Final Review

Final Details

- **Saturday (yes, Saturday) December 13th at 11AM**

- Location: (Based on Last Name)
  - **A-J**: SGM 101
  - **K-Z**: SGM 123
- Format: T/F, multiple choice, short answer, FiTB coding, coding snippets

Topics

In no particular order

- Basic C++ syntax, control flow
- Functions - pass by value
- Arrays, multidimensional arrays
- Images
- Pointers
- C-strings
- Pass-by-pointer
- Pass-by-C++-reference
- Pointer Arithmetic/arrays + pointers/arrays of pointers
- Dynamic multi-dimensional arrays
- Command line arguments
- Dynamic memory
- Deep/shallow copy
- C++ strings
- fstreams, stringstreams
- Basic objects: syntax, constructors/destructors
- Vectors/deques/STL
- Linked Lists
- Operator overloading
- Objects: Inheritance/Polymorphism
- Copy/Assignment semantics
- Exceptions
- Recursion
- Runtime (Big-O)

REVIEW

Review [1]

Const function arguments

- Will this code compile?
- Indicate what will be printed (assuming it compiles)

```
void f1(const vector<int>& x){
  x.push_back(103);
  x.push_back(104);
}

void f2(string& y){
  y = "Bye";
}

int main()
{
   vector<int> a; string b = "Hi";
   f1(a);
   f2(b);
   cout << b.size() << endl;
   return 0;
}
```

Const member functions

- What does the highlighted const keyword imply in the code below?

```
class Item
{ int val;
 public:
  void foo();
  int bar() const;
};

void Item::foo()
{ val = 5; }

int Item::bar() const
{ return val+1; }

void f1(const Item& arg) {
  int x = arg.bar(); // fine
  arg.foo(); // Compiler Error!
}
```

Review [2]

Constructor Initialization Lists

- What is the most efficient means to initialize the vals member to an initial array size of 20 and s to a user-defined argument?

```
class Thing {
 public:
  Thing(const std::string& s_init);
 private:
  vector<int> vals;
  string s;
};

Thing::Thing(const std::string& s_init)

{
   // is this the most efficient way?
   vals.resize(20);
   s = s_init;
}
```

Construction Order

- What is printed by the code below?

```
class ABC {
 public:
  ABC() { cout << "ABC" << endl; }
};
class DEF {
 public:
  DEF() { cout << "DEF" << endl; }
};
class XYZ {
  ABC m1;  DEF m2;
 public:
  XYZ()
    { cout << "XYZ" << endl; }
};
int main() {
  XYZ x1;
  return 0;
}
```

Review [3]

Friend Functions

- What does the highlighted `friend` keyword imply in the code below?
- What would break if we remove it?

```
class Complex
{
 public:
  Complex();
  Complex(double r, double i);
  friend Complex operator+(const int&, const Complex&);
 private:
  double real, imag;
};

Complex operator+(const int& lhs, const Complex &rhs)
{
  Complex temp;
  temp.real = lhs + rhs.real;   temp.imag = rhs.imag;
  return temp;
}
```

Friend Classes

- Can DEF::clear() access obj.x?
- If not, how can class ABC grant access to DEF?

```
class ABC {
 int x;  // data member
 public:

   ...
};

class DEF {
public:
  void clear(ABC& obj) { obj.x = 0; }
};
```

SOLUTIONS

Identify that Constructor

- Prototype what constructors are being called here
- s1
  - Student::Student()
    // default constructor
- s2
  - Student::Student(string, int) or
    Student::Student(const char*, int)
- dat
  - vector<int>::vector<int>(int);

```
class Student {
 public:
  // Default constructor
  Student( );

  // Initializing constructor
  Student(const string& name);
  ...
 private:
   string name_;
   int id_;
   vector<int> grades_;
};

int main()
{
  Student s1;
  Student s2("Tommy", 12345);

  vector<int> vals(10);
  ...
}
```

Review [1] Solutions

Const function arguments

- Will this code compile? **No, modification of x in f1()**
- Indicate what will be printed (assuming it compiles) – b.size() will be 3

```
void f1(const vector<int>& x){
  x.push_back(103);
  x.push_back(104);
}

void f2(string& y){
  y = "Bye";
}

int main()
{
   vector<int> a; string b = "Hi";
   f1(a);
   f2(b);
   cout << b.size() << endl;
   return 0;
}
```

Const member functions

- What does the highlighted const keyword imply in the code below?
  - No data members can be modified nor non-const member functions called

```
class Item
{ int val;
 public:
  void foo();
  int bar() const;
};

void Item::foo()
{ val = 5; }

int Item::bar() const
{ return val+1; }
```

Review [2] Solutions

Constructor Initialization Lists

- What is the most efficient means to initialize the vals member to an initial array size of 20 and s to a user-defined argument?

```
class Thing {
 public:
  Thing(const std::string& s_init);
 private:
  vector<int> vals;
  string s;
};

Thing::Thing(const std::string& s_init)
  : vals(20), s(s_init)
{



}
```

Construction Order

- What is printed by the code below?
  - **ABC**
    DEF
    XYZ

```
class ABC {
 public:
  ABC() { cout << "ABC" << endl; }
};
class DEF {
 public:
  DEF() { cout << "DEF" << endl; }
};
class XYZ {
  ABC m1;  DEF m2;
 public:
  XYZ() { cout << "XYZ" << endl; }
};
int main() {
  XYZ x1;
  return 0;
}
```

Review [3] Solutions

Friend Functions

- What does the highlighted `friend` keyword imply in the code below?
  - That function can access Complex private members
- What would break if we remove it?
  - Could not access rhs.real / rhs.imag

```
class Complex
{
 public:
  Complex();
  Complex(double r, double i);
  friend Complex operator+(const int&, const Complex&);
 private:
  double real, imag;
};

Complex operator+(const int& lhs, const Complex &rhs)
{
  Complex temp;
  temp.real = lhs + rhs.real;   temp.imag = rhs.imag;
  return temp;
}
```

Friend Classes

- Can DEF::clear() access obj.x? No
- If not, how can class ABC grant access to DEF?
  - Add friend definition

```
class ABC {
 int x;   // data member
 public:
  friend class DEF;
  ...
};

class DEF {
public:
  void clear(ABC& obj) { obj.x = 0; }
};
```

OPERATOR OVERLOADING REVIEW

Operator Overloading Review

Member or Non-member?

- How do you decide if you can make the operator overload function a member function of the class?
- When do you have to use a non-member operator function?

```
// arbitrary precision integer class
class BigInt {
  ...
};
int main(){
  BigInt x, y, z;
  x = y + 5;
}
```

Arguments

- For member function operator overloads, how many input arguments are needed for operator+? For operator! ?

```
// arbitrary precision integer class
class BigInt {
  ____ operator+(                 );
  ____ operator!(                 );
};
int main(){
  BigInt w, x, y, z;
  w = x + y;
  bool flag = !w;
}
```

Operator Overloading Review

Return types

- For class BigInt which models an arbitrary precision integer, what should the return type be for:
  - Operator+
  - Operator==

```
class BigInt {
 public:
   _________ operator+(const BigInt&);
   _________ operator==(const BigInt&);
};
int main(){
  BigInt w, x, y, z;
  w = x + y;
}
```

Chaining

- Do we need operator overload functions with 2-, 3-, 4-inputs, etc. to handle various use cases?

```
class BigInt {
  ...
};
int main(){
  BigInt w, x, y, z;
  w = x + y + z;
  cout << w << " is a bigint!" << endl;
}
```

SOLUTION

Operator Overloading Review

Member or Non-member?

- How do you decide if you can make the operator overload function a member function of the class?
  - If the left-hand side operand is a class instance
- When do you have to use a non-member operator function?
  - If the left operand of an operator is NOT an instance of the class, you cannot use a member function

```
// arbitrary precision integer class
class BigInt {
  ...
};
int main(){
  BigInt x, y, z;
  x = y + 5;
}
```

Arguments

- For member function operator overloads, how many input arguments are needed for operator+?
  - Only 1, the left side operand is 'this'
- for operator!
  - None, only operand is 'this'

```
// arbitrary precision integer class
class BigInt {
  ____ operator+(const BigInt& rhs);
  ____ operator!();
};
int main(){
  BigInt w, x, y, z;
  w = x + y;
  bool flag = !w;
}
```

Operator Overloading Review

Return types

- For class BigInt which models an arbitrary precision integer, what should the return type be for:
  - Operator+: BigInt (by value)
  - Operator==: bool

```
class BigInt {
 public:
   BigInt operator+(const BigInt&);
   bool operator==(const BigInt&);
};
int main(){
  BigInt w, x, y, z;
  w = x + y;
}
```

Chaining

- Do we need operator overload functions with 2-, 3-, 4-inputs, etc. to handle various use cases?
  - No, this is why the return type should be BigInt to allow for chaining: x.operator+(y).operator+(z), etc.

```
// arbitrary precision integer class
class BigInt {
  ...
};
int main(){
  BigInt w, x, y, z;
  w = x + y + z;
  cout << w << " is a bigint!" << endl;
}
```

REVIEW

Review [1]

- What is the correct prototype for the copy constructor call when c3 is created in the code to the right?
  - **`Complex(Complex);`**
  - `Complex(Complex &)`
  - **`Complex(const Complex &)`**

```
class Complex
{
 public:
  Complex();
  Complex(double r, double i);

  // What constructor definition do I
  // need for c3's declaration below

 private:
  double real, imag;
};

int main()
{
  Complex c1(2,3), c2(4,5)
  Complex c3(c1);


}
```

Review [2]

Which function?

- For each of the following, identify whether the copy constructor is called or the assignment operator
  - `Complex c1;`
    `Complex c2 = c1;`
  - `Complex c1;`
    `Complex c2(c1);`
  - `Complex c1, c2;`
    `c2 = c1;`

Default Versions

- What kind of copy does the default copy constructor and assignment operator perform?

```
class MyArray
{
  ...
 private:
  int* data; // ptr to dynamic array
  size_t len;
};
```

Review [3]

State the Rule of 3

- The rule of 3:

Assignment Operator Specifics?

- What extra considerations does the assignment operator need to handle vs. the copy constructor?
- What should operator= return?

```
class MyArray
{



 private:
  int* data; // ptr to dynamic array
};

MyArray& operator=(const MyArray& other)
{



}
```

SOLUTIONS

Review [1]

- What is the correct prototype for the copy constructor call when c3 is created in the code to the right?
  - **`Complex(Complex);`**
    - **We will see that this can't be right…**
  - `Complex(Complex &)`
    - **Possible**
  - **`Complex(const Complex &)`**
    - **`Best! (Making a copy shouldn't change the input argument, thus 'const')`**

```
class Complex
{
 public:
  Complex();
  Complex(double r, double i);

  // What constructor definition do I
  // need for c3's declaration below

 private:
  double real, imag;
};

int main()
{
  Complex c1(2,3), c2(4,5)
  Complex c3(c1);


}
```

Review [2]

Which function?

- For each of the following, identify whether the copy constructor is called or the assignment operator
  - `Complex c1;`
`Complex c2 = c1;`
    - `Copy constructor`
  - `Complex c1;`
`Complex c2(c1);`
    - `Copy constructor`
  - `Complex c1, c2;`
`c2 = c1;`
    - `Assignment operator`

Default Versions

- What kind of copy does the default copy constructor and assignment operator perform?
  - Shallow copy (member by member copy)

```
class MyArray
{
   ...
 private:
  int* data; // ptr to dynamic array
  size_t len;
};
```

Review [3]

State the Rule of 3

- The rule of 3:
  - If a class needs a user-defined version of any one of the 3: copy constructor, assignment operator, or destructor, it needs ALL 3.

```
class MyArray
{



 private:
  int* data; // ptr to dynamic array
};

MyArray& operator=(const MyArray& other)
{



}
```

Assignment Operator Specifics?

- What extra considerations does the assignment operator need to handle vs. the copy constructor?
  - Must clean up old resources before copying
  - Beware of self assignment
- What should operator= return?
  - A reference to an instance of the class which should be *this;

REVIEW QUESTIONS

Inheritance Review 1

- **T/F**: A student object has a name_ and id_ member
- **T/F**: Code from the Student class can access name_ and id_
  - What could you change to flip the T/F answer?
- What would change if Student inherited Person through private inheritance?

```
class Person {
 public:
  Person(string n, int ident);
  string get_name();
  int get_id();
 private:
  string name_; int id_;
};

class Student : public Person {
 public:
  Student(string n, int ident, int mjr);
  int get_major();
  double get_gpa();
  void set_gpa(double new_gpa);
 private:
  int major_; double gpa_;
};
int main()
{
  Student s1("Amanda", 12345, 1);
  cout << s1.get_name() << endl;
  return 0;
}
```

Inheritance Review 2

- Inheritance defines an ________ relationship between classes
- Composition defines a __________ relationship between two objects
- Protected access makes members accessible to _____________ but still not to _____________________

SOLUTIONS

Inheritance Review 1

- T/F: A student object has a name_ and id_ member
- T/F: Code from the Student class can access name_ and id_
  - What could you change to flip the T/F answer? Changing Person's access specifier to protected or public. Regardless of how Student inherits, name_ and id_ will be private to the Student class.
- What would change if Student inherited Person through private inheritance?
  - External clients (like main) would not be able to access the inherited members (from Person) of a Student object.

```
class Person {
 public:
  Person(string n, int ident);
  string get_name();
  int get_id();
 private:
  string name_; int id_;
};

class Student : public Person {
 public:
  Student(string n, int ident, int mjr);
  int get_major();
  double get_gpa();
  void set_gpa(double new_gpa);
 private:
  int major_; double gpa_;
};
int main()
{
  Student s1("Amanda", 12345, 1);
  cout << s1.get_name() << endl;
  return 0;
}
```

Inheritance Review 2

- Inheritance defines an is-a relationship between classes
- Composition defines a has-a relationship between two objects
- Protected access makes members accessible to a derived/child class but still not to external/3rd-party clients

Review Questions 1

- As we call processPerson(&p) what member functions will be called (e.g. Person::print_info, CSStudent::useComputer, etc.)
- As we call processPerson(&s)?
- As we call processPerson(&cs)?
- We use the terms static and dynamic binding when referring to which function will be called when virtual is NOT or IS present.

```
class Person {
 public:
  virtual void print_info() const; // name, ID
  void useComputer(); // stream a show
  string name; int id;
};
class Student : public Person {
 public:
  void print_info() const; // print major
  void useComputer(); // write a paper
  int major; double gpa;
};
class CSStudent : public Student {
 public:
  void print_info() const; // print OH queue pos
  void useComputer(); // fight with Codio
};

void processPerson(Person* p)
{ p->print_info();
  p->useComputer(); }

int main(){
  Person p(...);     processPerson(&p);
  Student s(...);    processPerson(&s);
  CSStudent cs(...); processPerson(&cs);
  // more
}
```

Review Questions 2

- What does "=0;" mean in the declarations to the right?
- What do we call a class with 1 or more of these kind of declarations?
- Is it okay that Student doesn't provide a useComputer() implementation?
- Can we declare Person objects?
- Can we declare pointers or references to Person objects?
- When should a class have a virtual destructor?

```
class Person {
 public:
  virtual void print_info() const = 0;
  virtual void useComputer(); // stream a show
  string name; int id;
};
class Student : public Person {
 public:
  void print_info() const; // print major
  int major; double gpa;
};
class CSStudent : public Student {
 public:
  void print_info() const; // print OH queue pos
  void useComputer(); // fight with Docker
};

void printPerson(Person* p) { p->print_info(); }
void compute(Person& p)     {  p.useComputer(); }

int main(){
  Person p(...);  // Allowed?
  Student s(...);    useComputer(s);
  CSStudent cs(...); printPerson(&cs);
  // more
}
```

SOLUTIONS

Review Questions 1

- As we call processPerson(&p) what member functions will be called (e.g. Person::print_info, CSStudent::useComputer, etc.)
  - Person::print_info() / Person::useComputer()
- As we call processPerson(&s)?
  - Student::print_info() / Person::useComputer()
- As we call processPerson(&cs)?
  - CSStudent::print_info() / Person::useComputer()
- We use the terms static and dynamic binding when referring to which function will be called when virtual is NOT or IS present.

```
class Person {
 public:
  virtual void print_info() const; // name, ID
  void useComputer(); // stream a show
  string name; int id;
};
class Student : public Person {
 public:
  void print_info() const; // print major
  void useComputer(); // write a paper
  int major; double gpa;
};
class CSStudent : public Person {
 public:
  void print_info() const; // print OH queue pos
  void useComputer(); // fight with Docker
};

void processPerson(Person* p)
{ p->print_info();
  p->useComputer(); }

int main(){
  Person p(...);     processPerson(&p);
  Student s(...);    processPerson(&s);
  CSStudent cs(...); processPerson(&cs);
  // more
}
```

Review Questions 2

- What does "=0;" mean in the declarations to the right?
  - Pure virtual function
- What do we call a class with 1 or more of these kind of declarations?
  - Abstract class
- Is it okay that Student doesn't provide a useComputer() implementation?
  - Yes, it inherits Person::useComputer()
- Can we declare Person objects? No
- Can we declare pointers or references to Person objects? Yes
- When should a class have a virtual destructor?
  - When at least one other virtual function is declared in the class

```
class Person {
 public:
  virtual void print_info() const = 0;
  virtual void useComputer(); // stream a show
  string name; int id;
};
class Student : public Person {
 public:
  void print_info() const; // print major
  int major; double gpa;
};
class CSStudent : public Person {
 public:
  void print_info() const; // print OH queue pos
  void useComputer(); // fight with Docker
};

void printPerson(Person* p) { p->print_info(); }
void compute(Person& p)     {  p.useComputer(); }

int main(){
  Person p(...);  // Allowed?
  Student s(...);    useComputer(s);
  CSStudent cs(...); printPerson(&cs);
  // more
}
```

Efficiency

Data Structure	Operations			
Vector	Push_back()	Push_front()	Get/at(location i)	Pop_front()
Deque	Push_back()	Push_front()	Get/at(location i)	Pop_front()
Singly-Linked List (w/ head ptr only)	Push_back()	Push_front()	Get/at(location i)	Pop_back()
Singly-Linked List (w/ head + tail ptr)	Push_back()	Push_front()	Get/at(location i)	Pop_back()
Doubly-linked list (w/ head + tail ptr)	Push_back()	Push_front()	Get/at(location i)	Pop_back()

Consider this class

- Does this class need to define a copy constructor? If so, define it.

```
class Student {
 public:
   Student(string name, char* mjr) {
     name_ = name;
     major = new char[strlen(mjr)+1];
     strcpy(major, mjr);
   }


   void addScore(int s)
     { scorse.push_back(s); }
 private:
   string name_;
   char* major;
   vector<int> scores;
};
```

Trace the output

```
#include <iostream>
using namespace std;
class Vehicle {
public:
 void drive() {
   honk();
   cout << "Vehicle::drive" << endl;
 }
 void honk()
   { cout << "Vehicle::honk" << endl; }
 virtual void brake()
   { cout << "Vehicle::brake" << endl; }
};
class Bus : public Vehicle {
public:
 void honk() { cout << "Bus::honk" << endl; }
 virtual void brake() {
   drive();
   cout << "Bus::brake" << endl;
 }
};
```

```
class SchoolBus : public Bus
{
public:
 virtual void brake() {
   honk();
   cout << "SchoolBus::brake" << endl;
 }
};

int main()
{
 Vehicle *v1 = new Bus();
 v1->brake();
 Vehicle *v2 = new SchoolBus();
 v2->brake();
 v2->drive();
 delete v1;
 delete v2;
 return 0;
}
// if destructors printed the class name,
// what would you see?
```

Recursion Tracing

```
int gc(int x, int y)
{
  if(y==0) return x;
  else return gc(y, x%y)
}

int main(){
  cout << gc(323 , 85) << endl;
  cout << gc(36, 15) << endl;
```

Recursion Tracing

- Trace this code

```
int m1(int* dat, int len) {
  int temp = -1;
  m2(dat, len, temp);
  return temp;
}

void m2(int* dat, int len, int& num) {
  if(len <= 1){
    num = *dat;
  }
  else if(num == -1){
    num = 0;
    m2(dat+1, len-1, num);
    num += *dat;
  }
  else {
    m2(dat+1, len-1, num);
  }
}

int main()
{
  int data[4] = {3, 6, 2, 9};
  cout << m1(data, 4) << endl;;
  // what will be output?
}
```

Programming I

- Zip 2 arrays of the same size (alternate taking from each array) into a new 3rd array and return that 3rd array.

Programming II

- Given a singly linked list storing values in Item structs (as shown below) and given a head pointer, write a function to delete the 2nd Item in the list (if it exists)?

```
struct Item {
  int val;
  Item* next
};
void deleteSecond(Item* head){









}
```

Programming III

- Make a `Change` class with data members:
  - quarters, dimes, pennies (no nickels)
  - Provides a constructor to initialize those three values to user-specified arguments but **normalizes** to use as many quarters as possible, then as many dimes, then as many pennies (i.e. if they pass 1 quarter, 3 dimes, and 11 pennies you'd want to store 2 quarters, 1 dimes, and 6 pennies
  - Support operator+ and operator==
    - Always re-"normalize" after adding
  - Support an ostream operator that shows the change in the normalized form "Q:2 D:1 P:6"