C++ Pointers and Objects

Outline Structs and Classes

- structs
- With member functions
- With protection
- Classes
- Member functions
- Initialization
- Default Constructor
- More to follow

Revisit Structs

```
struct studentInfo{
    std::string first;
    std::string last;
    int         age;
};
```

- Great way to aggregate data in 1 place
- But…

1. What if not initialized, or only partially initialized?
2. How do you tell if its correctly initialized?
3. How do you guard against modification?

Structs – Start Encapsulation

```
struct StudentInfo {
    std::string name;
    double midterm;
    double final;

    void    read();         //initialize name midterm and final
    void    write() const;  //output data

    double  grade() const;  //calculate final grade
};
```

- Interface in studentinfo.h, (implementation next)
- Data members wrapped with functions
- const – we will not modify internal members

Structs – Start Encapsulation

```
void    StudentInfo::read()
{
    cout<<"enter name";
    cin>>name;

    cout<<"enter midterm";
    cin>>midterm;

    cout<<"enter final";
    cin>>final;
}

void    StudentInfo::write() const
{
    cout<<"Name="<<name<<" Grade is"<<grade();
}
double  StudentInfo::grade() const
{
    return (midterm*.5 + final*.5);
}
```

- Implementation studentinfo.cpp
- Notice StudentInfo::
- Don’t need to pass members vars to functions
- But all data members still public. Can modify outside of function calls.

Structs – member access

```
#ifndef STUDENTINFO_H_
#define STUDENTINFO_H_

struct StudentInfo {

private:
    std::string name;
    double midterm;
    double final;
public:
    void   read();          //initialize name midterm and final
    void   write() const;   //output data

    double  grade() const;  //calculate final grade
};

#endif /* STUDENTINFO_H_ */
```

Structs are pretty much Classes

```
#ifndef STUDENTINFO_H_
#define STUDENTINFO_H_

class StudentInfo {

private:
    std::string name;
    double midterm;
    double final;
public:
    void    read();         //initialize name midterm and final
    void    write() const;  //output data

    double  grade() const;  //calculate final grade
};

#endif /* STUDENTINFO_H_ */
```

Classes – member access

- public: access is granted to all
- protected: accessible in the class that defines them and in classes that inherit from that class.
- private: only accessible within the class defining them

```
class AClass
{
public:
    //this is a default constructor, compiler writes
    //it for you IFF your class has no other constructors
    AClass();

    //destructor, note the ~,
    ~AClass(void);

protected:
    void protected_method();

private:
    int i;
};
```

Classes – constructors and destructors

- Constructors
  - Default
  - With parameters
  - Copy
- Destructors

```
#pragma once
class AClass
{
public:
    //this is a default constructor, compiler writes
    //it for you IFF your class has no other constructors
    AClass();

    //one param constructor
    AClass(int val);

    //this is copy constructor
    AClass(const AClass& other);

    //destructor, note the ~,
    ~AClass(void);

private:
    int i;
};
```

```
{                       //usage
    AClass ac1;         //default constructor
    AClass ac2(ac1);    //copy constructor
}                       //ac1 and ac2 destructors called
```

Classes – What goes in public interface

- Should be complete and MINIMAL
  - No rubbish getters and setters without a good reason!
  - Remember the smaller the public interface the easier it is to understand the object

- Should a function be part of a class?
  - General Rule: If a function changes the state of an object then it should be a member of the class.

Classes – What goes in, What stays out

- Compare(…) function: no state change, can be external

```
#ifndef STUDENTINFO_H_
#define STUDENTINFO_H_

class  StudentInfo {

private:
    std::string name;
    double midterm;
    double final;
public:
    void    read();         //initialize name midterm
    void    write() const;  //output data

    double  grade() const;  //calculate final grade

    std::string getname() const { return name; }
};

bool compare(const StudentInfo&, const StudentInfo&);

#endif /* STUDENTINFO_H_ */
```

```
void    StudentInfo::read()
{
    cout<<"enter name";
    cin>>name;

    cout<<"enter midterm";
    cin>>midterm;

    cout<<"enter final";
    cin>>final;
}

void    StudentInfo::write() const
{
    cout<<"Name="<<name<<" Grade is"<<grade();
}
double  StudentInfo::grade() const
{
    return (midterm*.5 + final*.5);
}

bool compare(const StudentInfo& x, const StudentInfo& y)
{
    return x.getname() < y.getname();
}
```

Classes – Initialized?

- Still not there yet, object members are initialized, that does not mean they are valid
- Add a isValid function

.h file

```
class  StudentInfo {

private:
    std::string name;
    double midterm;
    double final;
    bool bIsValid;
public:
    //default constructor
    StudentInfo();          // construct an empty
    bool    isValid() const {return bIsValid;}
    void    read();         //initialize name midt
```

.cpp file

```
void    StudentInfo::read()
{
    cout<<"enter name";
    cin>>name;

    cout<<"enter midterm";
    cin>>midterm;

    cout<<"enter final";
    cin>>final;
    bIsValid = true;
}
StudentInfo::StudentInfo(): midterm(0), final(0),bIsValid(false) {
```

Classes – Default Constructor

- Still not there yet, object is in an undefined state until read() initializes it

Use constructors to set state

Default: no arguments (.h file)

```
//default constructor
StudentInfo();          // construct an empty `Student_info' object
```

Still Uninitialized: Fix with initializer list (.cpp file)

```
StudentInfo::StudentInfo(): midterm(0), final(0),bIsValid(false) { }
```

Summary so far

- Structs have default public scope
- Classes have default private scope
- Structs are pretty much classes
- Constructors – provide one or the compiler will (and it will be wrong)
- Destructors – always called, do clean up of mem, handles, network connections etc. here)