CSCI 200: Foundational Programming Concepts & Design

Exam 2 Review

1. What is printed?

```
void my_func( int &x, int y ) {
    x = 52;
    y = 7;
}

int main() {
    int x = 0;
    int y = 0;
    my_func( x, y );
    cout << "x = " << x << endl;
    cout << "y = " << y << endl;
    return 0;
}
```

```
x = 52
y = 0
```

2. String

Write a function called string_append that receives a string as input and outputs a string.

The function needs to return a string that appends to the parameter the text

" is a super coder."

```
std::string string_append(std::string input){
    return input + " is a super coder"
}
```

3. Code

```
// Gnome.h
class Gnome {
public:
  Gnome();
  Gnome( int, int );
  int getValue1() const;
  int getValue2() const;
private:
  int _value1;
  int _value2;
};
```

```
// main.cpp
#include <iostream>
using namespace std;

#include "Gnome.h"

int main() {
  Gnome a( 10, 25 );
  cout << a._value1 << " "
       << a._value2 << endl;
  return 0;
}
```

a.getValue2()
a.getValue1()

** cannot access member data
access control error

3. Questions

a) What message would the compiler display?

b) Correctly rewrite the line of code to correct the error.

the callee will not be changerd by the function

c) What is the purpose of const in the two member functions?

d) What is Gnome() and why doesn't it have a return type? bc it's a constructor

4. What is legal?

```
// Gnome.h
class Gnome {
public:
  Gnome();
  Gnome( int, int );
  int getValue1() const;
  int getValue2() const;
private:
  int _value1;
  int _value2;
};
```

```
// main.cpp - assume appropriate headers
int main() {
  Gnome g1;
  Gnome g2(); a function named g2, return Gnome datatype
  g1._value1 = 52;
  int _value1;
  _value1 = g1.getValue1();
  Gnome g3 = g1;
  g3.g2();
  cout << _value1 << endl;
  cout << _value2 << endl;
  return 0;
}
```

5. Short Answer

- Suppose you have developed a class called MyClass with private data members x and y of type int.
  a) Write the function header for this class's default constructor.
  b) Write the function implementation for this class's default constructor that sets x and y to 0.

```
MyClass(){
x = 0;
y = 0;
}
```

```
MyClass::Myclass
private:
int _x;
int _y;
```

6. Functions

```
Circle Circle::doSomething( const Circle &C ) {
  // does something here
}
```

a) What is the name of the function?

doSomething

b) Is this function a member function? If yes, to what class?

yes, to Circle class

7. Functions cont.

```
Circle Circle::doSomething( const Circle &C ) {
  // does something here
}
```

a) What does the first Circle represent? return type

b) What does the second Circle represent? function belong to that class

c) What does the third Circle represent? data type of the parameter

d) What does the const represent? it means the parameter will not be changed

8. Constructors

- Which of the following are valid constructors? Justify the issue if one exists.
  a) `BankAccount::BankAccount() const`
  b) `BankAccount::BankAccount( double balance )`
  c) `void BankAccount::BankAccount()`
  d) `BankAccount::BankAccount( const string &acct, double balance )`

Valid Constructors
b) BankAccount::BankAccount(double balance) This is a valid constructor. It has the correct name (matching the class), return type, and a valid parameter list.
d) BankAccount::BankAccount(const string &acct, double balance) This is also a valid constructor. It follows all the rules: correct name, no return type, and a valid parameter list.
Invalid Constructors
a) BankAccount::BankAccount() const
Issue: Construcors cannot be declared as const. A constructor's purpose is to initialize or modify the object's member variables to put it into a valid state. A const qualifier on a member function promises that it will not modify any memb variables. These two concepts are contradictory.
c) void BankAccount::BankAccount()
Issue: Constructors cannot have a return type, not even void. The compiler knows it's a constructor because its name mat the class name. Specifying any return type makes the compiler think it's a regular member function that just happens to the same name as the class, which is illegal.

9. Member Functions

- Which of the following are valid member functions implementation headers? Justify the issue if one exists.

a) `double HotDog::getPrice() const`

b) `Triangle::calculateArea()`

in general, we want to pass by reference

c) `Buffalo Buffalo::buffalo( Buffalo buffalo )`

d) `void Dog::fetchBall`

e) `double AlarmClock::ring( float )`

Valid

a) double HotDog::getPrice() const This is a valid implementation for a const member function that returns a double.

e) double AlarmClock::ring(float) This is valid. It's legal to omit the parameter name in the function's definition, as long as the type (float) is specified.

Invalid

b) Triangle::calculateArea()

Issue: It's missing a return type. All functions (except constructors and destructors) must specify a return type (like double, int, or void).

c) Buffalo Buffalo::Buffalo(Buffalo buffalo)

Issue: This is an invalid copy constructor. The parameter is passed by value, which would cause infinite recursion

10. What is printed?

```
// Gnome.h
class Gnome {
public:
  Gnome();
  Gnome( int, int );
  int getValue1() const;
  int getValue2() const;
  int diff();
  int diff( const Gnome &G );
private:
  int _value1;
  int _value2;
};
```

```
int Gnome::diff() {
  return _value2 - _value1;
}
int Gnome::diff( const Gnome &G ) {
  return this->_value2 - G._value1;
}
```

```
int main() {
  Gnome a( 10, 25 ), b( 5, 20 );
  cout << a.diff() << " "
       << a.diff( b ) << endl;
  return 0;
}
```

15 20

11. Army of Gnomes!

```
// Gnome.h
class Gnome {
public:
  Gnome();
  Gnome( int, string );
  int getValue1() const;
  string getName() const;
private:
  int _value1;
  string _name;
};
```

- Declare a vector of Gnomes. Then add two Gnomes:
  - harry with value 35
  - sally with value 38

```
int main() {

vector <Gnomes> gomes;
gomes[0] = Gome(35, “harry”);
gomes[1] = Gome(38, “sally”);

}
```

12. Composition

```
class Chair { // in Chair.h
public:
  Chair();
  Chair( int, int, int, double );
  // all getters and setters
private:
  int _height, _width, _depth;
  double _price;
};
class Table { // in Table.h
public:
  Table();
  Table( int, int, int, double );
  // all getters and setters
private:
  int _height, _width, _depth;
  double _price;
};
```

- Write a .h file to define a new class DiningSet. DiningSet has two chairs and one table, a bool on whether the set is sold, and a getPrice() function.

13. Composition

a) Write the function implementation of the Chair's default constructor. Use 10.0 for the price and 1 for the height, width, and depth.

b) Write the implementation of getPrice() for your DiningSet class. getPrice() is equal to the sum of the table and chairs price.

14. Pointers

```
01 int a = 5;
02 int b = 6;
03 int *c = &a;
04 int *d = &b;
05 int *e = new int(7);
06 int *f = new int;
07 int *g = new int;
08 f = c;    it's a shallow copy, pointing f to what c is pointing at
09 *g = *c; it's a deep copy.
10 a = 8; also updateing c and f
11 *d = 9;
12 *f = 1;
13 *g = 2;
14 *c = 3;
15 delete e;
cannot delete stack memory (seg fault) #16
16 delete f;
17 delete g;
```

1. What is the final value of a & b?
2. What do c, d, e, f, g point to?
3. f is what type of copy of c?
4. g is what type of copy of c?
5. Which of Lines 15, 16, 17 will result in an error? Why? What is the error?

1. 5, 6
2. memory address of integers
3. shallow
4. shallow
5. memory leak at #8, because there's still some memory left un-deleted

15. Pointers Part 2

```
#include <iostream>
using namespace std;

void foo(int* pX) {
  *pX = 4;
}

void bar(int*& pY) {
  *pY = 5;
}

int main() {
  int *b = new int(2);
2 cout << "1 - " << *b << endl;
  foo(b);
4 cout << "2 - " << *b << endl;
5 bar(b);
  cout << "3 - " << *b << endl;
  return 0;
}
```

1. What is the output?
2. Sketch out the memory usage.

First, int *b = new int(2); creates a pointer b on the stack. This pointer b stores the memory address of a new integer on the heap, which has the value 2.

Next, foo(b); is called. This is pass-by-value, so a local copy of the pointer, pX, is created. Both pX and b point to the exact same memory address (the heap integer 2). The line *pX = 4; follows this address and changes the value at that location from 2 to 4. Since b in main points to that same address, the value seen by *b is now 4.

Finally, bar(b); is called. This is pass-by-reference, so pY becomes an alias (another name) for the pointer b. pY is not a copy; it is b. The line *pY = 5; follows the address stored in b (via the alias pY) and changes the value on the heap from 4 to 5. Therefore, *b now evaluates to 5. In this entire program, the pointer b itself never changes which address it points to.

16. Pointers Part 3

```
#include <iostream>
using namespace std;

void foo(int* pX) {
  pX = new int(4);
}

void bar(int*& pY) {
  pY = new int(5);
}

int main() {
  int *b = new int(2);
  cout << "1 - " << *b << endl;
  foo(b);
  cout << "2 - " << *b << endl;
  bar(b);
  cout << "3 - " << *b << endl;
  return 0;
}
```

```
1 - 2
2 - 2
3 - 5
```

1. What is the output?
2. Sketch out the memory usage.

First, int *b = new int(2); creates a pointer b on the stack. This pointer b stores the memory address of a new integer on the heap, which has the value 2.

Next, foo(b); is called. This is pass-by-value, so a local copy of the pointer, pX, is created. The line pX = new int(4); changes only this local copy pX to point to a new integer 4. The original pointer b in the main function is not affected and still points to the integer 2. The new integer 4 is now leaked.

Finally, bar(b); is called. This is pass-by-reference, so pY becomes an alias (another name) for the pointer b. The line pY = new int(5); changes pY to point to a new integer 5. Since pY is just an alias for b, this action also changes the original pointer b in main to point to the new integer 5. The original integer 2 is now leaked.

17. Analysis

- What is the run time of the following block of code?

```
int matches = 0;
string line1, line2;
getline(cin, line1);
getline(cin, line2);
int shorterLine = min(line1.length(), line2.length());
for(int i = 0; i < shorterLine; i++) {
  if(line1.at(i) == line2.at(i)) {
    matches++;
  }
}
if(line1.length() == line2.length() && line1.length() == matches) {
  cout << "Lines are equal" << endl;
} else {
  cout << "Lines are not equal" << endl;
}
```

O(n)

18. Analysis 2

- What is the run time of the following block of code?

```
int matches = 0;
string line1, line2;
getline(cin, line1);
getline(cin, line2);
for(int i = 0; i < line1.length(); i++) {
  for(int j = i; j < line2.length(); j++) {
    if(i == j) {
      if(line1.at(i) == line2.at(j)) {
        matches++;
      }
    }
  }
}
if(line1.length() == line2.length() && line1.length() == matches) {
  cout << "Lines are equal" << endl;
} else {
  cout << "Lines are not equal" << endl;
}
```

O(n^2)

19. Analysis 3

- What is the run time of the following block of code?

```
int matches = 0;
string line1, line2;
getline(cin, line1);
getline(cin, line2);
for(int i = 0; i < line1.length(); i++) {
  for(int j = i; j < line2.length(); j++) {
    if(i == j) {
      if(line1.at(i) == line2.at(j)) {
        matches++;
      }
      break;
    }
  }
}
if(line1.length() == line2.length() && line1.length() == matches) {
  cout << "Lines are equal" << endl;
} else {
  cout << "Lines are not equal" << endl;
}
```

O(n^2)

20. Analysis 4

- Of Questions 17, 18, 19:
  - Which have the best performance? 17
  - The worst? 18 19 are equally bad

21. The Big 3

- What are the Big 3?
- What is the Rule of 3?
- Why should we follow the Rule of 3? What can occur if we don't?
- How do the Big 3 relate to shallow/deep copies? What is the difference between the two?

three special member functions that are crucial for managing resources, especially dynamically allocated memory, within a class

Double deletion: Multiple objects pointing to the same memory, leading to errors when destructors are called.
Memory leaks: Resources not being properly deallocated.
Dangling pointers: Pointers referring to already deallocated memory

If a class manages resources (like dynamically allocated memory), the default compiler-generated versions of these functions might perform a "shallow copy," simply copying pointer values instead of the underlying resources.

22. Programming Paradigms

- What is the difference between Procedural Programming and Object-Oriented Programming?

```
procedural -> describe the outcome of
oop -> use objects to describe data & operations
```

- Write an example block of code that illustrates each style in use.

```
Dog Ann = new Dog();
Ann.bark();

Dog(dog()).bark();
```

23. File I/O

- Given a file named “xc.txt” with the following data

```
n

x1  x2  x3  …  xn
```

  - Where the first integer in the file, `n`, states how many integers will follow in the file (`n` will be at least 1)

```
#include <fstream>

ifstream fin (“...”);

vector<std::int> array;
if (!fin.fail()){
    while(!fin.eof()){
        array.push_back(c);
}}

int index = 0;
for (size_t i=1; i<array.size(); i++){
    if (array[index] > array[i]){
    index= i;
}
}
```

n-1

- Write a program to read in all the integers and print out the largest & smallest integer.