CS 162
Intro to Programming II

Classes
Encapsulation

• The **interface** (is in the class) is separate from the **implementation** (how the functions work)

• Changing the implementation should not change the interface
  – Any using your object will not be affected
Example- Interface

• student.hpp
  
    class student {
    public:
        void print();
        std::string name;
    
    private:
        int id;
    }
  
• Public first as that is all a user can see
Example- Implementation

• student.cpp

```cpp
#include <iostream>
#include "student.hpp"

int main(int argc, char** argv)
{
    student s;
    s.print();
}

void student::print() {
    std::cout << "Student name: " << name << std::endl;
    std::cout << "Student id: " << id << std::endl;
}
```

OOP

• Class is the code ‘pattern’
• Object is the instantiation of the pattern
• Analogy
  – Class is like a cookie cutter
  – Object is like a cookie
Accessing Members

• **Dot operator** eg. `s.print()` says that `print()` is a member of object `s`
  – Used with the object name (ie. `s`)

• **Scope resolution operator** eg. `student::print()` says `print()` is a member of class `student`
  – Used with the class name (ie. `student`)
  – Used inside the class definition
Access Specifiers

- **Private**: can only reference this method/variable within the class
  - Member variables are usually private
  - Why?
- **Public**: can reference this method/variable anywhere
  - Allows other objects of code to use your object
int main(int argc, char** argv) {
    student s;  //note a
    s.print();
}

• Note a- this creates (instantiates) an object of class student

• There is no constructor in student.cpp

• A default constructor is provided
Constructor- continued

• Automatically called when an object of that class is declared
• Initializes member variables and other things related to the object
• Instantiation is assigning system resources
• The default constructor only instantiates the object
  – All data members get default values for their data type
Rules

• Must have same name as the class
• Cannot have a return value
• Constructors are public
• If you don’t define any constructors, C++ will automatically create a default constructor for you that leaves blank member variables

• TIP: Always define a constructor.
  – leaving member variables uninitialize leads to lots of errors
Constructors- student.cpp

```cpp
student::student() {
    id = -1;
    name = NULL;
}
student::student(string nameValue) {
    id = -1;
    name = nameValue;
}
student::student(int idValue, string nameValue) {
    id = idValue;
    name = nameValue;
}
```
Constructors- student.hpp

class student { 
public: 
    student(); 
    student(string name); 
    student(int id, string name); 
    void print(); 
    string name;

private: 
    int id;
};
Constructors- student.cpp

• Alternately, use an initialization list
  
  ```cpp
  student::student() : id(-1), name(NULL) {
  }
  
  student::student(string nameValue) : id(-1),
  name(nameValue) {
  }
  
  student::student(int idValue, string
  nameValue) : id(idValue), name(nameValue) {
  }
  ```
Calling Constructors

• Declare a variable of type student
  
  student s(“Bob”);

• Explicit call
  
  s = student(1,”Adam”);

• Memory allocation
  
  student* s = new 
  student(2,”Eve”);
Accessors/Mutators

• Accessors (example)

```cpp
int student::get_id() {
    return id;
}
string student::get_name() {
    return name;
}
```

• Mutators (example)

```cpp
void student::set_name(string newname) {
    name = newname;
}
```

• Neither is required
Accessors/Mutators

• Header file is now:

class student {
    public:
        student();
        student(string name);
        student(int id, string name);
        int get_id();
        string get_name();
        void set_name(string newname);
        void print();
    private:
        int id;
        char* name;
};
Inline Functions

• Useful for short functions
• Textual substitution, similar to #include
  – Code for the function is inserted where the function call was located
• Saves the overhead of a function call
Example

class student {
    public:
        int get_id() { return id; }
        string get_name() { return name; }
    // there is no semi-colon after the brace

    private:
        int id;
        char* name;
}