Introduction to Object -Oriented Programming with C++

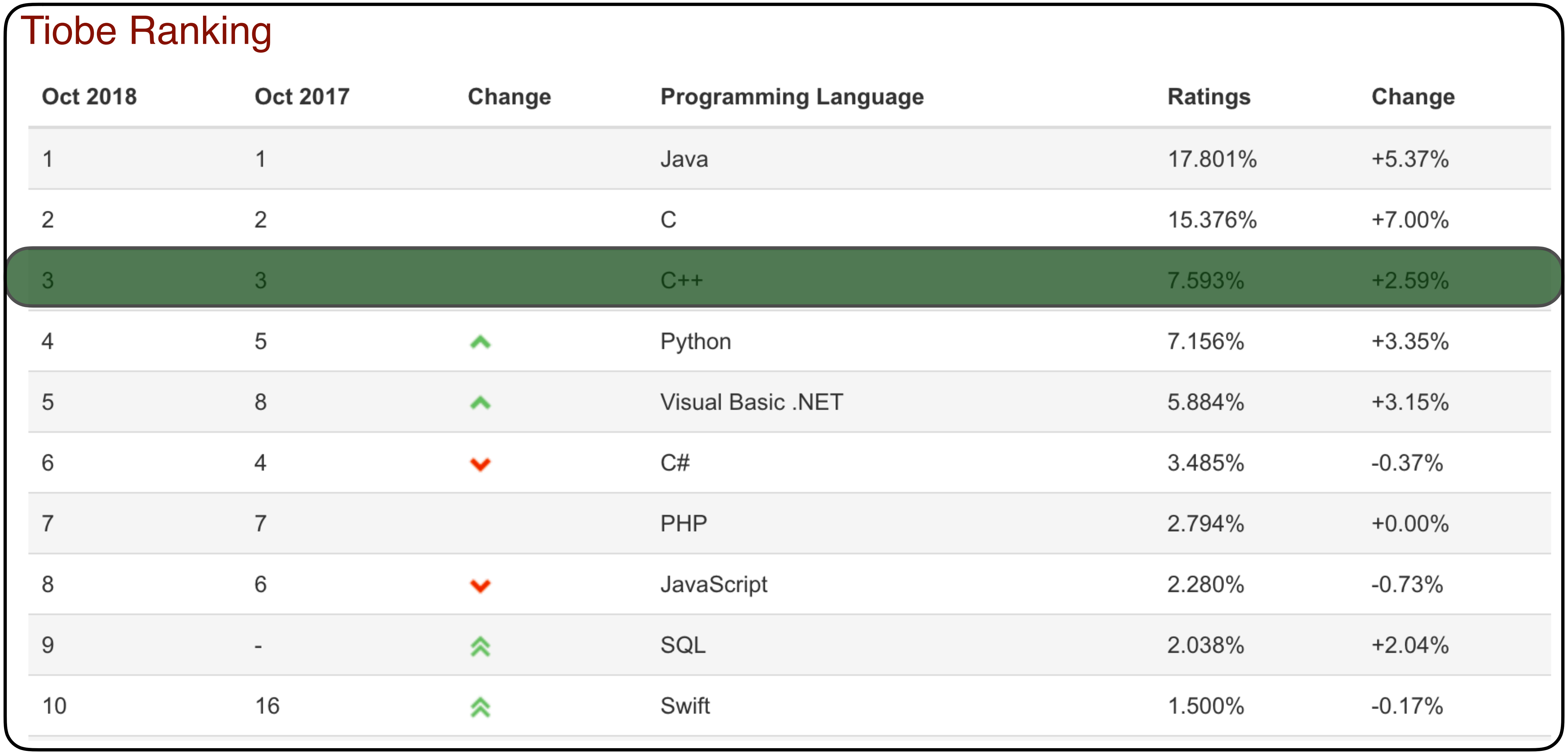
Olivier Mattelaer
UCLouvain
CP3 & CISM

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 affecting any code that uses the object

Why C++



Tiobe Ranking

Oct 2018	Oct 2017	Change	Programming Language	Ratings	Change
1	1		Java	17.801%	+5.37%
2	2		C	15.376%	+7.00%
3	3		C++	7.593%	+2.59%
4	5	▲	Python	7.156%	+3.35%
5	8	▲	Visual Basic .NET	5.884%	+3.15%
6	4	▼	C#	3.485%	-0.37%
7	7		PHP	2.794%	+0.00%
8	6	▼	JavaScript	2.280%	-0.73%
9	-	▲▲	SQL	2.038%	+2.04%
10	16	▲▲	Swift	1.500%	-0.17%

- 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

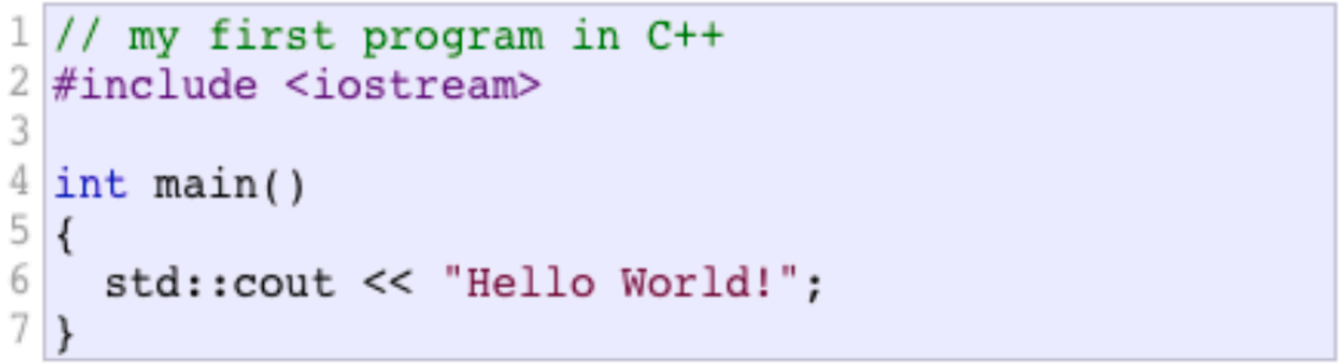
- Basic of C++
  - ➡ Presentation of concept
  - ➡ Code presentation
  - ➡ Exercise
- Introduction to Class/object in C++
  - ➡ Presentation of concept
  - ➡ Code presentation
  - ➡ Exercise
- (Multi) Inheritance
  - ➡ Presentation of concept
  - ➡ Code presentation
  - ➡ Exercise

Program of today

- Basic of C++
  - ➡ Presentation of concept
  - ➡ Code presentation
  - ➡ Exercise
- Introduction to Class/object in C++
  - ➡ Presentation of concept
  - ➡ Code presentation
  - ➡ Exercise
- / Inheritance
  - ➡ 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

- 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++

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/

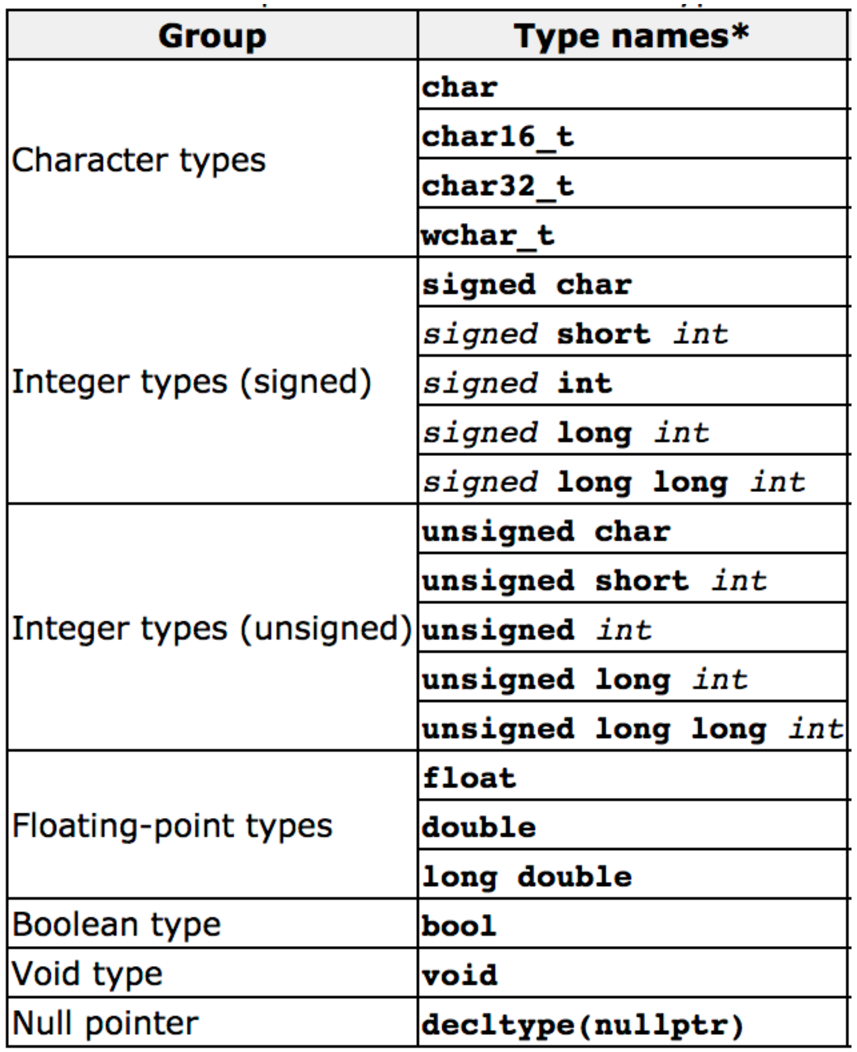
Select C++14 (bottom left)

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

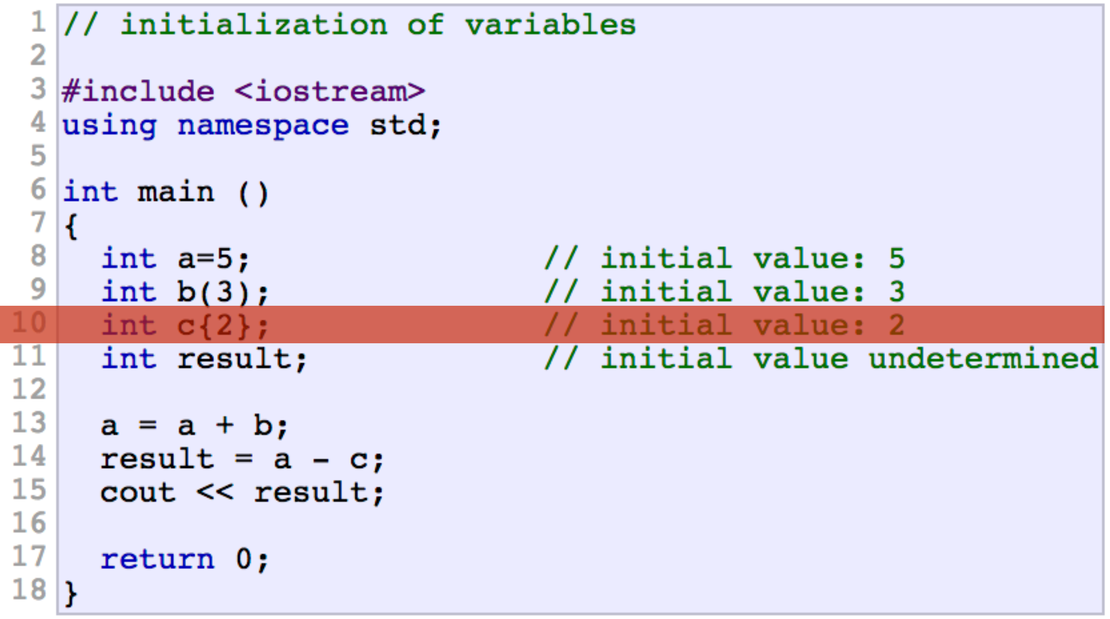
Basic of C++ : variables

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



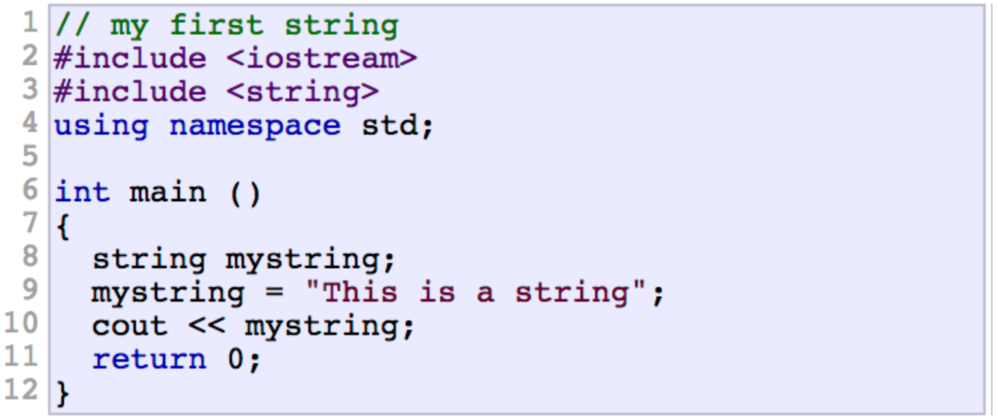
Group	Type names*
Character types	**char**
	char16_t
	char32_t
	wchar_t
Integer types (signed)	**signed char**
	signed **short** *int*
	signed **int**
	signed **long** *int*
	signed **long long** *int*
Integer types (unsigned)	**unsigned char**
	unsigned short *int*
	unsigned *int*
	unsigned long *int*
	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

http://cpp.sh/8yl

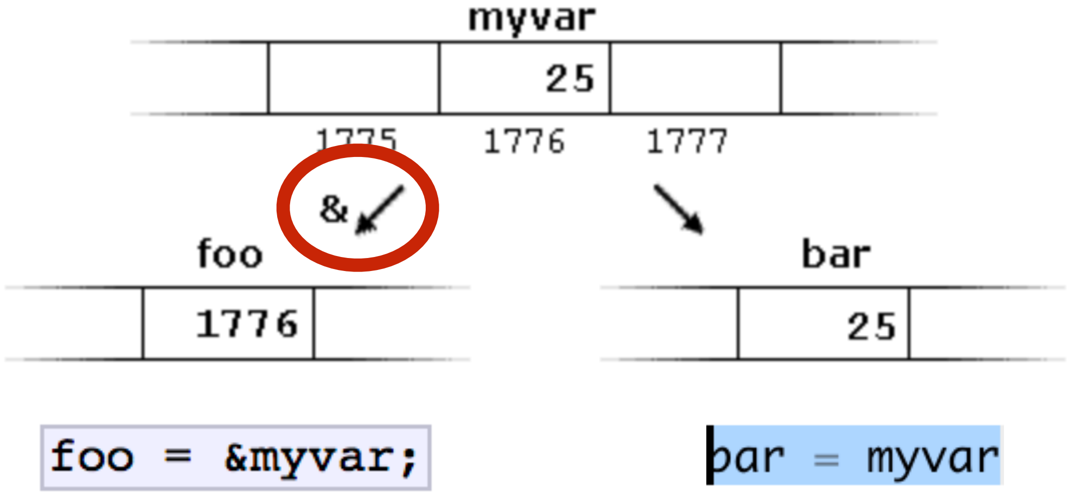


```
// my first string
#include <iostream>
#include <string>
using namespace std;

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

http://cpp.sh/7d4

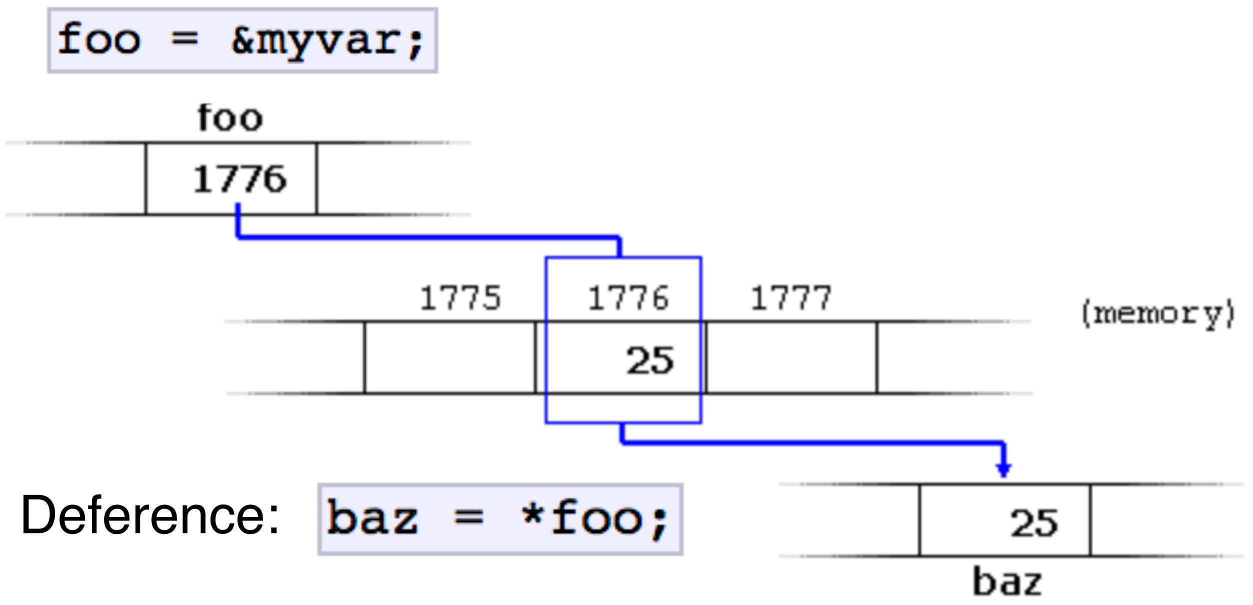
Basic of C++: pointer

Pointer = position in memory of the variable



```
foo = &myvar;
```

```
bar = myvar
```



```
foo = &myvar;
```

Deference:

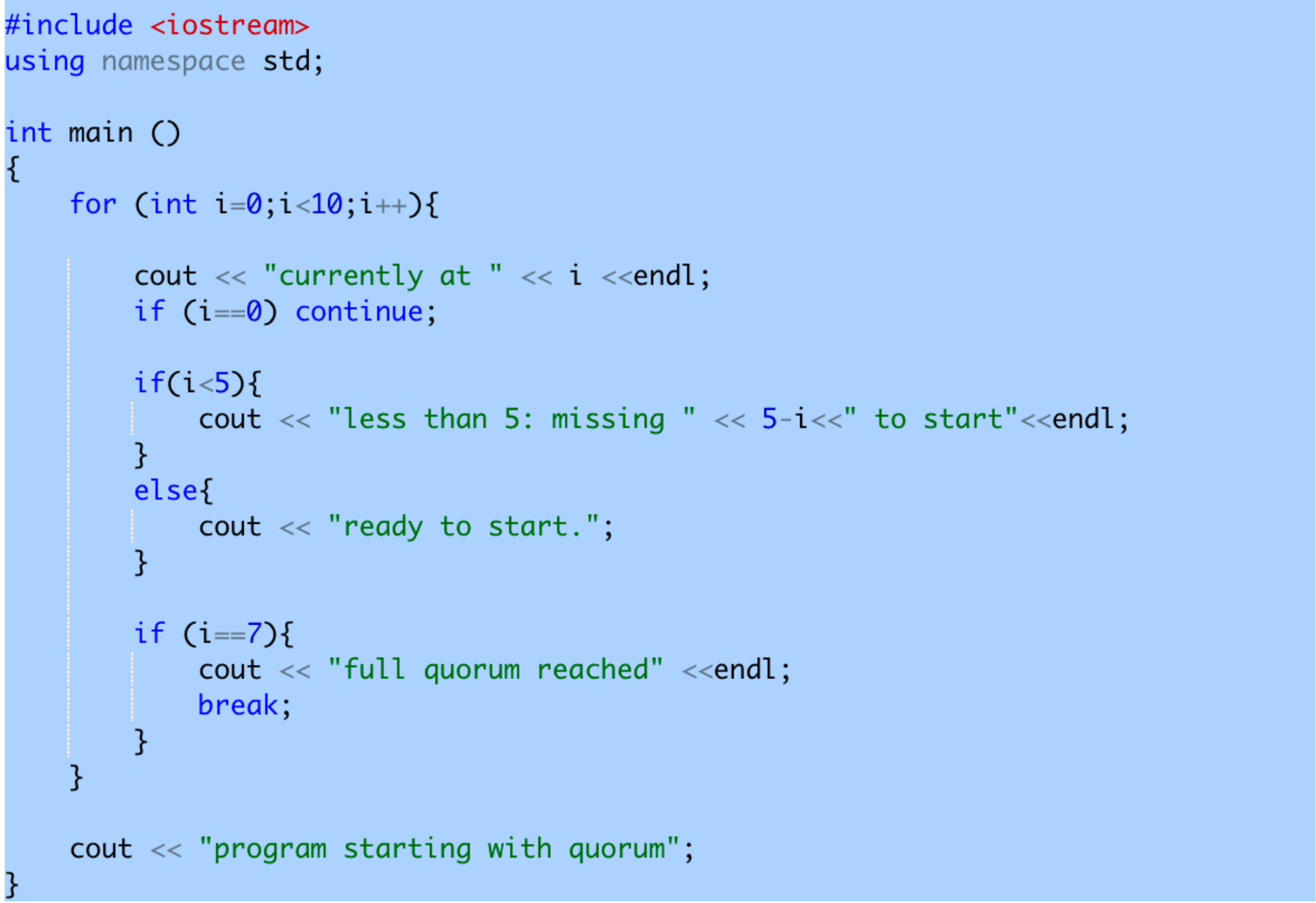
```
baz = *foo;
```

- 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++: loop/branching

- If statement
- Loop



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

int main ()
{
    for (int i=0;i<10;i++){

        cout << "currently at " << i <<endl;
        if (i==0) continue;

        if(i<5){
            cout << "less than 5: missing " << 5-i<<" to start"<<endl;
        }
        else{
            cout << "ready to start.";
        }

        if (i==7){
            cout << "full quorum reached" <<endl;
            break;
        }
    }

    cout << "program starting with quorum";
}
```

http://cpp.sh/2jy27v

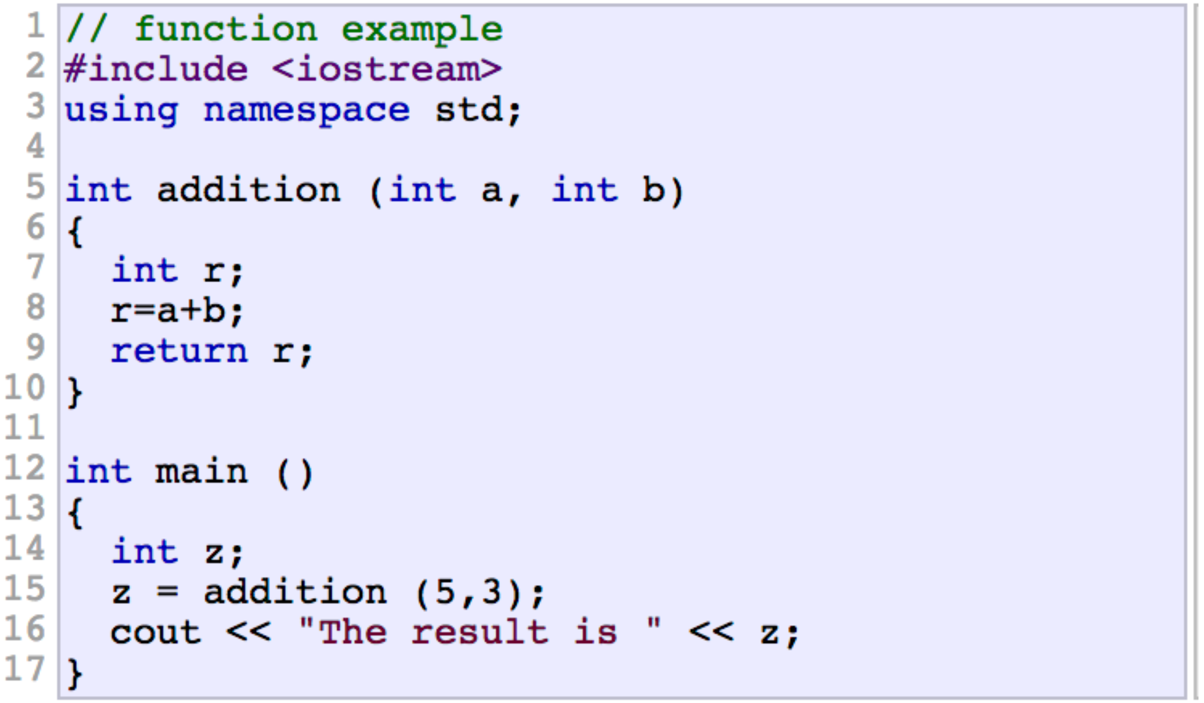
- For Loop
  - Initialisation
  - Condition
  - Update
- While loop
  - while(condition){}
- Do while loop
- Do{}while();

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

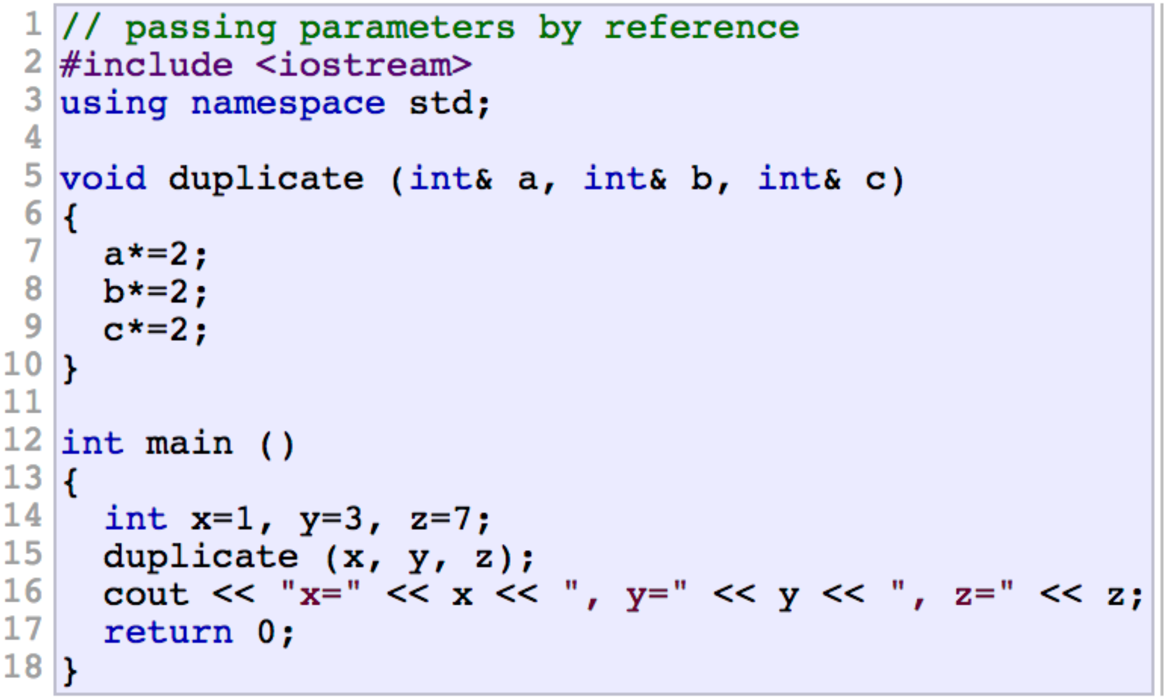


```cpp
// function example
#include <iostream>
using namespace std;

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

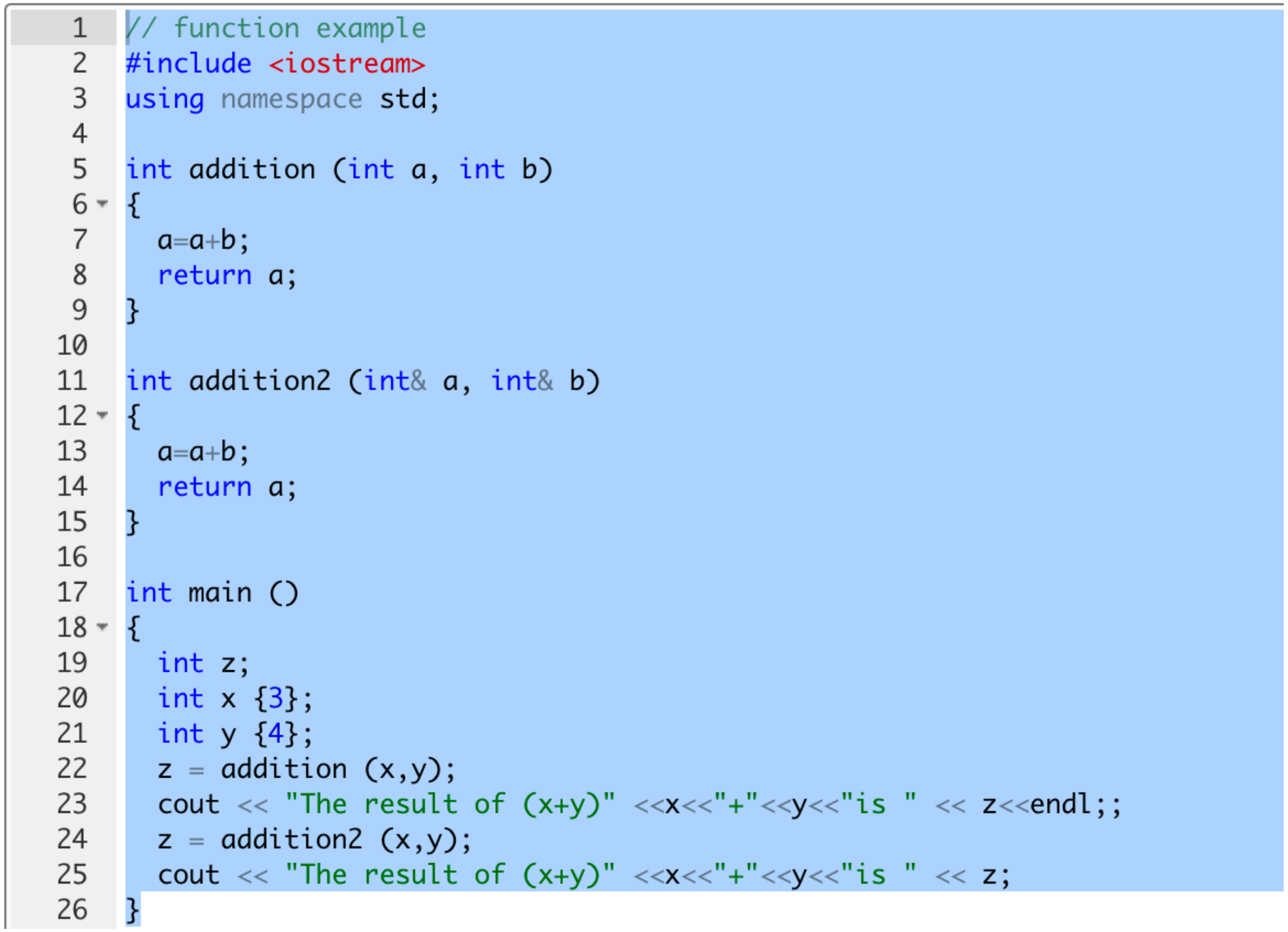


```cpp
// passing parameters by reference
#include <iostream>
using namespace std;

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

Basic of C++: functions

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

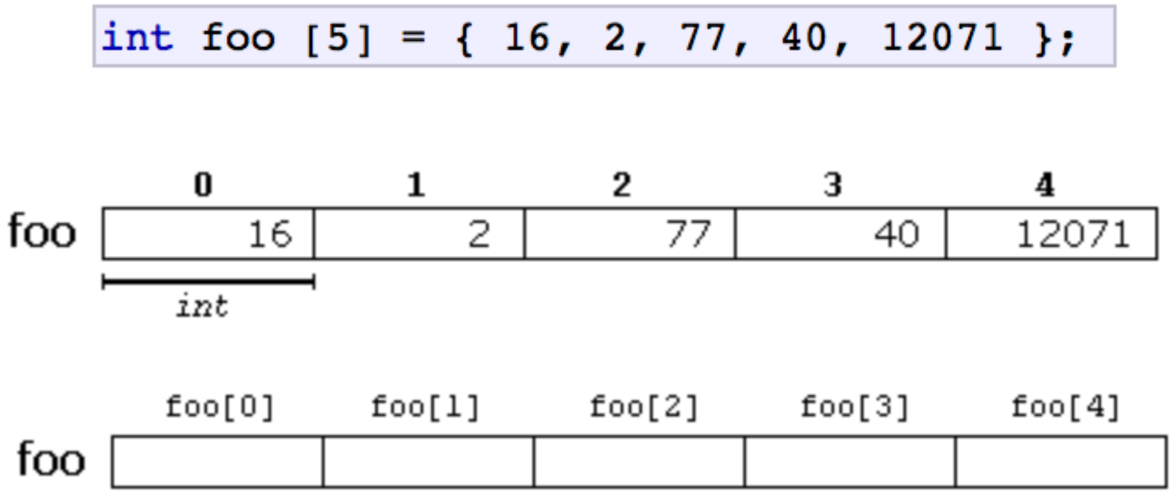


```
// function example
#include <iostream>
using namespace std;

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

int addition2 (int& a, int& b)
{
  a=a+b;
  return a;
}

int main ()
{
  int z;
  int x {3};
  int y {4};
  z = addition (x,y);
  cout << "The result of (x+y)" <<x<<"+"<<y<<"is " << z<<endl;;
  z = addition2 (x,y);
  cout << "The result of (x+y)" <<x<<"+"<<y<<"is " << z;
}
```

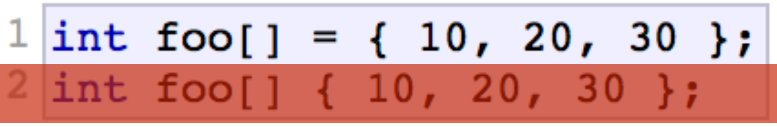
cpp.sh/74wt

Basic of C++: Array

Array = sequential memory space of the same type



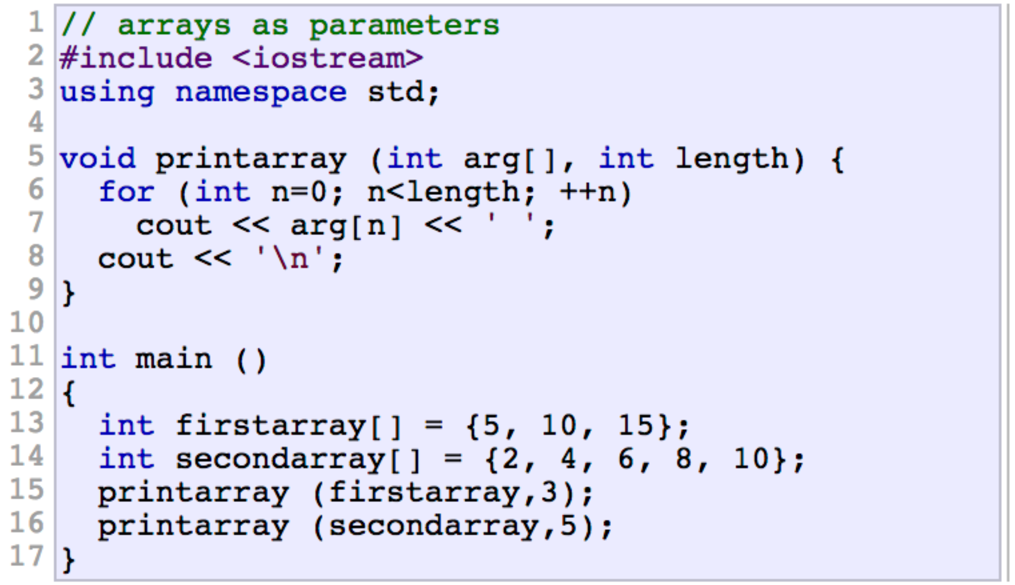
```
int foo [5] = { 16, 2, 77, 40, 12071 };
```



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

C++11

cpp.sh/6fzb



```
// arrays as parameters
#include <iostream>
using namespace std;

void printarray (int arg[], int length) {
  for (int n=0; n<length; ++n)
    cout << arg[n] << ' ';
  cout << '\n';
}

int main ()
{
  int firstarray[] = {5, 10, 15};
  int secondarray[] = {2, 4, 6, 8, 10};
  printarray (firstarray,3);
  printarray (secondarray,5);
}
```

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

cpp.sh/7aot

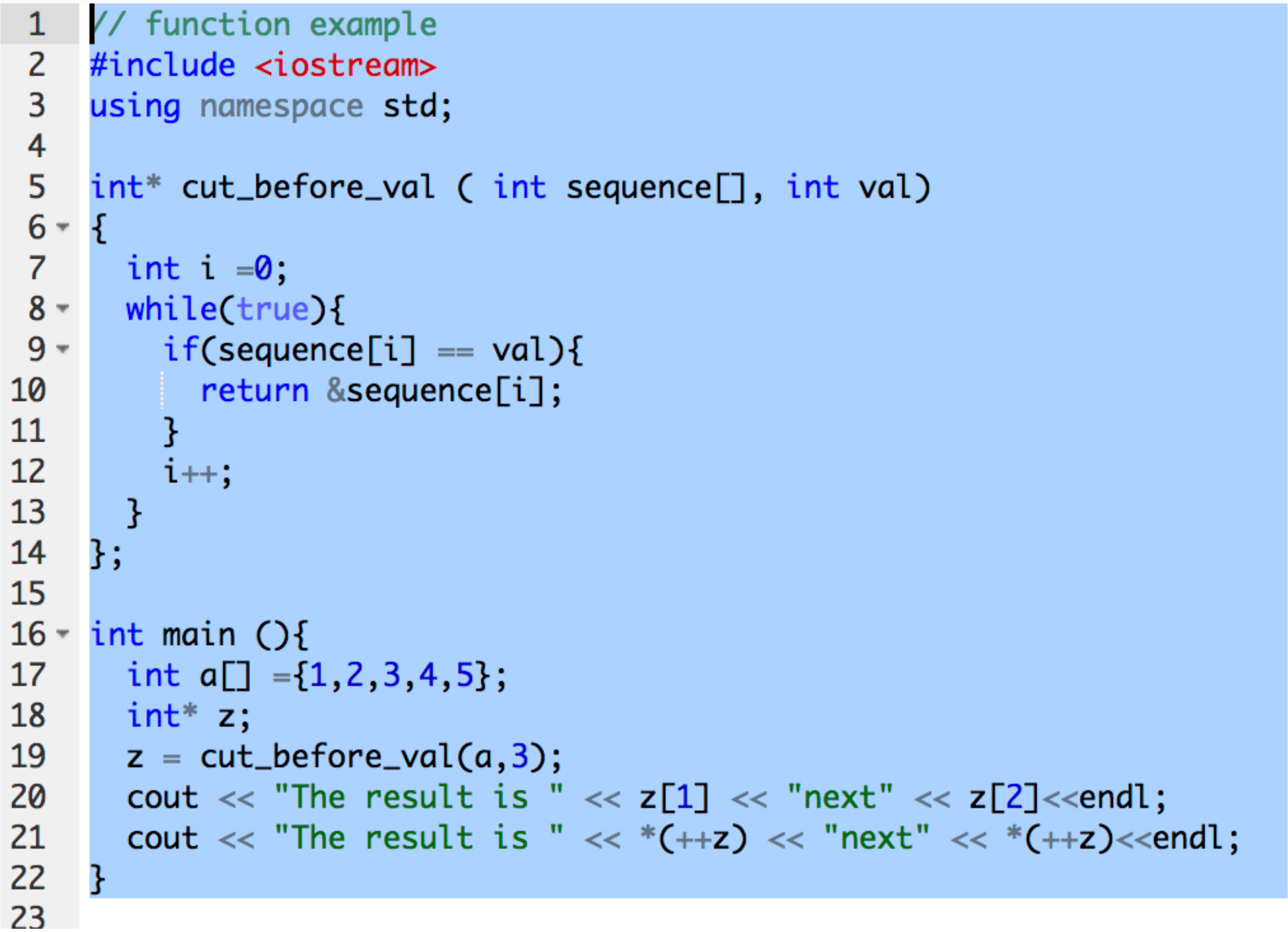
Exercise I

- Check that you can compile the Hello World example
- Define a function that take 3 float and return the average
  - ➡ Explore different method (variable/reference)
- 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
  - ✦ http://www.cplusplus.com/reference
  - ✦ http://www.cplusplus.com/doc/tutorial/

Solution

part I : cpp.sh/6ar2x

part II: cpp.sh/3wr4



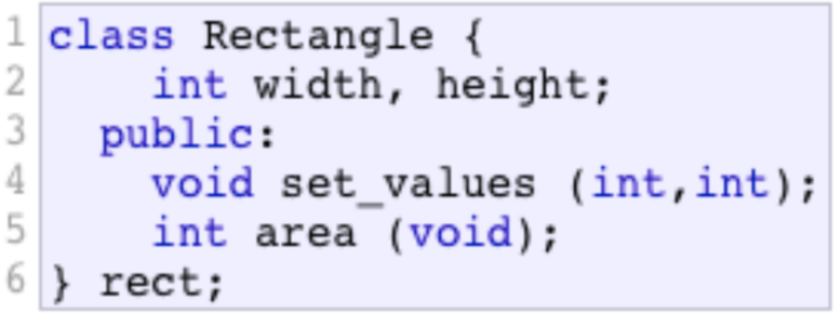
```
// function example
#include <iostream>
using namespace std;

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

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

```

Classes

classes = data structure with functions

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

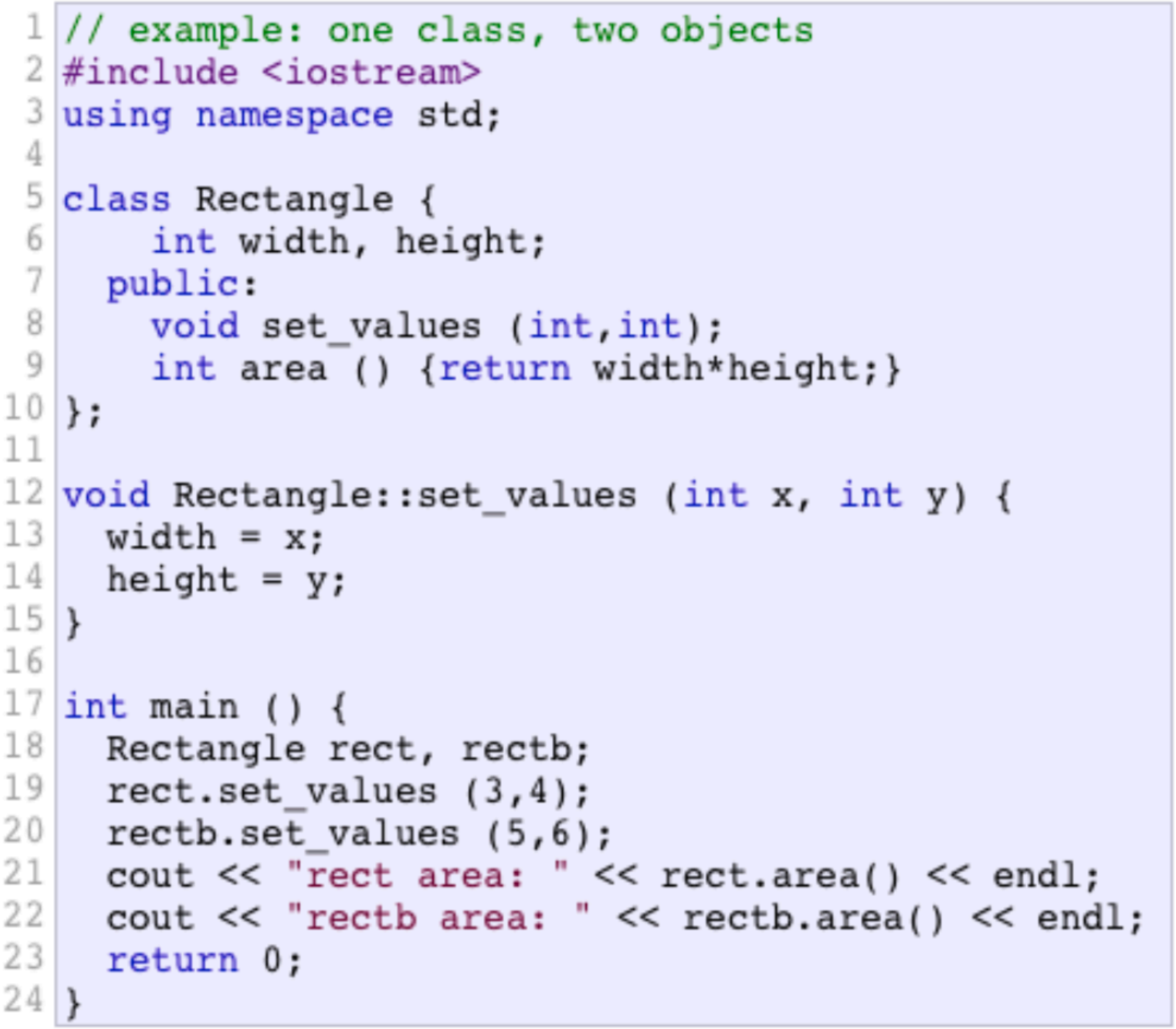
- We can define a class “Car”
  - ➡ Defines the structure
    - ✦ Which property available: attribute
      - model, colour, has_autodrive,nb_door
    - ✦ Which function can be applied.
      - change_battery, add_fuel,…
- Class is a new type like “int/float”
  - ➡ Car mytesla;
    - ✦ “mytesla” is an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



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

First Example

http://cpp.sh/8ac

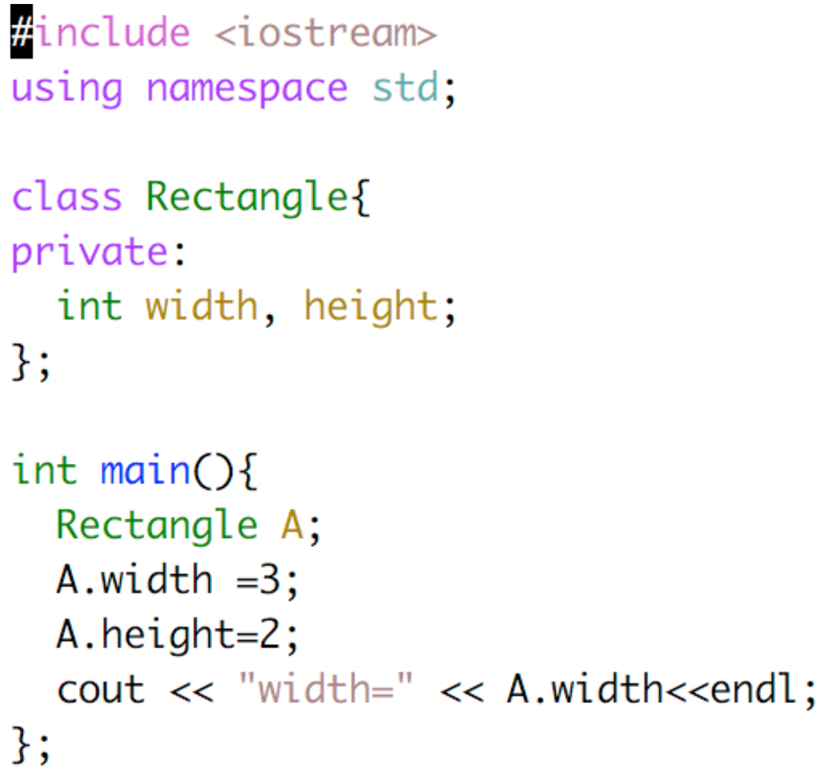


```
// example: one class, two objects
#include <iostream>
using namespace std;

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

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

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

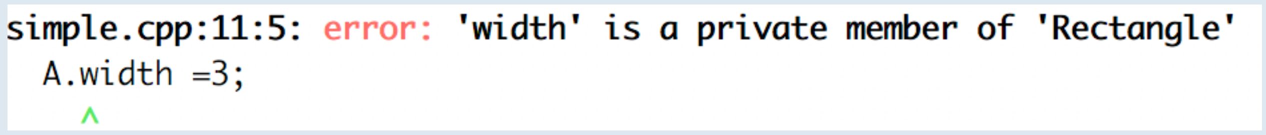
- width/height are private
- A public function allows to set those values!
- private attribute ensure that no one mess up those variables.

Visibility of attribute/function

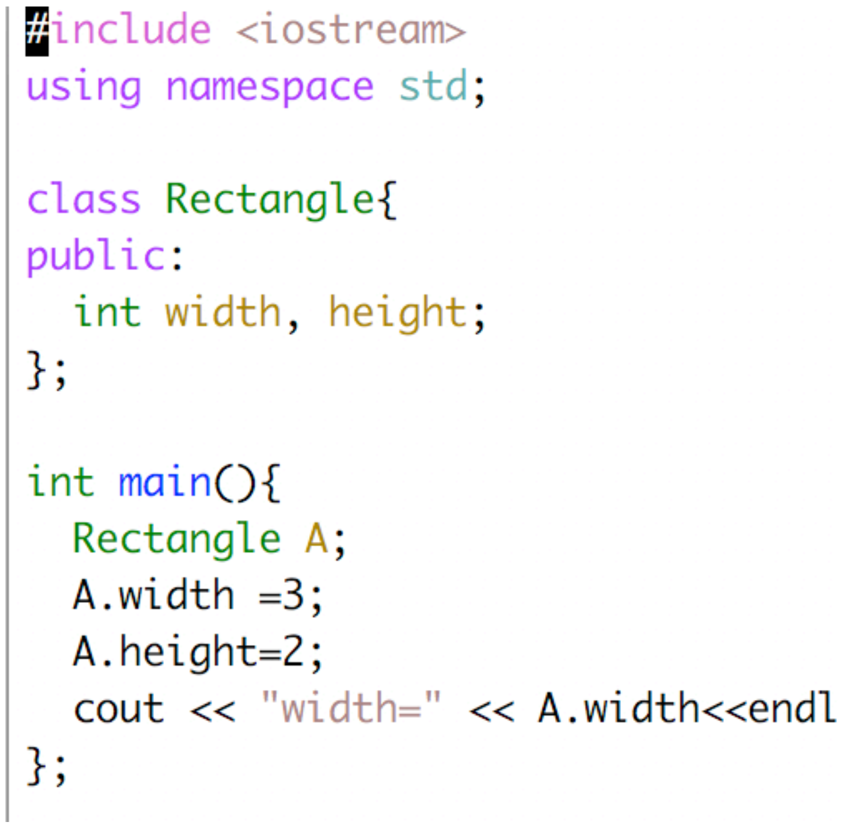
private	protected	public
Only accessible from other instance of the same class Accessible from friends DEFAULT	Accessible from other instance of the same class Accessible from friends Accessible from instance of the derived/child class	Accessible from everywhere where the object is visible READ and WRITE!



```
#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;
};
```

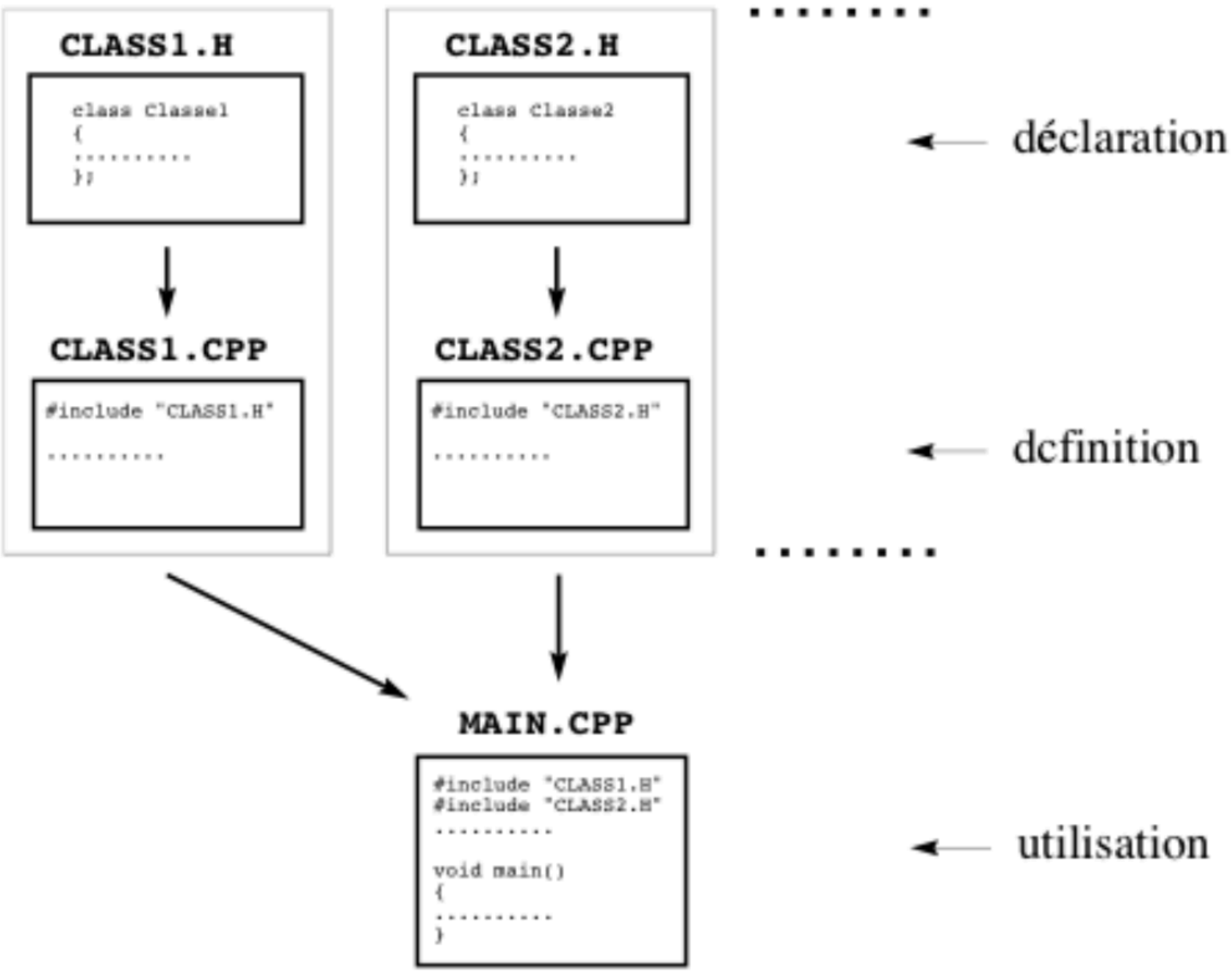


```
simple.cpp:11:5: error: 'width' is a private member of 'Rectangle'
  A.width =3;
    ^
```



```
#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
};
```

Code Structure



```
class Classe1
{
..........
};
```

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

```
class Classe2
{
..........
};
```

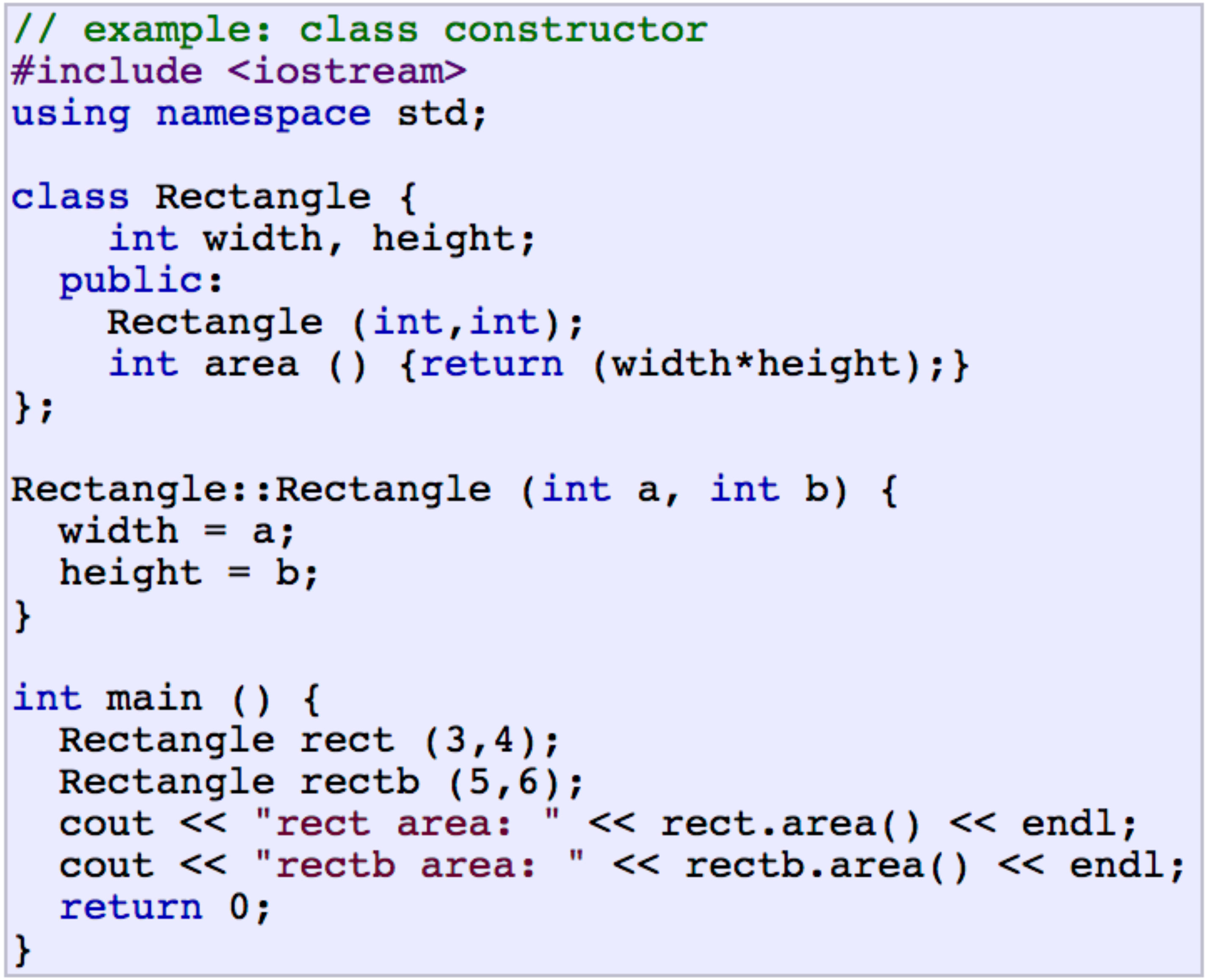
```
#include "CLASS2.H"
..........
```

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

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

Constructor

constructor = function called after the object is created

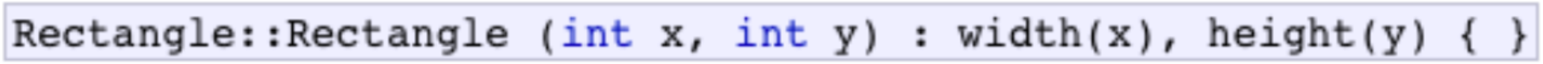
cpp.sh/8lr



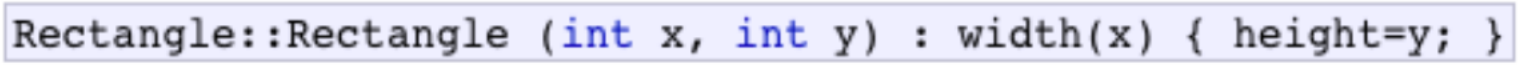
```
// example: class constructor
#include <iostream>
using namespace std;

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!

- Shortcut for setting attribute



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

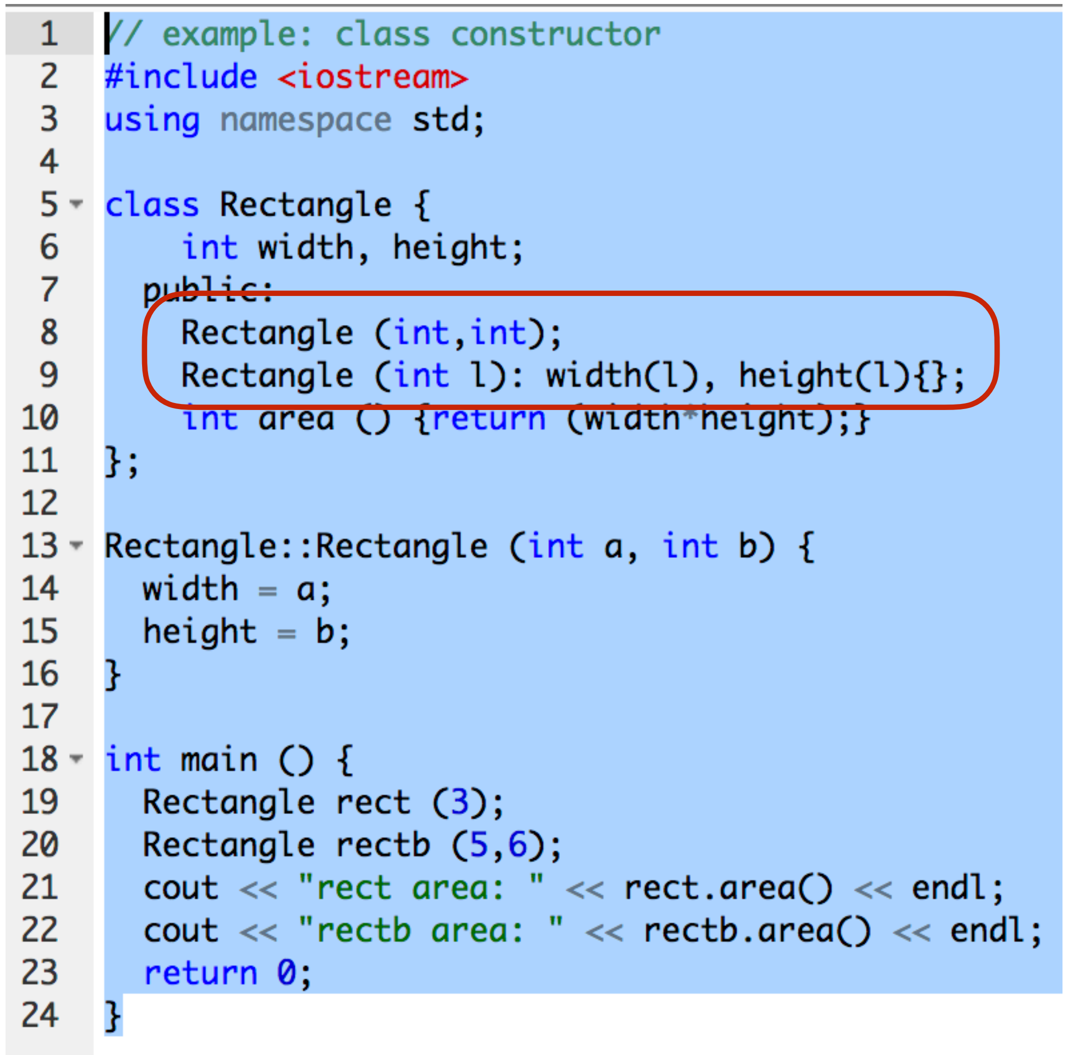


```
Rectangle::Rectangle (int x, int y) : width(x) { height=y; }
```

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.

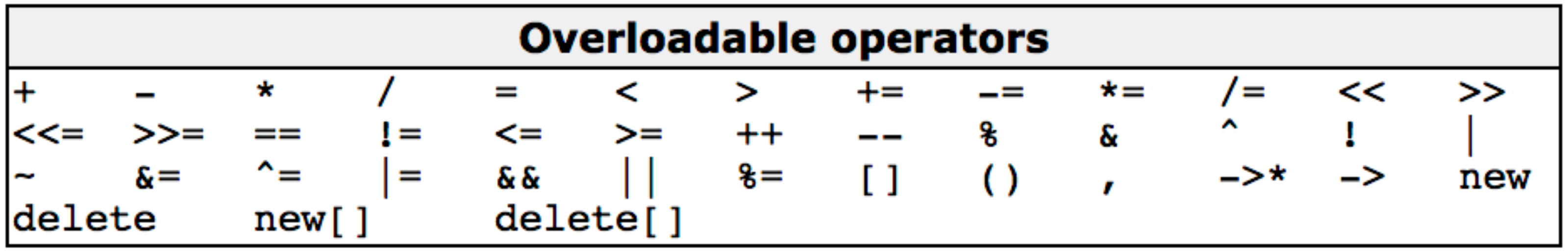


```
// example: class constructor
#include <iostream>
using namespace std;

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 +

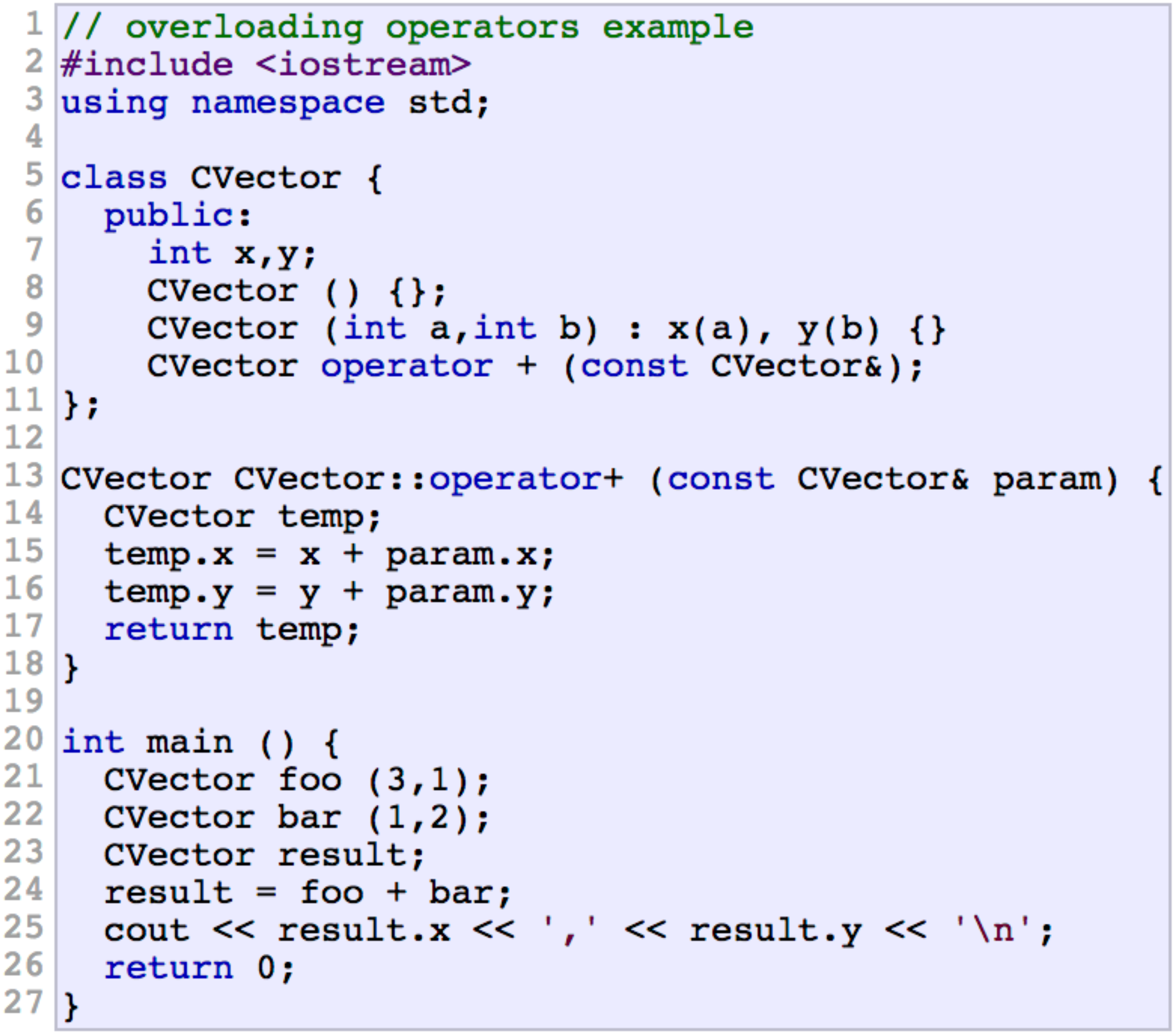
Overloading

Overloading = more than one function with the same name



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

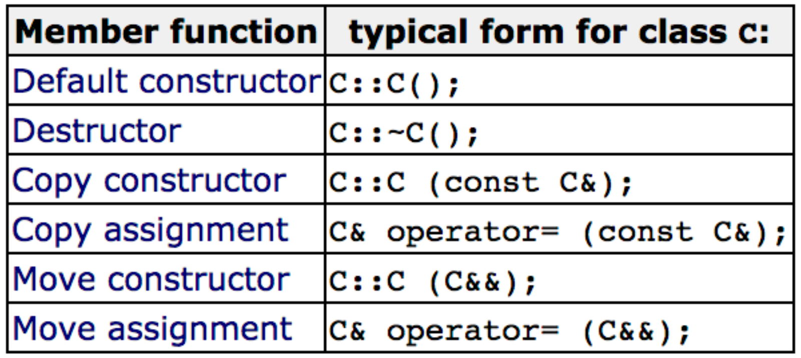
cpp.sh/27l



```
// 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;
}
```

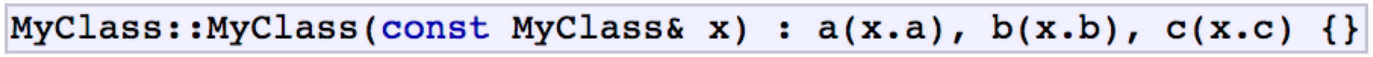
Special members

Special members = member functions implicitly defined

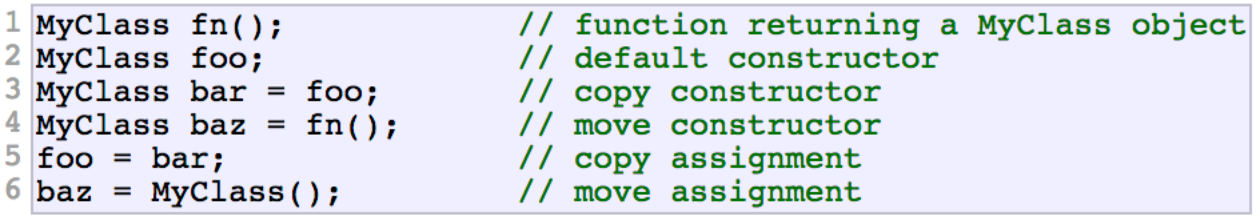


Member function	typical form for class C:
Default constructor	`C::C();`
Destructor	`C::~C();`
Copy constructor	`C::C (const C&);`
Copy assignment	`C& operator= (const C&);`
Move constructor	`C::C (C&&);`
Move assignment	`C& operator= (C&&);`

- 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



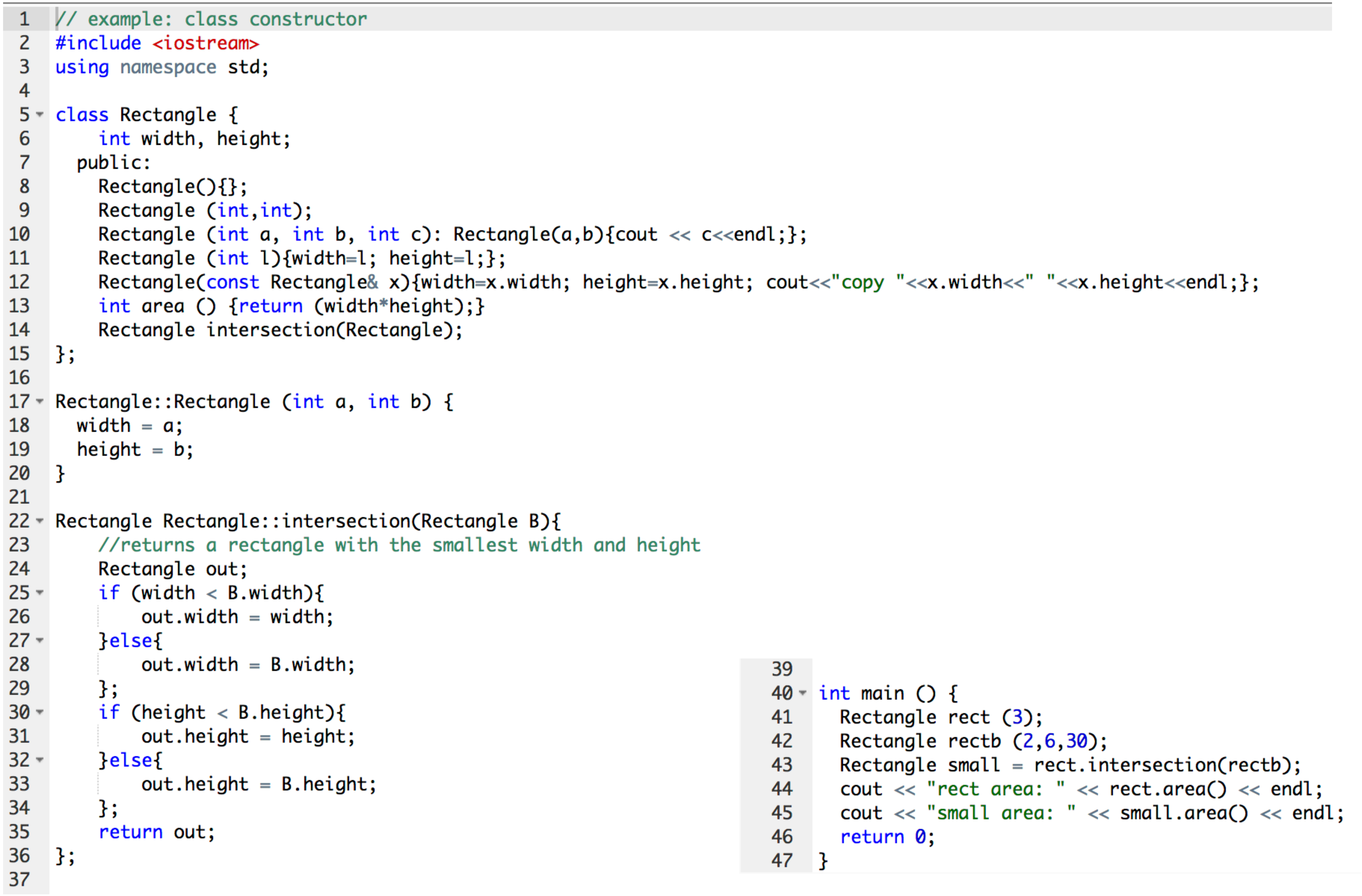
```
MyClass::MyClass(const MyClass& x) : a(x.a), b(x.b), c(x.c) {}
```



```
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;
}
```

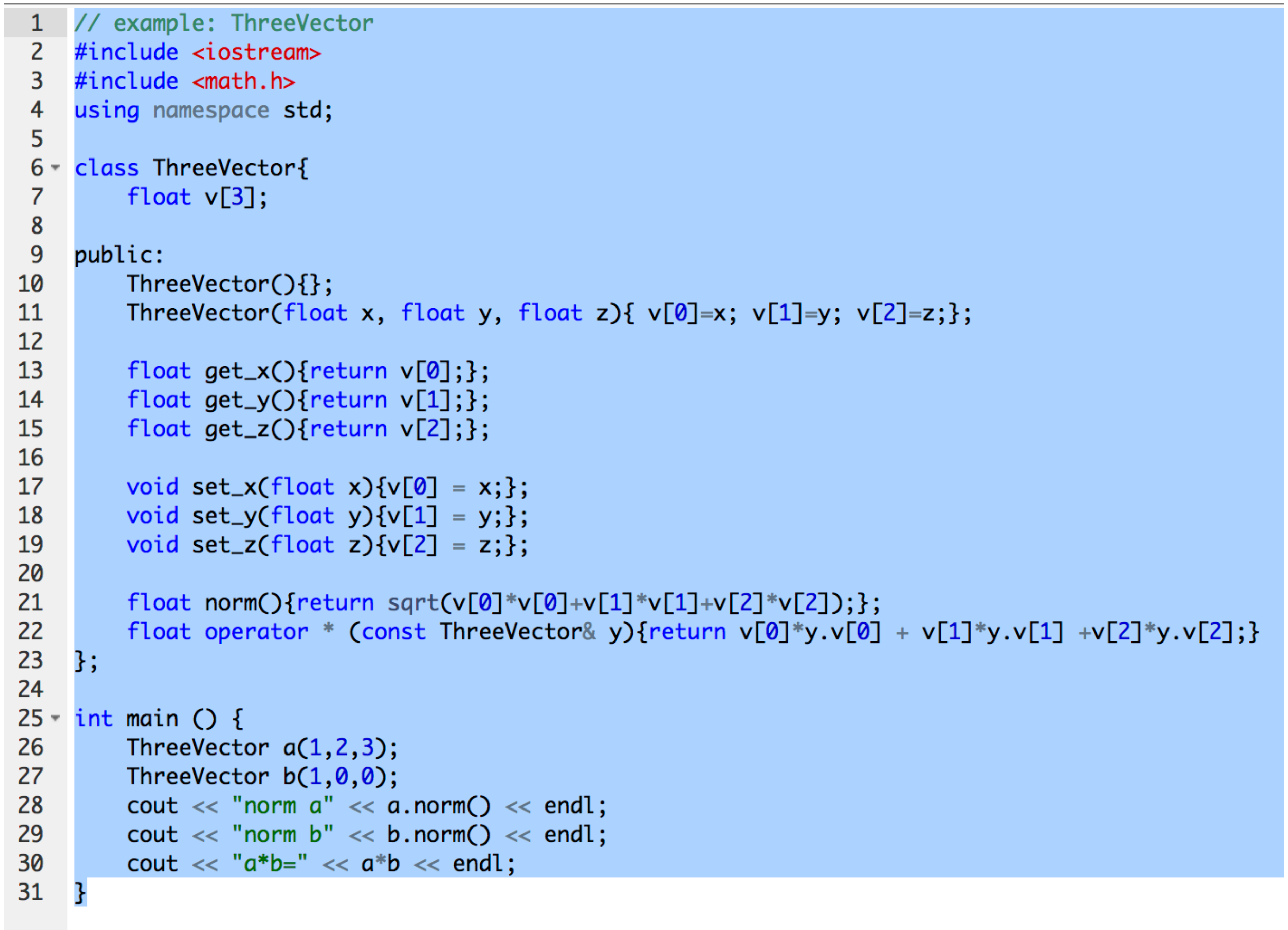
Exercise II

- 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 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)

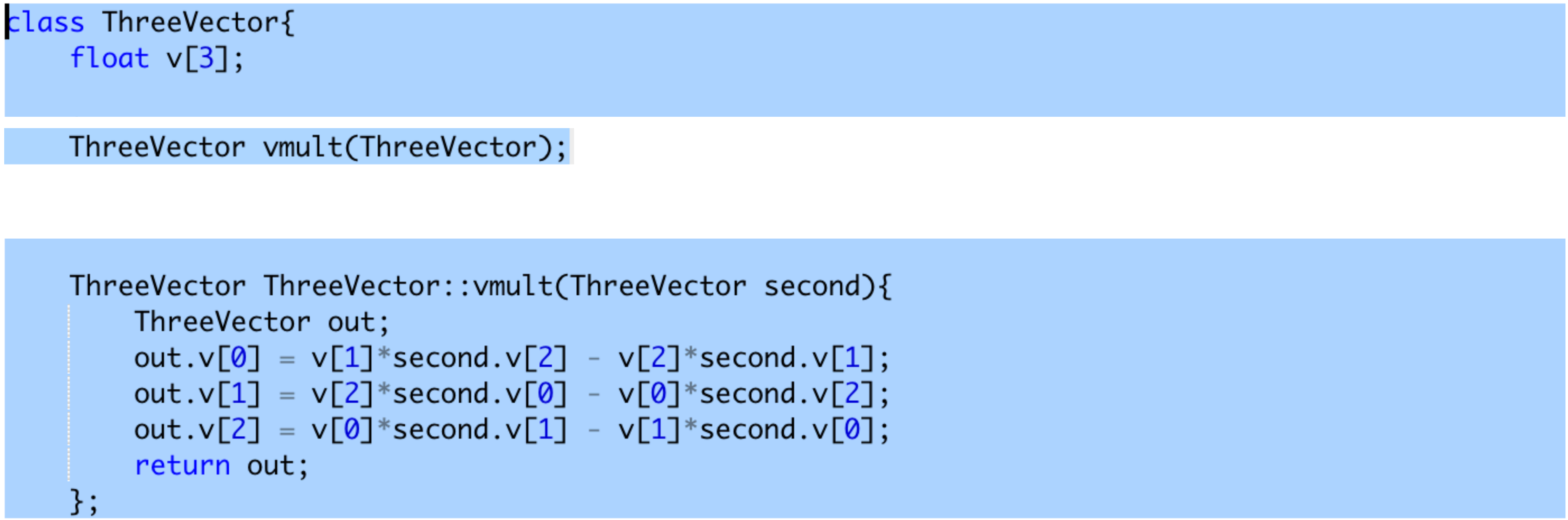
Solution

cpp.sh/6vgu2c



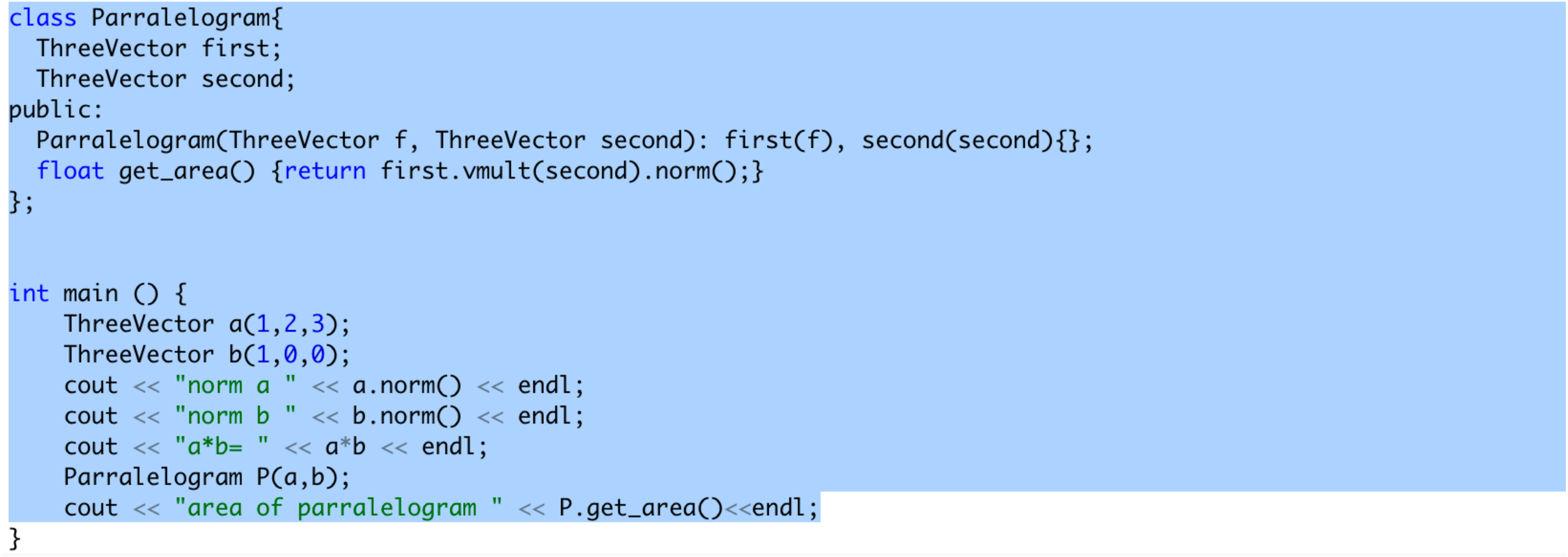
```cpp
// example: ThreeVector
#include <iostream>
#include <math.h>
using namespace std;

class ThreeVector{
    float v[3];

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

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

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

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

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

Solution



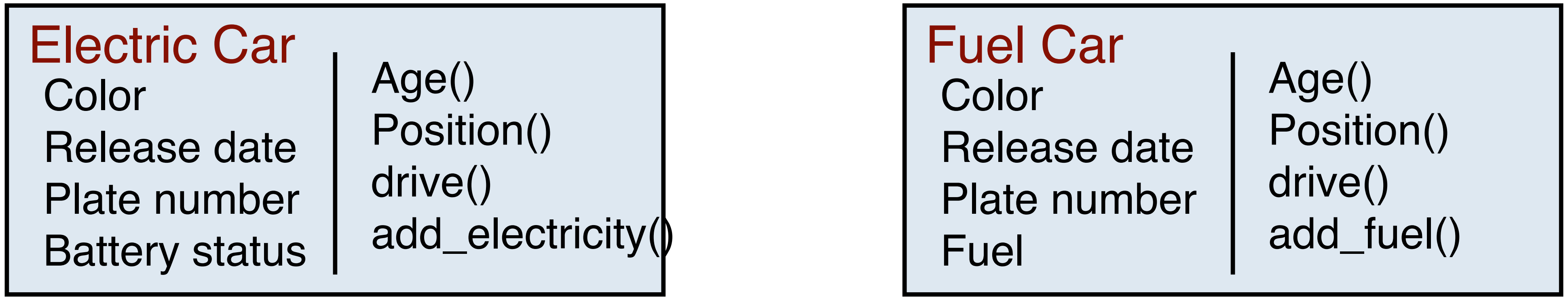
```cpp
class ThreeVector{
    float v[3];

    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



```cpp
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

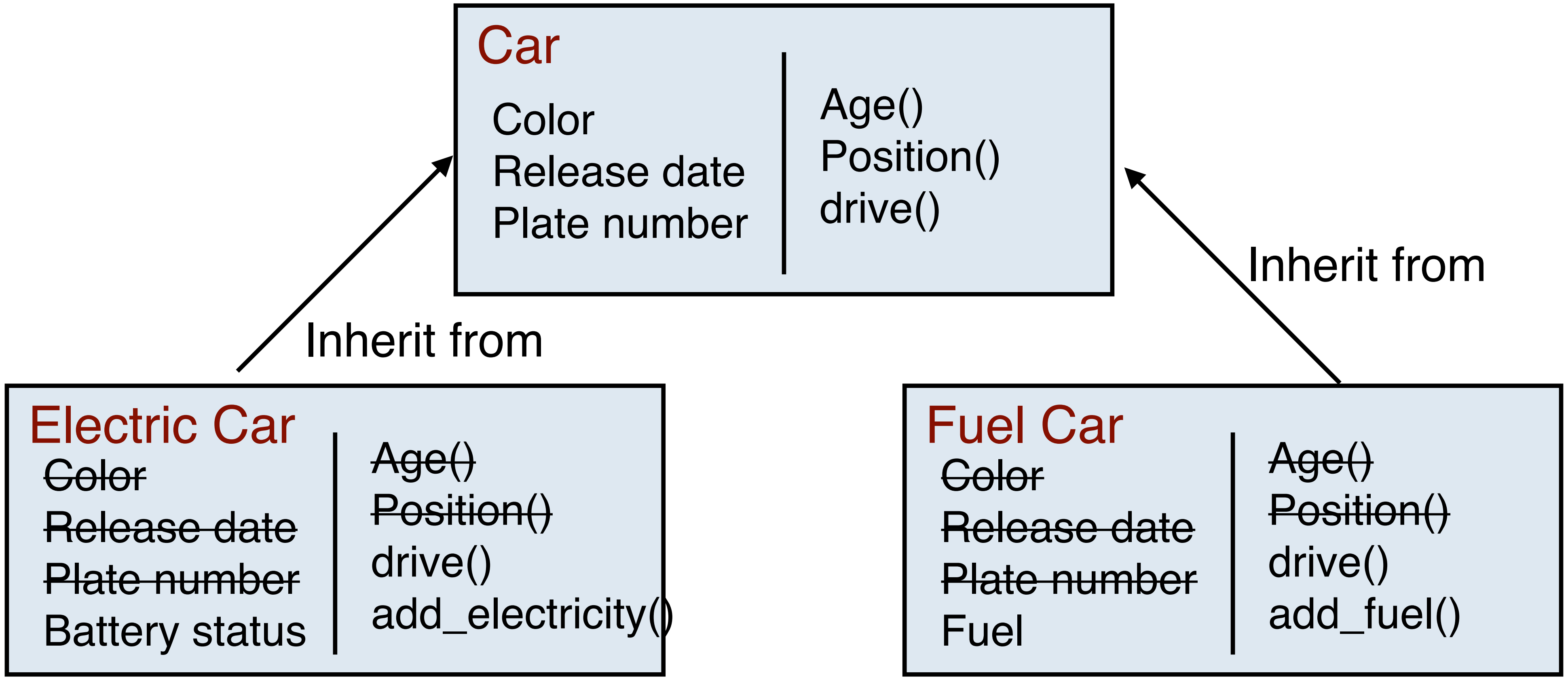
Age()
Position()
drive()
add_electricity()

Fuel Car

Color
Release date
Plate number
Fuel

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

Inheritance



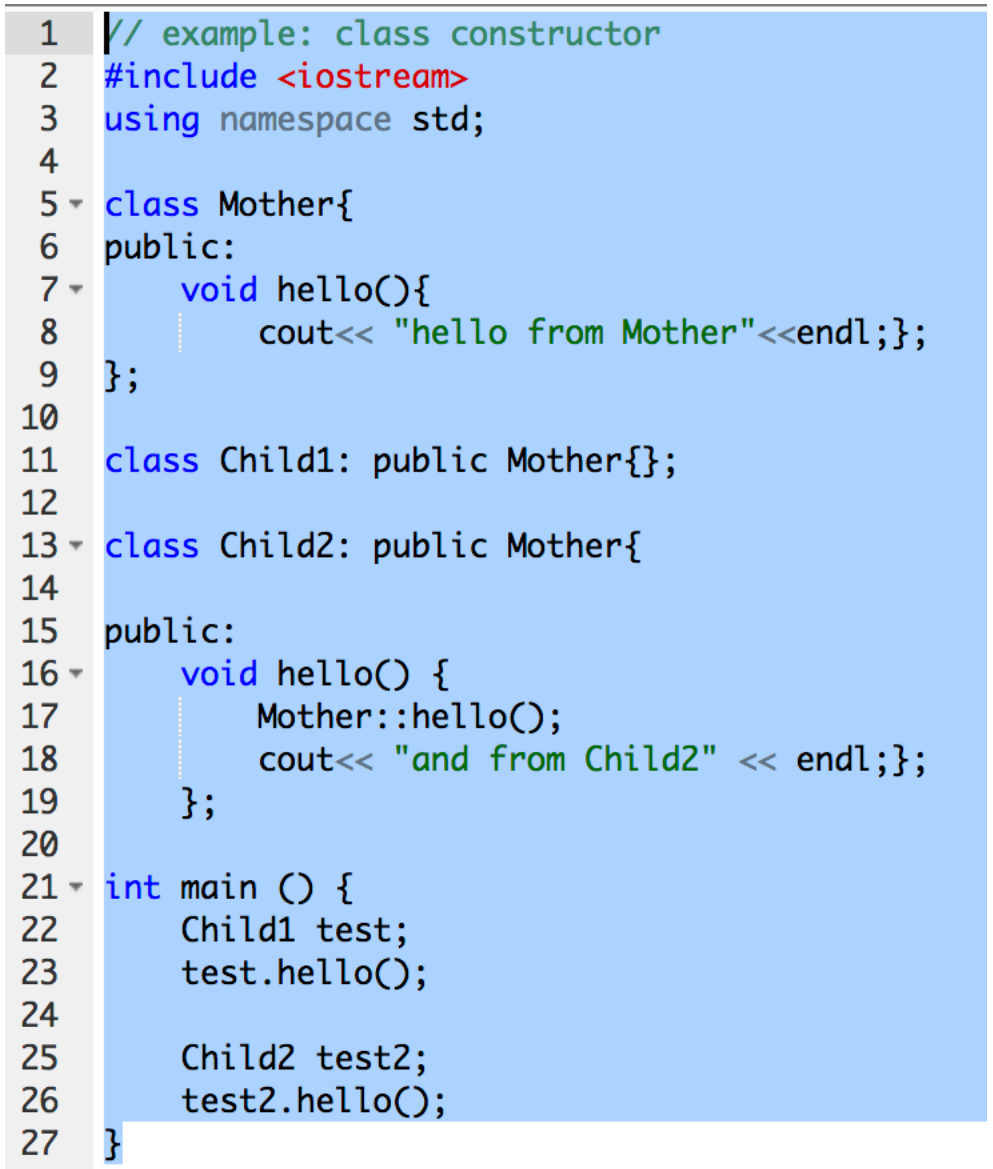
- The two class (Electric/fuel car) does not to redefine their structure just what they changed compare to the original class!
- They can change/ammend the behaviour

Inheritance

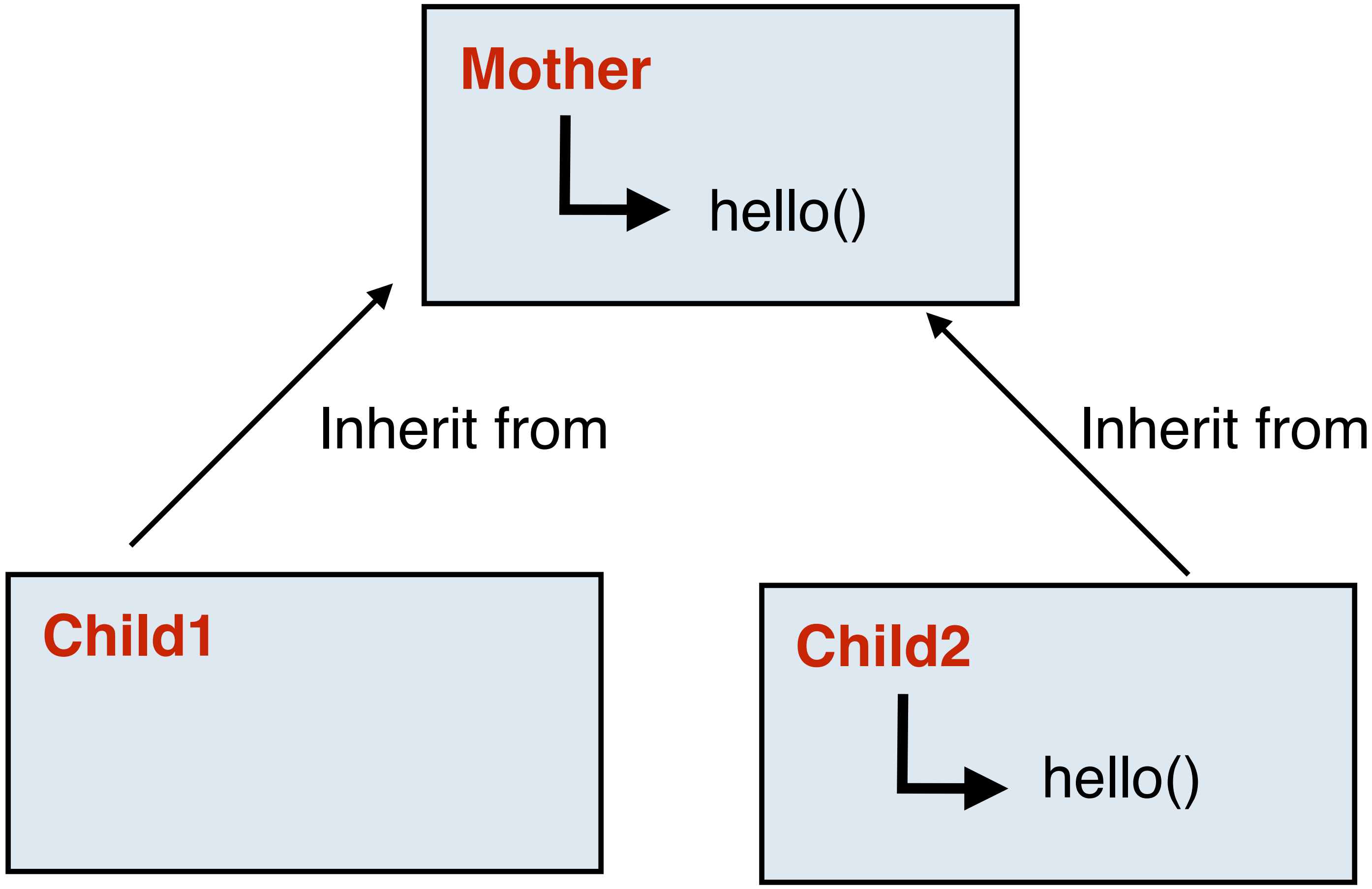
Inheritance = new classes which retain characteristics of the base class.

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

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();
}
```



Mother

hello()

Inherit from

Inherit from

Child1

Child2

hello()

Can still call hello()

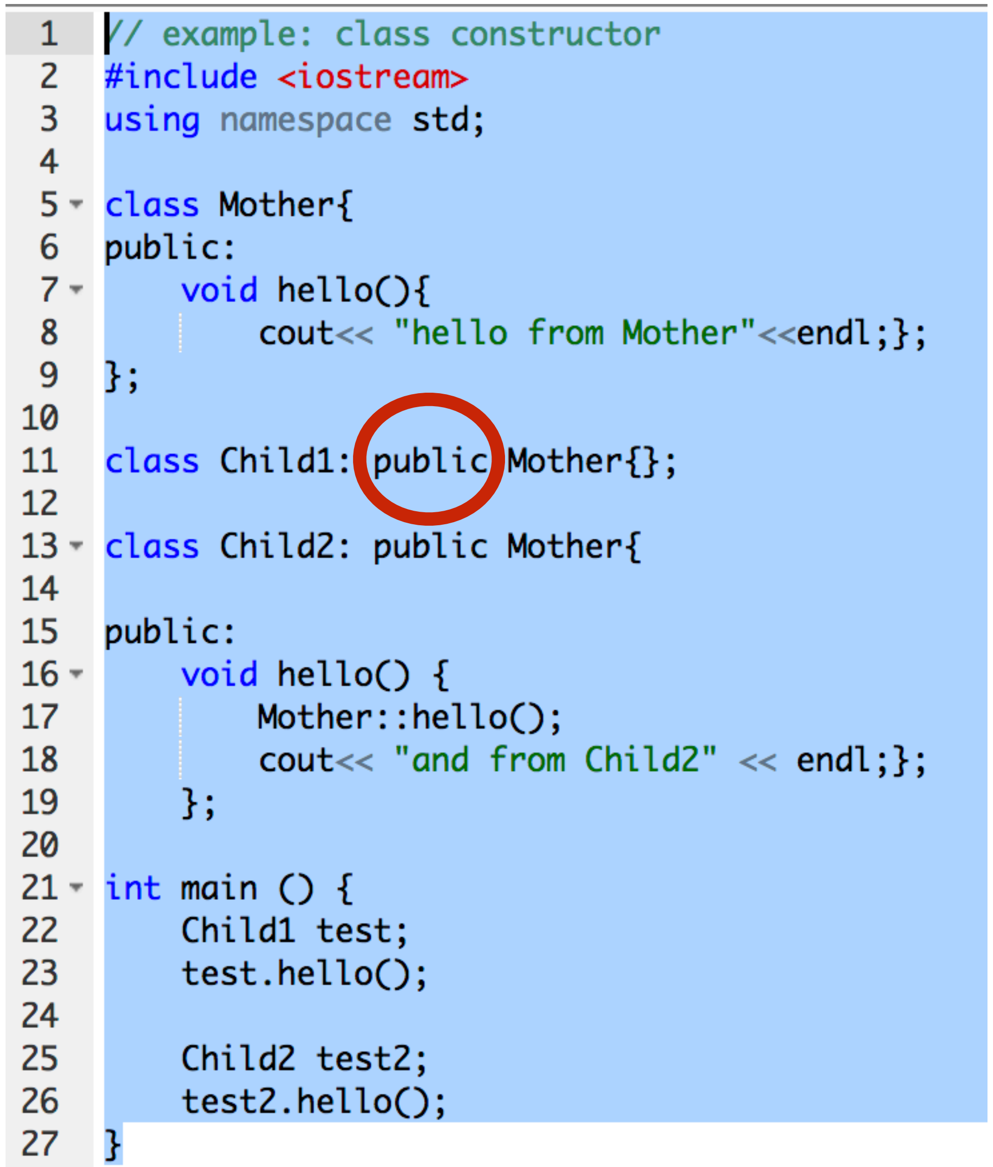
Overwrite hello()
-> can still call the mother one if needed

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

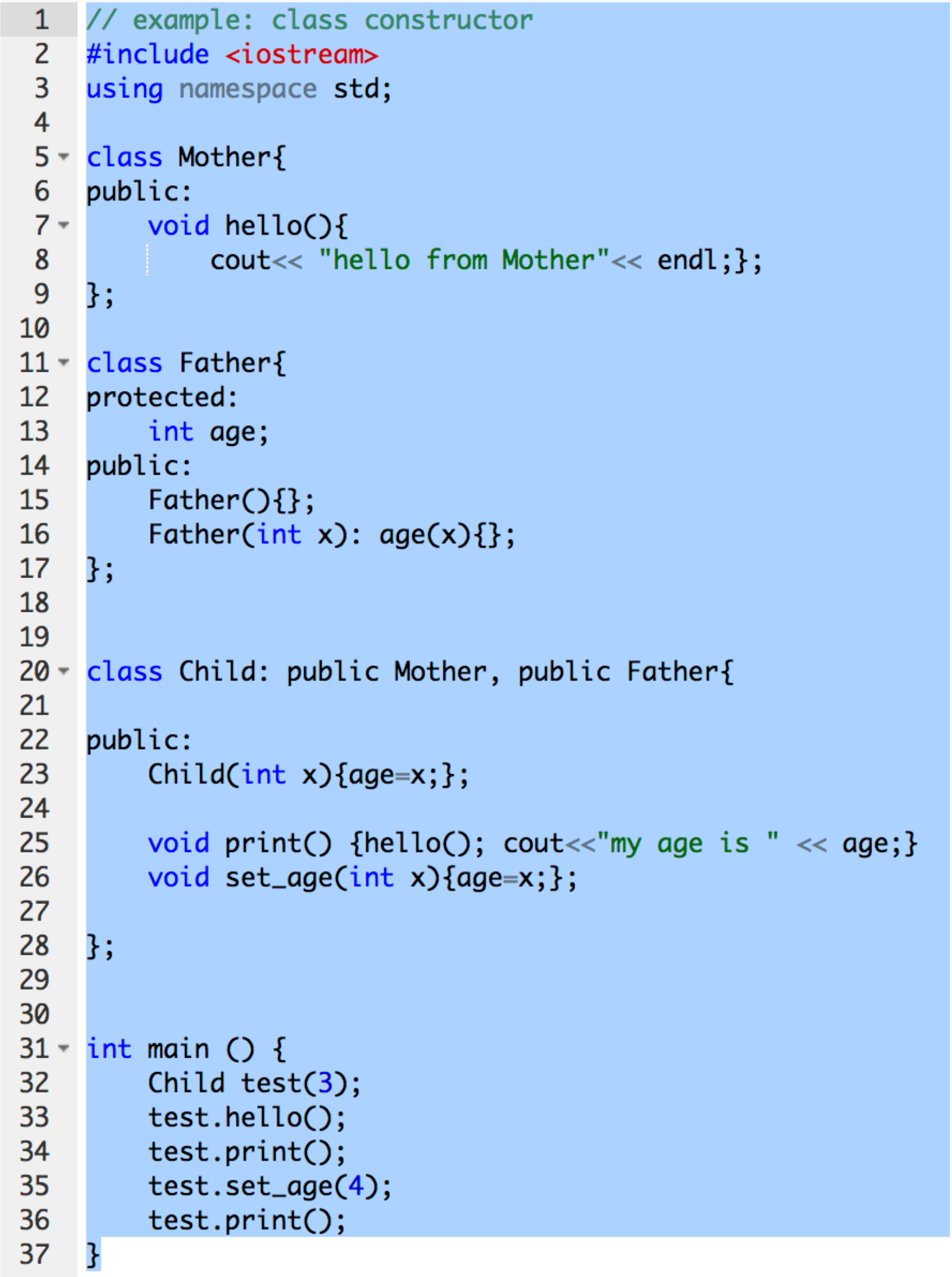


```
// example: class constructor
#include <iostream>
using namespace std;

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();
}
```

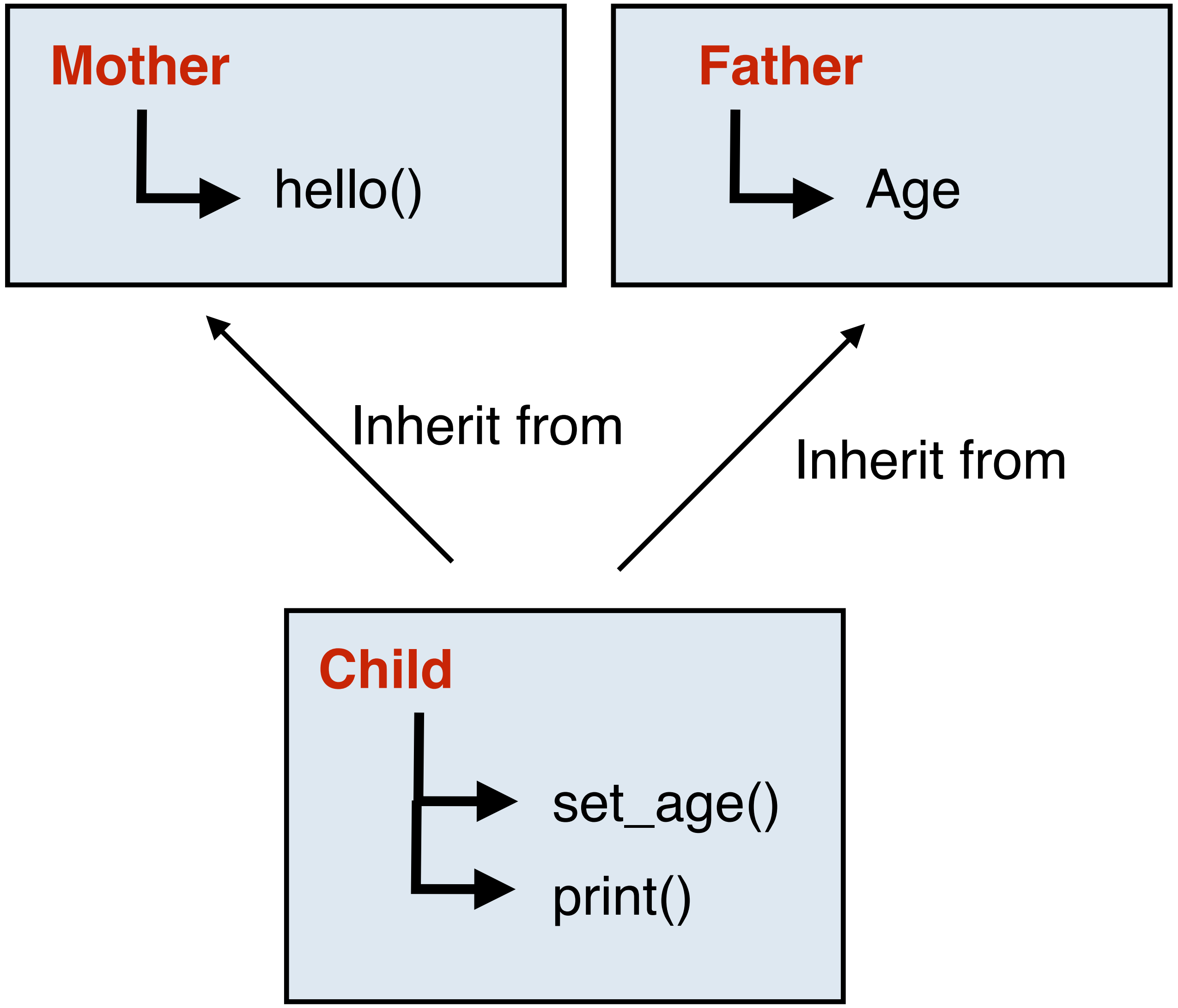
- “public” tells the maximum level of visibility of the attribute coming from the base class
- Private argument are not passed to the child (but they still exits!)
- Constructor/Destructor are **not** passed to the child
- Assignment operator (operator =) are **not** passed to the child

Multi-inheritance

cpp.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();
}
```

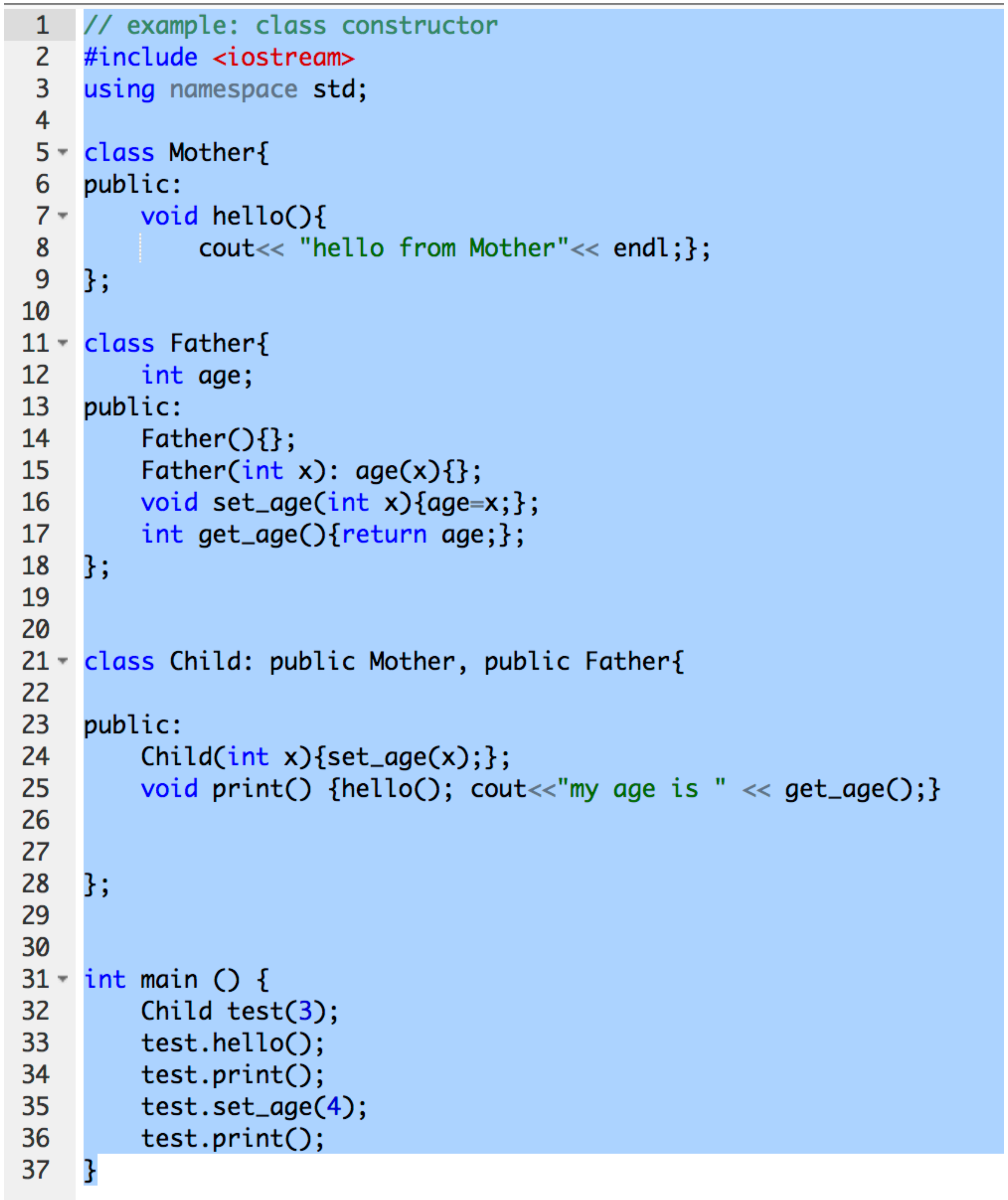


Can still call hello()

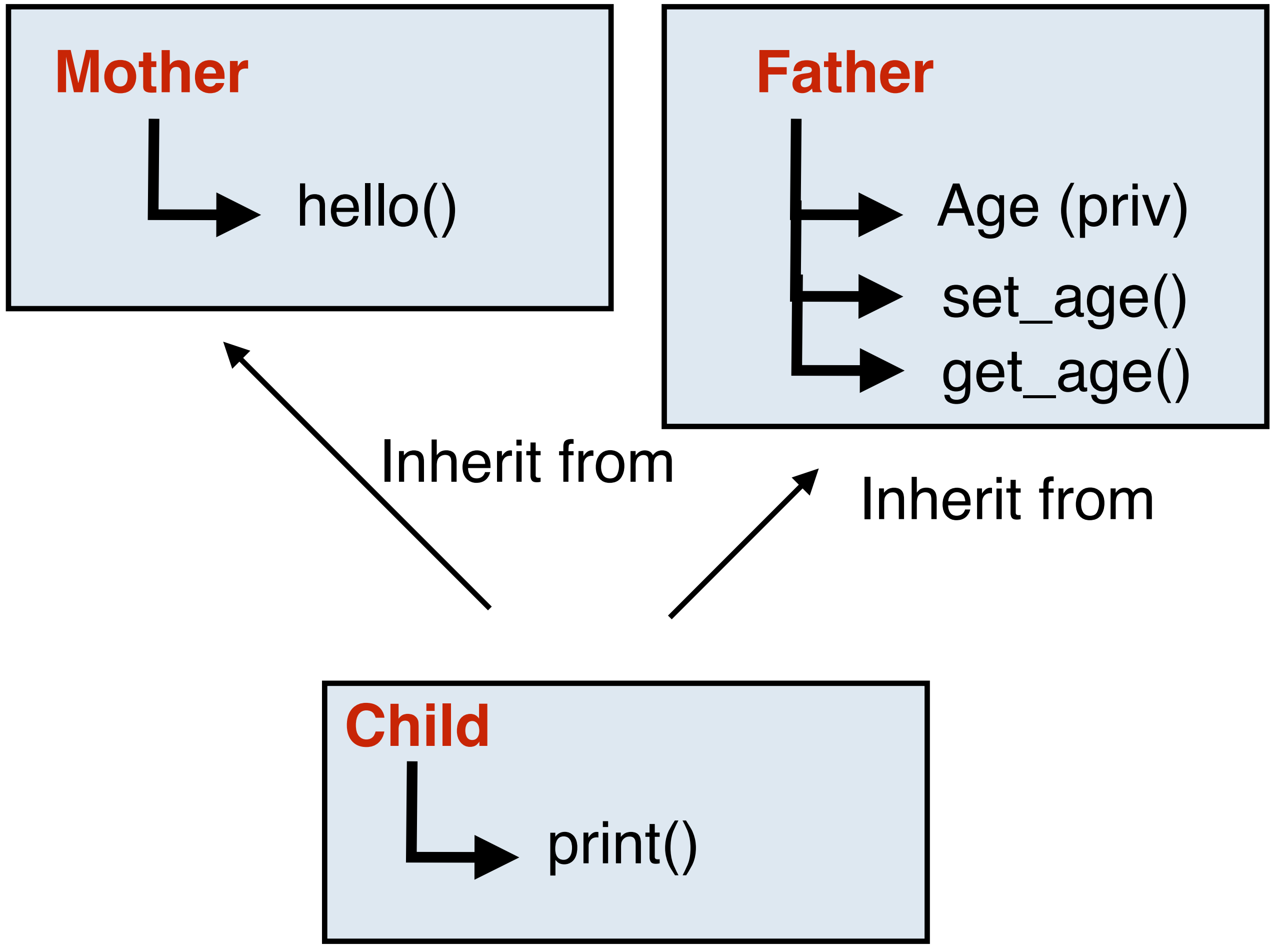
Can access to age (protected)

Multi-inheritance

cpp.sh/8vev



```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()

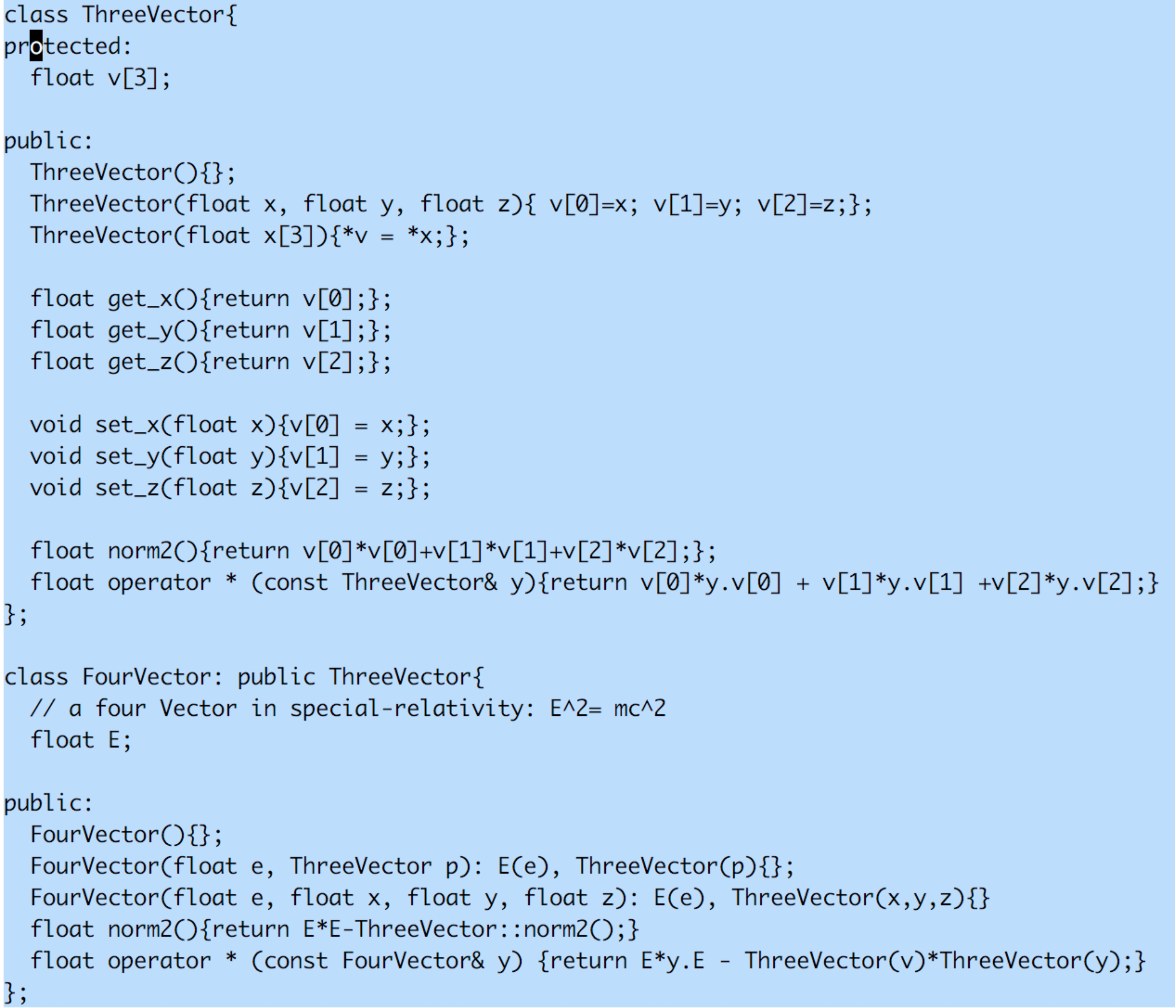
Can not call age (since private)
But can call the public routine of father which set/get the age variable

Exercise III

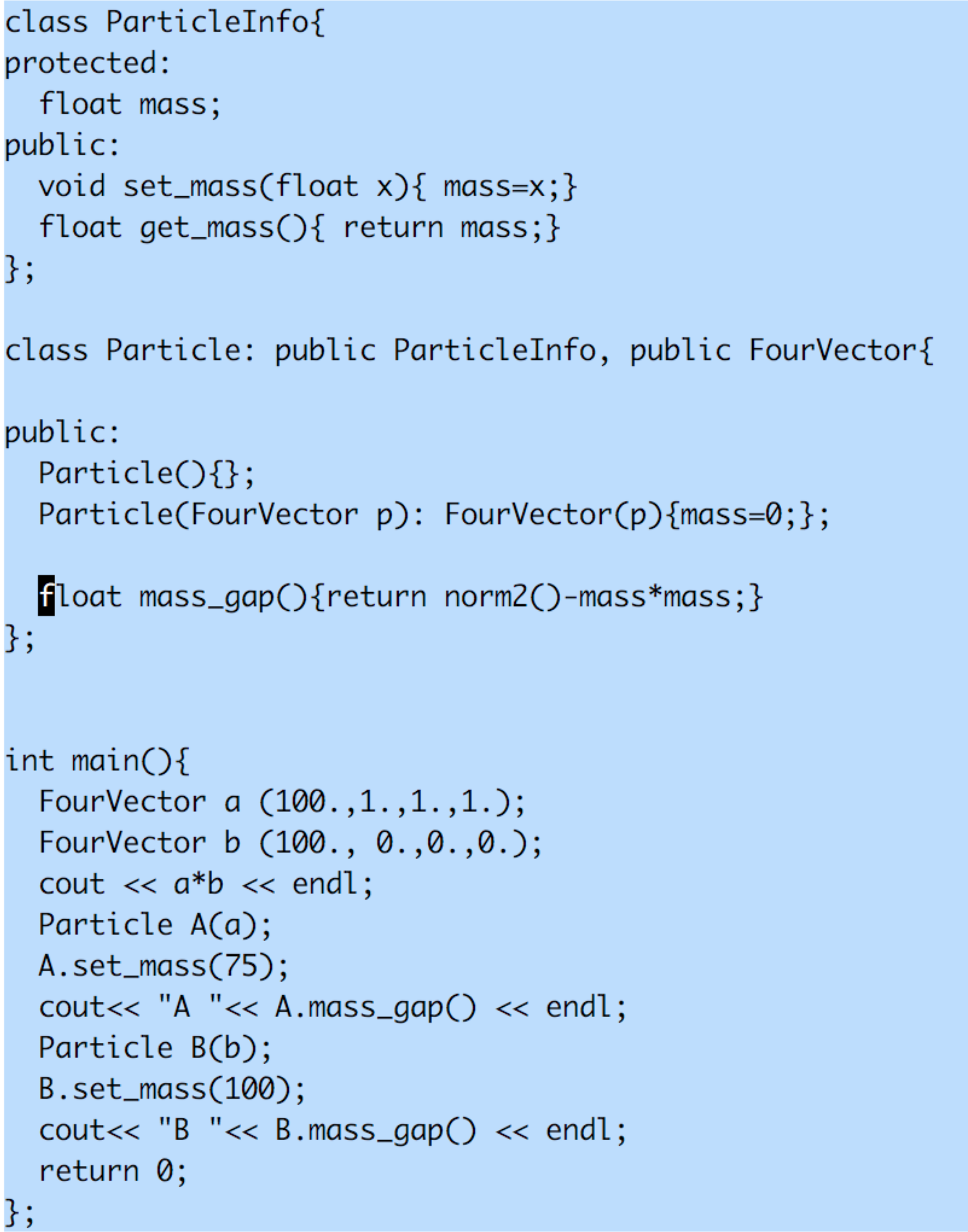
- Define a class Four-Vector which inherit from your class 3 vector
  - ➡ Define the norm like in special relativity
    - ✦ x*x = x[0]x[0] -x[1]x[1]-x[2]x[2]-x[3]x[3]
- Define a class ParticleInfo
  - ➡ Has some attribute (mass/width)
- Define a class Particle which inherit from both class
  - ➡ Define a function which computes the difference between the mass square and the norm squared.

Solution

cpp.sh/2jen



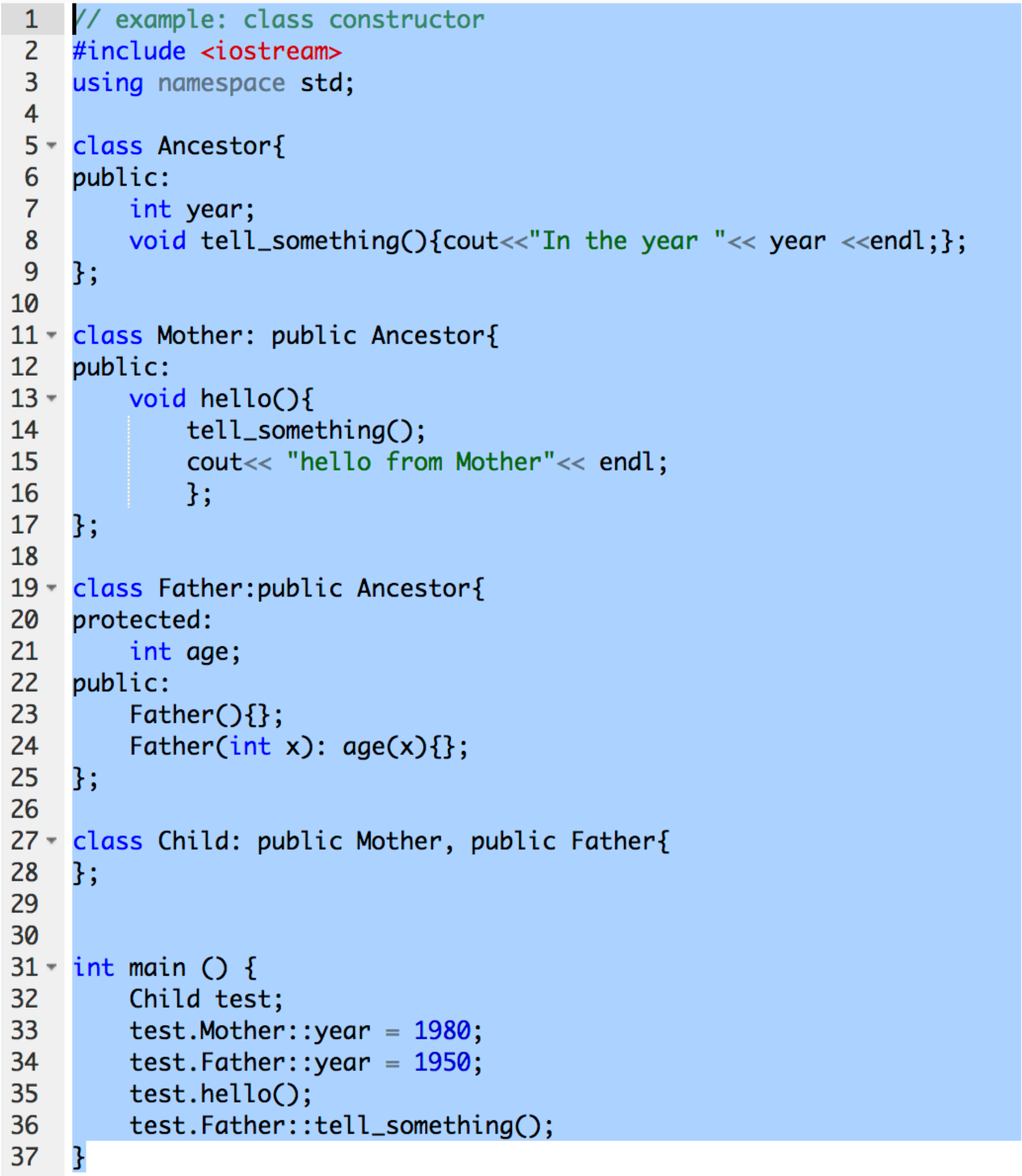
```
class ThreeVector{
protected:
  float v[3];

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);}
};
```



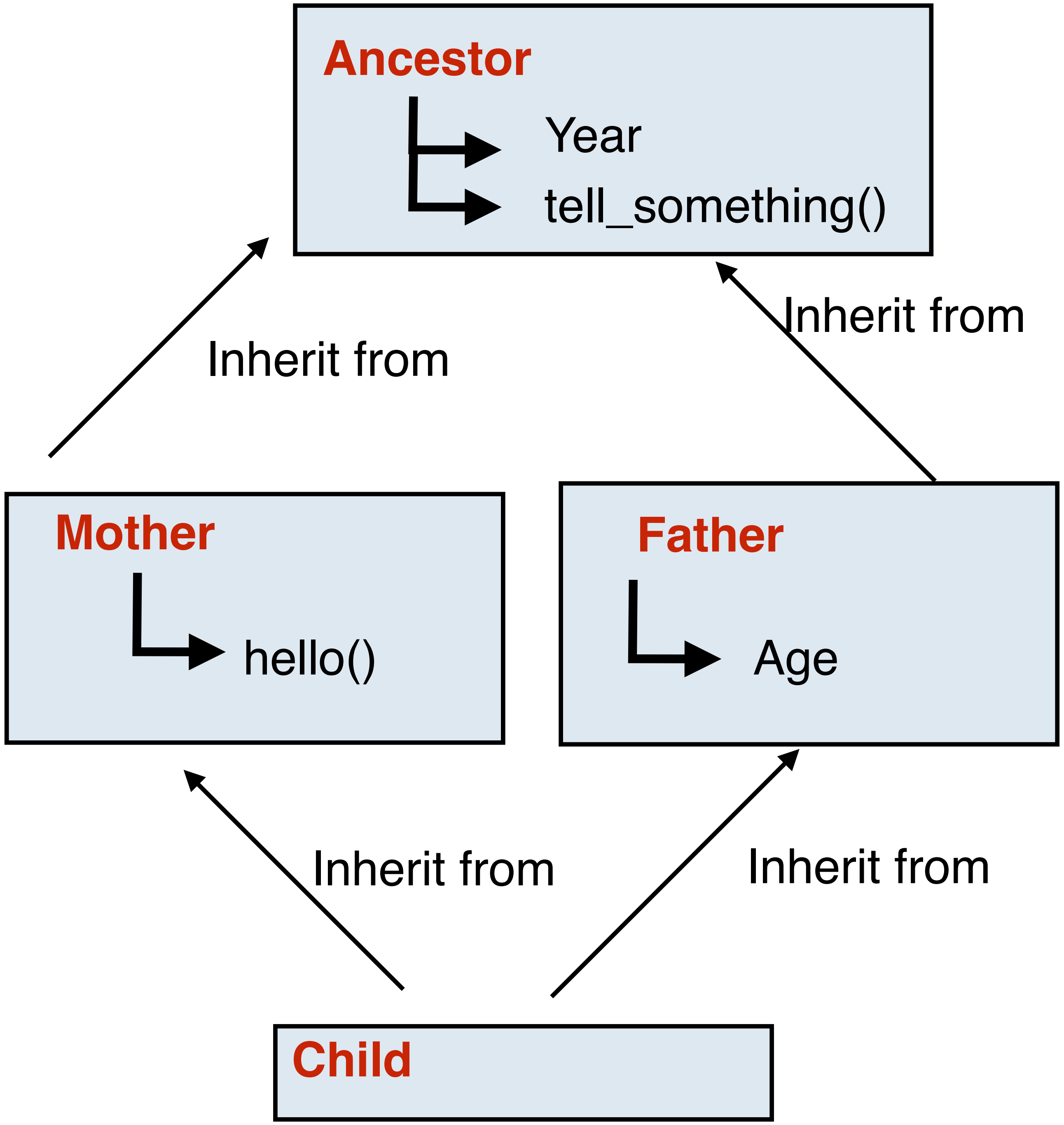
```
class ParticleInfo{
protected:
  float mass;
public:
  void set_mass(float x){ mass=x;}
  float get_mass(){ return mass;}
};

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

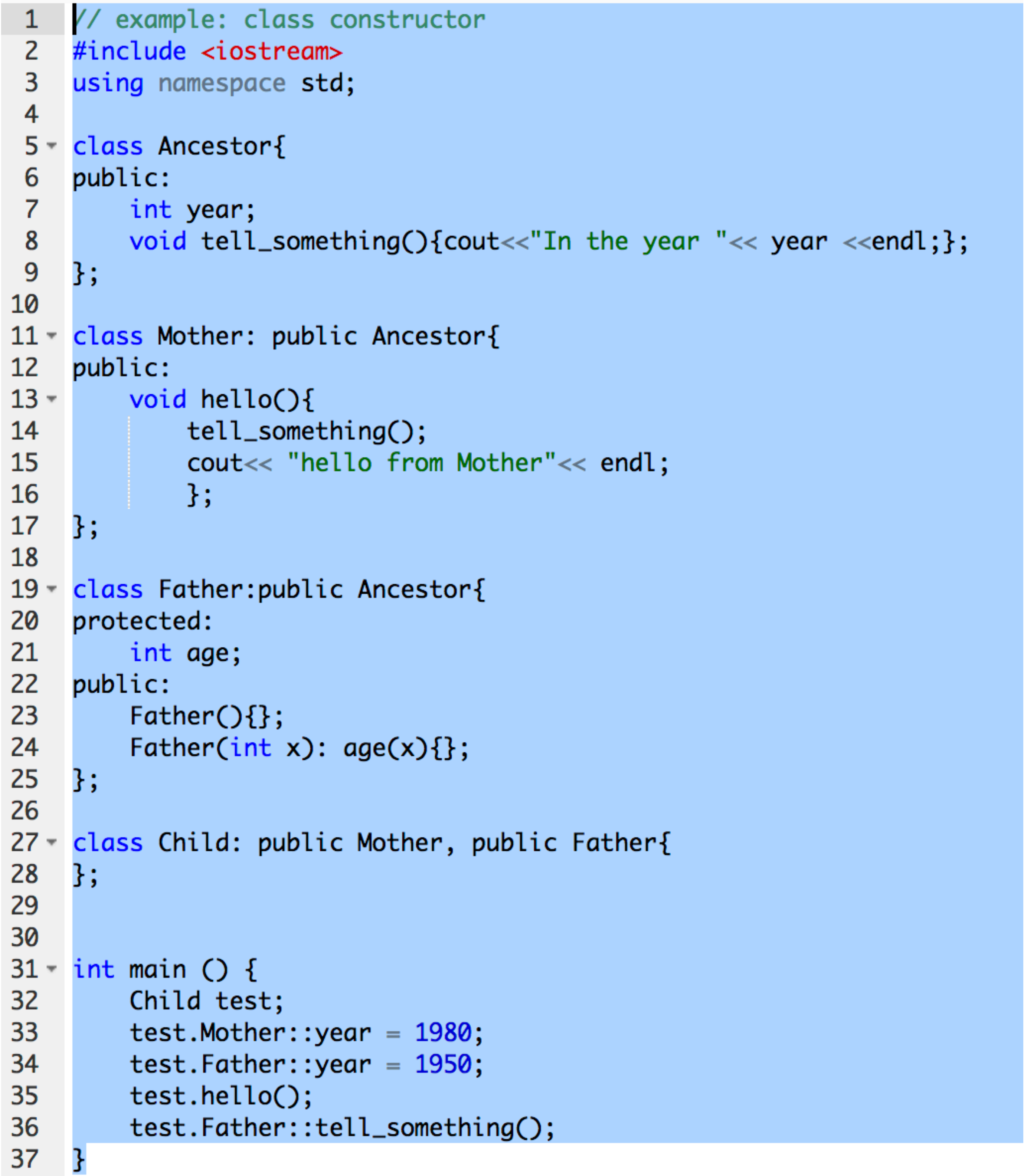


```
// example: class constructor
#include <iostream>
using namespace std;

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();
}
```



Diamond Diagram

cpp.sh/4inoj



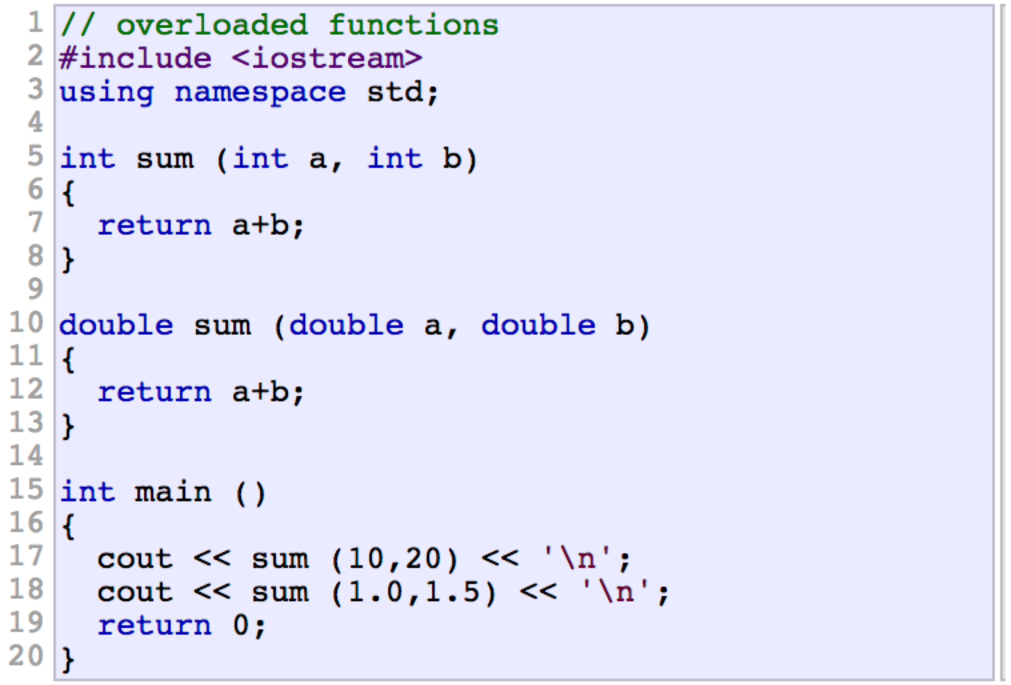
```
// example: class constructor
#include <iostream>
using namespace std;

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!

Template

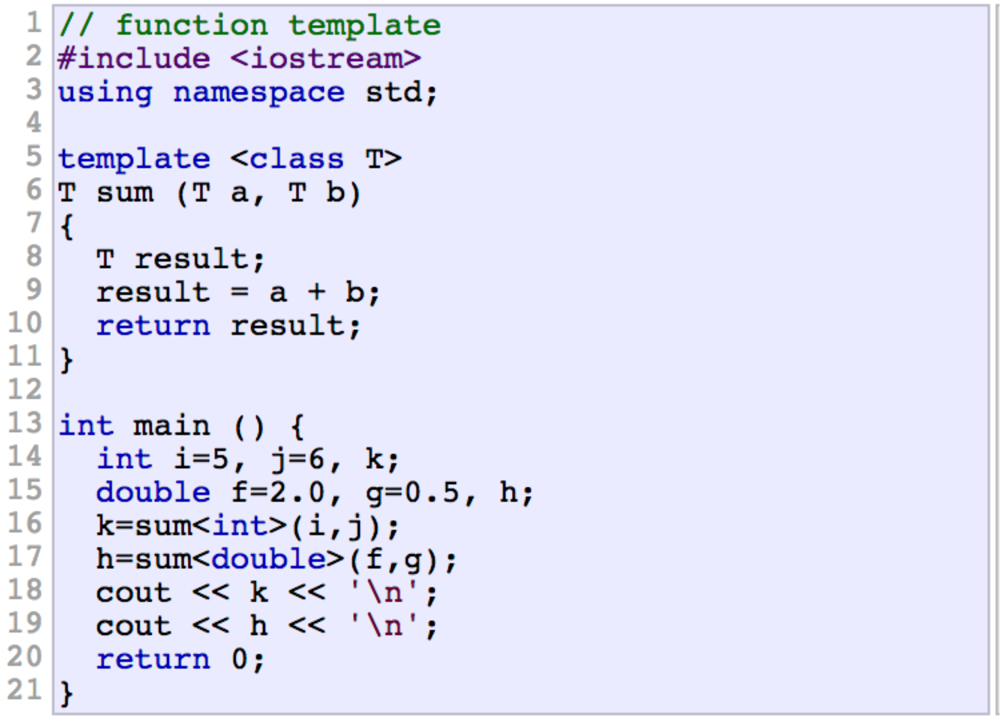
Template = define functions class with generic type

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



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

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

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

int main ()
{
  cout << sum (10,20) << '\n';
  cout << sum (1.0,1.5) << '\n';
  return 0;
}
```

cpp.sh/4jq



```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 IV

- Update your four-vector class to include
  - ➡ Scalar Multiplication via Template Method
- Test Multi-Heritage on your class
  - ➡ Test virtual heritage on one/two parent class/…
- Have fun…

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/…