CLASS SPECIFICATION AND OBJECTS IN CPP

Classes

Classes are created using the keyword **class**. A class declaration defines a new type that links code and data. This new type is then used to declare objects of that class. Thus, a class is a logical abstraction, but an object has physical existence. In other words, an object is an *instance* of a class. A class declaration is similar syntactically to a structure. A simplified general form of a class declaration was shown. Here is the entire general form of a **class** declaration that does not inherit any other class.

class *class-name {*

private data and functions

access-specifier:

data and functions

access-specifier:

data and functions

// ...

access-specifier:

data and functions

} object-list;

The *object-list* is optional. If present, it declares objects of the class. Here, *access-specifier*is one of these three C++ keywords:

public

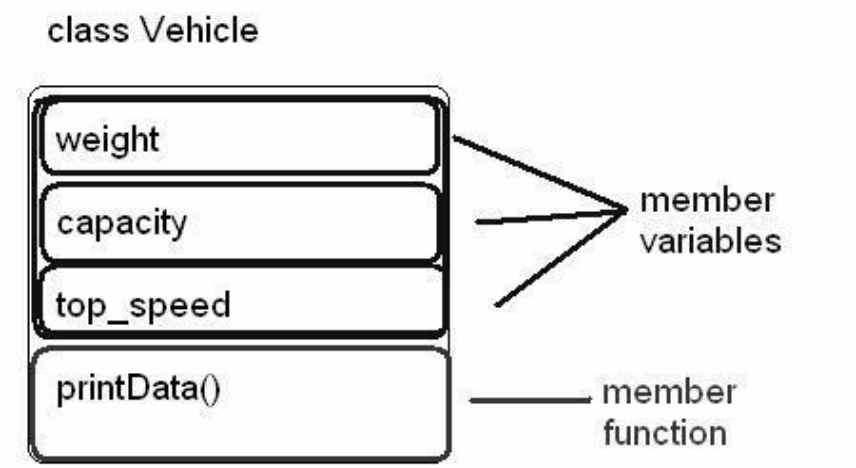
private

protected

2.2 Class Objects

Classes and objects

C++ has classes **A class** is a user-defined type. The variables of this type are **objects.** A class can be obtained from a structure if some member functions are added.



Member data and member functions can both be accessed using variable-to-member access operator Each object will have separate separate copy of the member data within itself but only one copy of member function exists

Private and public

A member (variable or function) can be **private** or **public** The keywords are also known as access modifiers or access specifiers a "good" class keeps its member variables private = **data hiding** and uses public member functions to access or change each private variable. Class members are private by default whereas struct members are public Objects Variables of classes are known as objects.

2.3 Scope resolution operator

The Scope Resolution Operator

As you know, the **::** operator links a class name with a member name in order to tell the compiler what class the member belongs to. However, the scope resolution operator has another related use: it can allow access to a name in an enclosing scope that is "hidden" by a local declaration of the same name.

For example, consider this fragment:

```
int i; // global i
void f()
```

```
{
int i; // local i
i = 10; // uses local i
.
.
.
}
```

As the comment suggests, the assignment **i = 10** refers to the local **i**. But what if function **f( )** needs to access the global version of **i**? It may do so by preceding the **i** with the **::** operator, as shown here.

```
int i; // global i
void f()
{
int i; // local i
::i = 10; // now refers to global i
.
.
.
}
```

2.4 Access members

The arrow operator is used to access members of the object. Here is a short program that creates a class called **balance** that links a person's name with his or her account balance. Inside **main( )**, an object of type **balance** is created dynamically.

```
#include <iostream>
#include <new>
#include <cstring>
using namespace std;
class balance {
double cur_bal;
char name[80];
```

```
public:
void set(double n, char *s) {
cur_bal = n;
strcpy(name, s);
}
void get_bal(double &n, char *s) {
n = cur_bal;
strcpy(s, name);
}
};
int main()
{
balance *p;
char s[80];
double n;
try {
p = new balance;
} catch (bad_alloc xa) {
cout << "Allocation Failure\n";
return 1;
}
p->set(12387.87, "Ralph Wilson");
p->get_bal(n, s);
cout << s << "'s balance is: " << n;
cout << "\n";
delete p;
return 0;
}
```

Because **p** contains a pointer to an object, the arrow operator is used to access members of the object.

Source : http://elearningatria.files.wordpress.com/2013/10/cse-iii-object-oriented-programming-with-c-10cs36-notes.pdf