C++ and Object Oriented Programming Refresh

Click to add text

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

Objects and Classes

- Classes are the blue-print for objects. Defining the functionality and properties of fully instantiated objects.
- 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"
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.

OOP: Interitance

- Certain classes may use the same members and functions. I.e the Cat, Dog, and Duck class may all use the Animal

Inheritance

When a class is inherited all functions and data members are inherited

- although *not* all of them will be accessible in the derived class.

Access specifiers:

- **Public:** accessible everywhere
- **Protected:** accessible in derived classes/objects
- **Private:** accessible only inside an class/object

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

    int age() const
    {
        return m_age;
    }

private:
    int m_age;
};
```

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

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

    int age() const
    {
        return m_age;
    }

protected:
    int m_age;
};
```

Inheritance Example

- A Cat **is an** Animal

Inheritance access specifier:

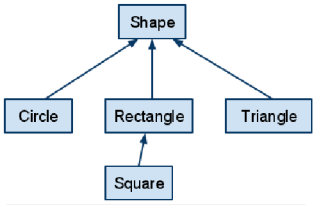
- Public: all public members/functions are available inside and outside the derived class
- Protected: public and protected parts of base is accessible in derived classes/objects
- Private: public protected parts are private in base class are private in derived

```
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;
    }
};
```


Polymorphism

- Base class Shape establishes the common interface to classes deriving from Shape
- Polymorphism allows the programmer to implement different *draw* methods for derived classes
- The **virtual** keyword is used to allow polymorphism
- Shape is an abstract class.




```
class Shape
{
public:
    virtual void draw() = 0;
    virtual double calculateArea();
};

class Rectangle : public Shape
{
public:
    void draw()
    { ... }
};
```

```
Shape *s1 = new Rectangle();
s1->draw();
Shape *s2 = new Circle();
s2->draw();
```