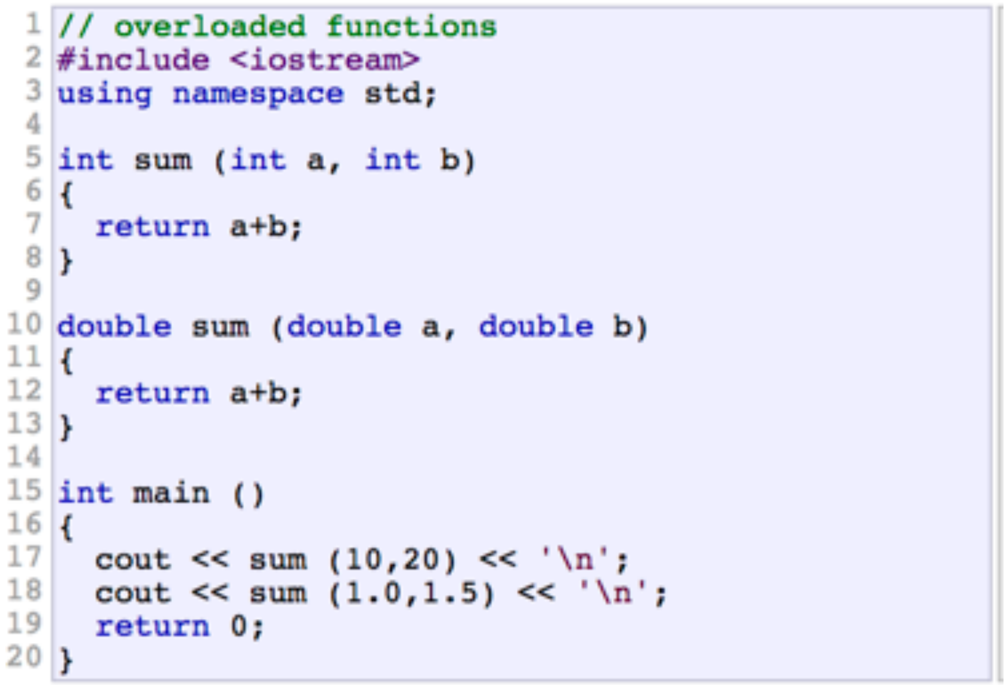


- We can use a pointer of the class CPolygon (CPolygon*) with object from his derived class
- Note that from pointer you can access attribute/member function with ->
- Carefull which function you access with polymorphism

Template

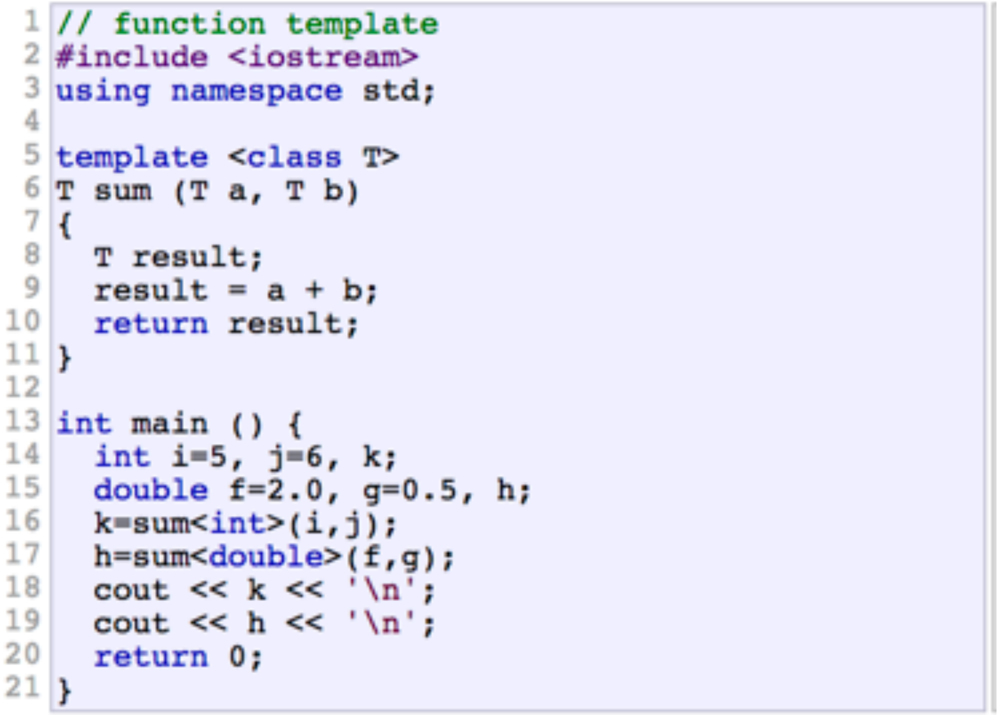
Template = define functions class with generic type

- Repeat yourself is bad but often you have to have the exact same definition but for different type
  - ➡ Polymorphism can be use (use pointer of base class)
  - ➡ Template is a more general solution (no need of pointer)



```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.sh/4jay

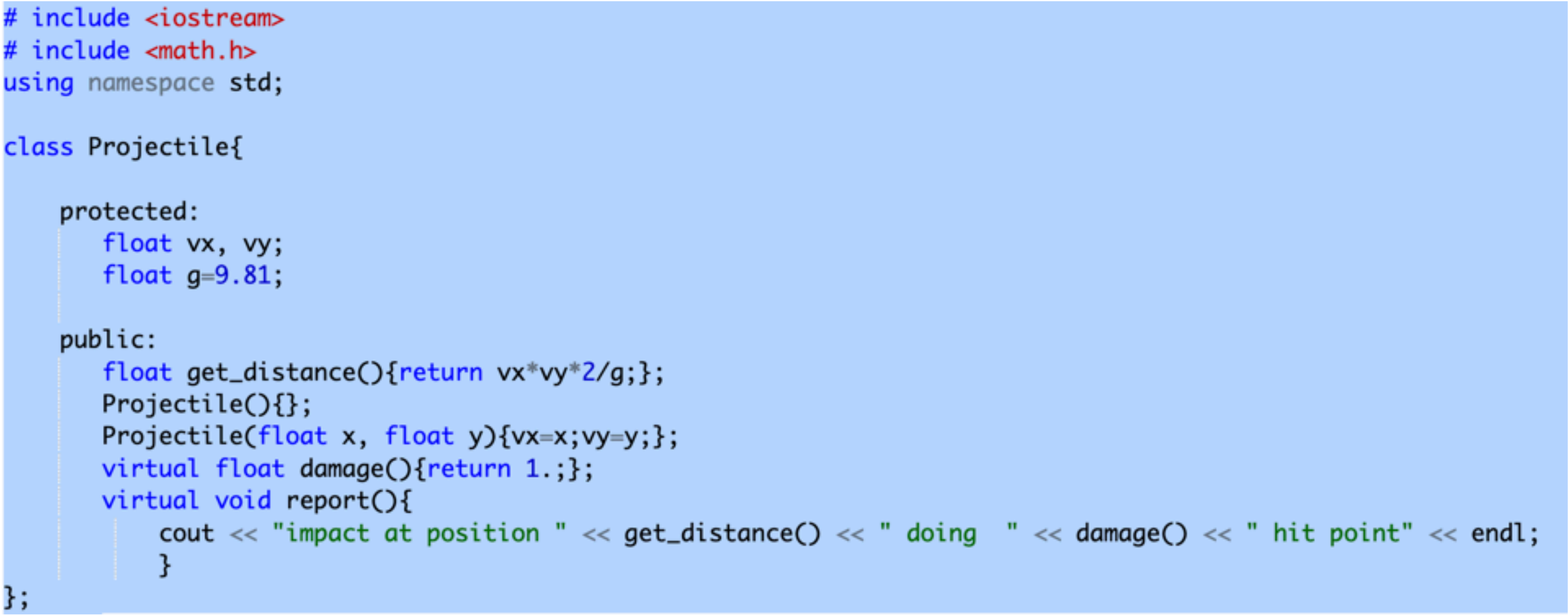


```cpp
// function template
#include <iostream>
using namespace std;

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;
}
```

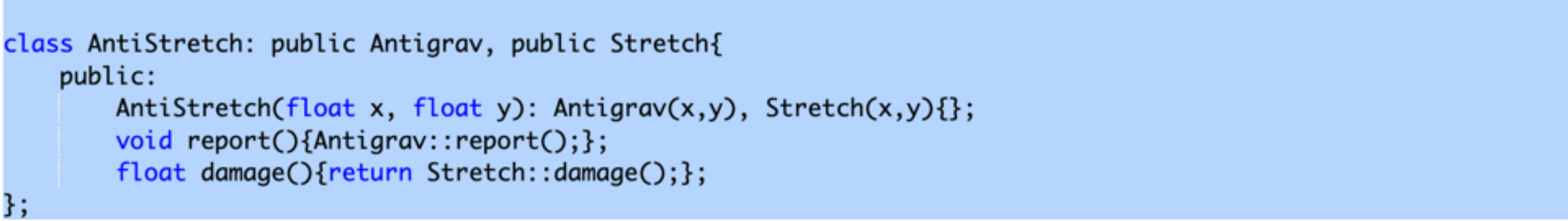
Exercise III

- Define two subclass
  - That defines two type of projectile where
    - One that has reduced gravity
    - One that makes damage proportional to the distance
- Define a class that inherit from both class
  - That has reduced gravity and damage proportional to the distance

cpp.sh/2ayss



```
# include <iostream>
# include <math.h>
using namespace std;

class Projectile{

    protected:
       float vx, vy;
       float g=9.81;

    public:
       float get_distance(){return vx*vy*2/g;};
       Projectile(){};
       Projectile(float x, float y){vx=x;vy=y;};
       virtual float damage(){return 1.;};
       virtual void report(){
           cout << "impact at position " << get_distance() << " doing  " << damage() << " hit point" << endl;
           }
};
```



```
class AntiStretch: public Antigrav, public Stretch{
    public:
        AntiStretch(float x, float y): Antigrav(x,y), Stretch(x,y){};
        void report(){Antigrav::report();};
        float damage(){return Stretch::damage();};
};
```

cpp.sh/2ayss

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/…