C++ and Object Oriented Programming Refresh

C++ is a superset of C.

- What you know from C will also apply in C++.
- C++ is a object oriented programming (OOP) language
- OOP is programming paradigm that uses "object" - data structures consisting of data members and functions.
- OOP includes techniques for:
  - Data abstraction, encapsulation, modularity
  - Polymorphism
  - Inheritance


```
#include <iostream>

int main(int argc, char *argv[])
{
    std::cout << "hello world" << std::endl;
    return 0;
}
```

Objects and Classes

- **Classes** are the blue-print for objects. Defining the functionality and properties of fully instantiated objects.
- **Functions** may defined to do computations
- **Objects** are *instances* of classes - representing a self-contained module with data and functions.

animal.h

```
class Animal
{
public:
    Animal(int age) : m_age(age)
    {
    }

    int age() const
    {
        return m_age;
    }

private:
    int m_age;
};
```

main.cpp

```
#include <iostream>
#include "animal.h"

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

int main()
{
    // Creating a new animal (on stack)
    Animal someAnimal(5);
    std::cout << someAnimal.age() << std::endl;

    // On heap
    Animal *otherAnimal = new Animal(19);
    std::cout << otherAnimal->age() << std::endl;

    delete otherAnimal;
    return 0;
}
```


Data Abstraction and Encapsulation

- We, the users of the Animal class/object
  - Don't have to worry about the implementation details an Animal.
  - We only are concerned with the Animal ***interface***.
- We may *extend* the Animal class to provide additional functionality.

Objects and Classes

- **Classes** are typically ***declared*** in a .h file
- **Classes** are typically ***defined*** in a .cpp file

animal.h

```
class Animal
{
public:
    Animal(int age);

    int age() const;

private:
    int m_age;
};
```

animal.cpp

```
#include "animal.h"

Animal::Animal(int age) : m_age(age)
{
}

int Animal::age() const
{
    return m_age;
}
```


Objects and Classes

- **Objects** are created on the **stack** or on the **heap**.
- Stack objects are created by invoking the constructor.
  - Dies automatically
- Heap objects are created invoking the ***new*** operator and constructor.
  - Must be ***deleted***

main.cpp

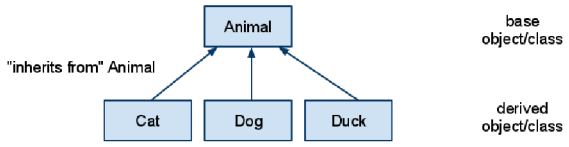
```
#include <iostream>
#include "animal.h"
int main()
{
    // Creating a new animal (on stack)
    Animal someAnimal(5);
    std::cout << someAnimal.age() << std::endl;

    // On heap
    Animal *otherAnimal = new Animal(19);
    std::cout << otherAnimal->age() << std::endl;

    delete otherAnimal;
    return 0;
}
```

OOP: Interitance

- Certain classes may use the same members and functions. I.e the Cat, Dog, and Duck class may all use the Animal class functionality.
- This relationship can be arranged in a hierarchy:



- We also say "a cat object **is an** animal"
- We can now efficiently reuse our code!

Inheritance Example

- A Cat **is an** Animal

Inheritance access specifier:

- **Public**: all public members/functions are available inside and outside the derived class
- **Protected**: public and protected parts of base is accessible in derived classes/objects
- **Private**: public protected parts are private in base class are private in derived

```
class Animal
{
public:
    Animal(int age) : m_age(age)
    {
    }

    int age() const
    {
        return m_age;
    }

private:
    int m_age;
};
```

```
class Cat : public Animal
{
public:
    Cat(int age) : Animal(age)
    {
    }

    int calculateCatYears()
    {
        // Cannot access m_age directly
        return age()*humanToCatFactor;
    }
};
```


Inheritance Example

- A Cat **is an** Animal

Inheritance access specifier:

- **Public**: all public members/functions are available inside and outside the derived class
- **Protected**: public and protected parts of base is accessible in derived classes/objects
- **Private**: public protected parts are private in base class are private in derived

```
class Animal
{
public:
    Animal(int age) : m_age(age)
    {
    }

    int age() const
    {
        return m_age;
    }

private:
    int m_age;
};
```

```
class Cat : public Animal
{
public:
    Cat(int age) : Animal(age)
    {
    }

    int calculateCatYears()
    {
        // Cannot access m_age directly
        return age()*humanToCatFactor;
    }
};
```

To make age available in derived classes, we must make it ***public*** or ***protected***.

Resources

Here are a couple of links for C++ material:

- http://www.cplusplus.com/doc/tutorial/
- http://www.parashift.com/c++-faq-lite

If you use IRC, you can also find help on #c++ on EFNet or similar.

Also you can ask questions on this website: stackoverflow.com