Introduction to Object Oriented programming with C++

http://www.cplusplus.com/doc/tutorial/

CISM/CÉCI Training Sessions - 21/10/2016

Etienne Huens

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 the data are kept as a single unit called object
  - a user can access the data is via the object's methods
  - the internal workings of an object may be changed without affecting any code that uses the object

Structure of a program

http://cpp.sh/2dd

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

1 : comment (also /*...*/)

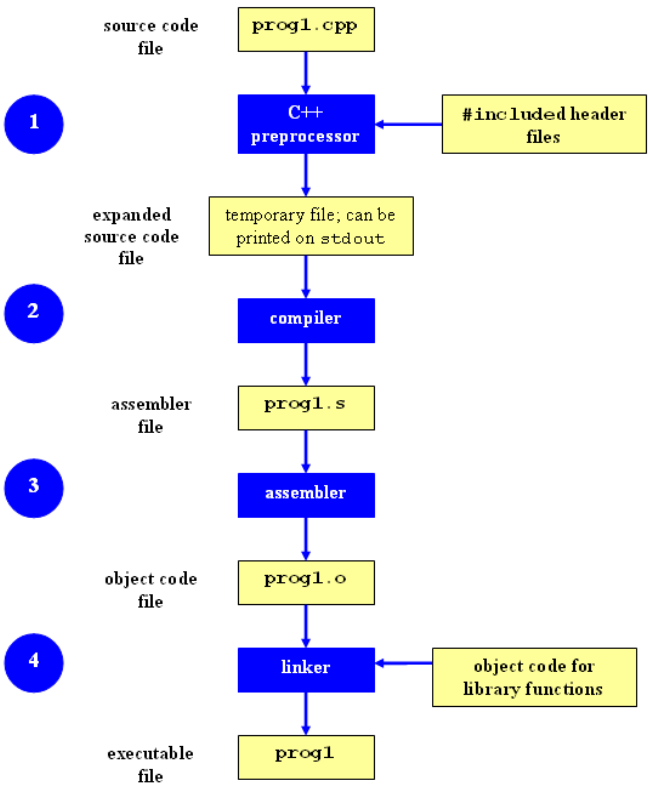
2 : directive interpreted by prepocessor. include a section of standard C++ code : header iostream

4 : declaration of a function. main is called when the program is run.

5,7 : braces to indicate the beginning and the end of the definition

6 : statement. standard character output device + insertion operator + argument + semicolon (end of statement)

Compilation



The C++ preprocessor copies the contents of the included header files into the source code file, generates macro code, and replaces symbolic constants defined using #define with their values. -E

The expanded source code file produced by the C++ preprocessor is compiled into the assembly language for the platform. -S

The assembler code generated by the compiler is assembled into the object code for the platform. -c

The object code file generated by the assembler is linked together with the object code files for any library functions used to produce an executable file.

Classes

classes expand the concept of *data structures*

- contain data members (attributes)
- also contain functions as members (methods)

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

- private members of a class are accessible only from within other members of the same class (or from their "friends")
- protected members are accessible from other members of the same class (or from their "friends"), but also from members of their derived classes.
- public members are accessible from anywhere where the object is visible.

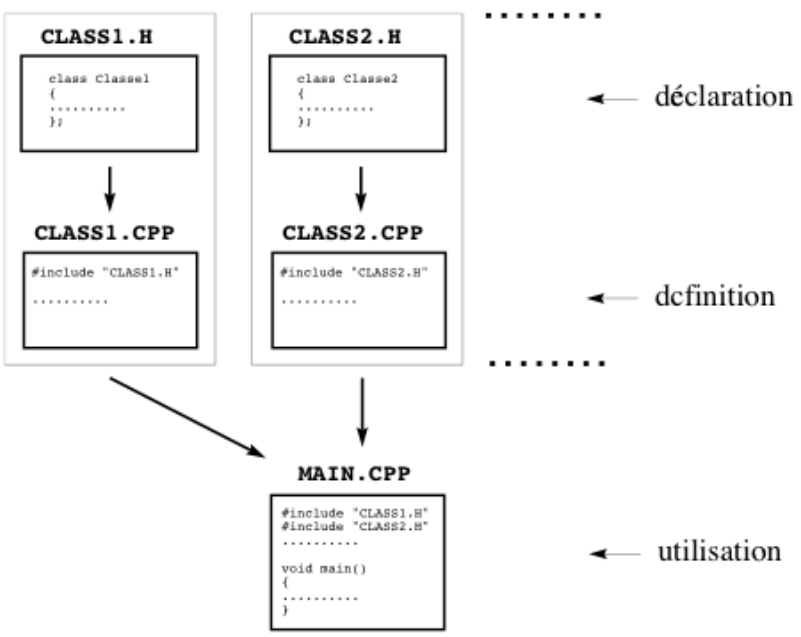
Classes

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;
}
```

ex : perimeter

Classes



Classes : constructor

http://cpp.sh/8lr

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

Overloading constructors

Like any other function, a constructor can also be overloaded with different versions taking different parameters: with a different number of parameters and/or parameters of different types. The compiler will automatically call the one whose parameters match the arguments.

ex : http://cpp.sh/8lr -> Rectangle();

Member initialization in constructors

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

The constructor for this class could be defined, as usual, as:

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

But it could also be defined using member initialization as:

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

Or even:

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

Using classes

```
1  // member initialization
2  #include <iostream>
3  using namespace std;
4
5  class Circle {
6      double radius;
7    public:
8      Circle(double r) : radius(r) { }
9      double area() {return radius*radius*3.14159265;}
10 };
11
12 class Cylinder {
13     Circle base;
14     double height;
15   public:
16     Cylinder(double r, double h) : base (r), height(h) {}
17     double volume() {return base.area() * height;}
18 };
19
20 int main () {
21   Cylinder foo (10,20);
22
23   cout << "foo's volume: " << foo.volume() << '\n';
24   return 0;
25 }
```


Overloading operators

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

```
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 };
```

```
1 c = a + b;
2 c = a.operator+ (b);
```

& : actual variable itself, rather than a copy, is used as the parameter in the subroutine

const : cannot be altered by the program

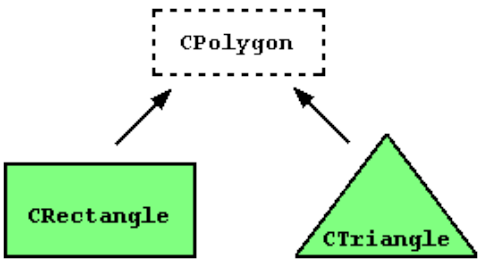
Overloading operators

http://cpp.sh/27l

```
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 }
```


Inheritance

Classes in C++ can be extended, creating new classes which retain characteristics of the base class. This process, known as inheritance, involves a base class and a derived class: The derived class inherits the members of the base class, on top of which it can add its own members.



http://cpp.sh/9m2

This public keyword after the colon (:) denotes the maximum accessible level the members inherited from the class that follows it (in this case Polygon) will have in the derived class (in this case Rectangle).

```
// 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;
}
```


Function template

http://cpp.sh/4jq

Same syntax as regular function and preceded by template keyword and a series of template parameters enclosed by <>

```
// 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;
}
```

source : http://www.cplusplus.com/doc/tutorial/