Dec 10th

Inclass :

Tuesday **8am** NH 454

Online

Friday **12** CH71

Saturday 10am CH71

Office Hours During Finals Week

- Monday 11-11:50am
- Thursday 9-9:50am

FAB 120-19

We Learned:

- → • OOP
- → • Hierarchies / Inheritance (single)
- → • Dynamic Binding
- *→ • Operator Overloading
  - rvalue
  - lvalue
  - friend
  - const object
- *→ • Java – Differences – NO I/O, Exception
- ~~• Copy Constructors~~
- ~~• RTTI~~
- → • Exception Handling ← C#

- *→ • DATA STRUCTURES
- → • User Defined Type Conv. ∟ C#
- → ~~• Templates~~

Final Exam Topics

CODING

- Inheritance, (dynamic binding)
- Operator Overloading
- (Little) Java

Data Structures (coding)

- LLL
- CLL
- DLL
- BST

Concept

- c++ Operator Overloading
- c++ Exception handling, ~~templates~~, Conversions
- Java

Final *(100 points)*

CS 202: Programming Systems

Name:______________________________

Email Address: ________________

1. C++ Operator Overloading (25 points)

Assume you are writing a string class where the private data member is a char *

a) Select an rvalue operator and show the prototype

char operator ++ (int);

string operator + (const string &);

2nd operand

c) Select an lvalue operator and show the prototype

string & operator = (const string &);

d) Discuss any efficiency issues that may arise with any of the above operations

RVALUE — copy constructor!

c) Show how one of the operators in a) or b) can be used (e.g., from main). Show all variable definitions

string obj1, obj2;

obj1 = obj2;

f) In what situations can an operator be overloaded as a member function rather than a friend?

First Operand , LVALUE ops are members

g) List three different operators that are RVALUE operators (you do NOT need to provide prototypes this time. Just show the symbols.

+ - * / %

< > <= >= == !=

++ (postfix)

2. C++ Dynamic Binding and Hierarchics (25 points)

a) Explain the process of operator overloading used in conjunction with dynamic binding for…

member operators:

"virtual"

non-member friend operators:

Virtual Helper Member Function

b) What is the difference conceptually between an abstract base class and just a base class?

- what is the difference syntactically?

Pure virtual Function

d) In what situations does user defined type conversions help us:

e) On the other hand, explain what happens if we were to use type conversions….

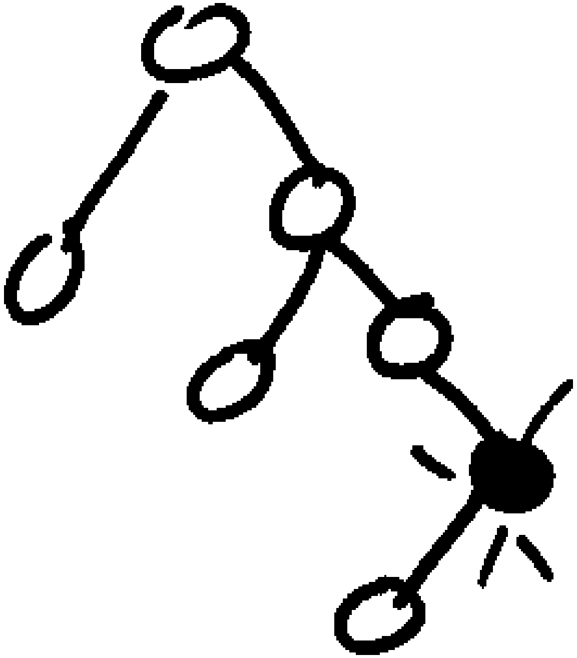
3.(25 points) C++ Coding questions **WRITE ALL OF THE FUNCTION(s) that you use (except you may call strcmp, strcpy, and strlen)**

a. Write the code for a copy constructor for a binary search tree

Use the following LL node structure (you may not modify this!)

```
struct node {
        char * name;  //a person's first and last name
        node * next;
};
```

- Copy BST
- Copy LLL, DLL, CLL
- Add a node to the end of LLL, DLL, CLL
- Remove a node from end of LLL, DLL, CLL
- Remove Largest (or Smallest) item from BST



How would the above change if the node structure was replaced with a node class, with private data (and no friends):

b) Now write a C++ function to remove the last node from a doubly linked list (given a head pointer – but not a tail pointer)

4.(25 points) Java

Java C++

a. List 4 differenccs between Java and C++

1. Abstract vs Pure Virtual

2. No Prototypes vs Class interfaces w implementation

3. Garbage Collector vs deallocate (No delete)

4. No Op. Ov.

b.List 4 similarities between Java and C++

1. New

2. OOP

3. References are Pointers

4. ; (almost)

3.

4.

c. Explain the process of dynamic binding in Java

Show code for how a dynamically bound function can be called in Java ← BaseRef = new derived();

BaseRef.function();

b. When programming in Java, explain why the issues of deep versus shallow copies are different than they were in C++….

1. Garbage collection

2. Impossible to pass an object of a class by "VALUE" when we pass an object - we are always passing a Reference by VALUE

Show in Java how to create an object of a class:

```
list obj;
```

list *obj;
C++

```
obj = new list();
```

Show in Java how to create an array of 10 ints

```
int array[];
```

```
int [] array;
```

C++ int *array;

```
array = new int[10];
```

```
template <class TYPE>
class stack     typename
{                int
   public:
      stack();   ~stack();
       void pop(TYPE&);
       void push(const TYPE&);
       void display();
   private:
       TYPE * array;
        int size;
        int top;
};
template <class TYPE>
void stack<TYPE>::push(const TYPE& data)
{         ADD
        //Error
```

```
            array [top] = data;
            ++top;
                          ↑
                    Assuming = is overloading
 }
template <class TYPE>
void Stack <TYPE>:: pop (TYPE & result)
{
      if (top > φ)
      {
            --top;
            result = array [top];
                      ↑
      }
   }
```

```
template <class TYPE>                          constant
stack <TYPE>:: stack(                              ↓
                       int sz = 100)
{
        array = new TYPE [sz];
         size = sz;
         top = 0;

  }
 template <class TYPE>
 stack<TYPE>:: ~stack()
 {       delete [] array;
          top = 0;       ↑
 }        size = 0;
```

```
template < class TYPE >
void  stack < TYPE > :: display ()
{
      cout << array [top-1];
 }
```