University of Michigan – Dearborn
Department of Computer & Information Science
CIS 200
Final Examination

April 26-27, 2021

Time: 24 Hours
Total Marks: 70

Professor: J. Shen
Open Book

Name: ____Demetrius Johnson_____

Email: ____meech@umich.edu_____

Directions:

Answer each of the following questions in the space provided in this exam booklet. If you must continue an answer (e.g. in the extra space on the last page, or on the back side of a page), make sure you clearly indicate that you have done so and where to find the continuation.

Where a discourse is called for, please be concise and precise. Write legibly; no marks can be given for answers, which cannot be decrypted.

Marks for each major question are given at the beginning of the question. There are a total of 70 marks.

Good luck.

---

1. ____/10 2.____/10 3._____/10 4._____/10

5. ____/10 6.____/10 7.____/10

Total: _____

Preface

Dear students,

It is my honor to have had the opportunity in teaching you in CIS 200 this semester. As the Turing award winner, Jim Gray, pointed out, we are now experiencing the fourth Science Paradigm (Data-driven) in human history. Big data and artificial intelligence are exploding and penetrating into every discipline in our society. I am happy to take this learning ride with you over all the fundamental data structures in computer science and the syntax of C++ language. I am confident that your hard work today will be paid off in the future, and the techniques and skills you learnt today will be used to serve our society in a better way.

Jie Shen,
Dearborn, Michigan.
April 24, 2021

Question 1 (10 marks) Variables and Expressions

(1) When should you use long long int?

We should use a long long int when we are dealing with non-discrete (no decimals needed) values for a problem and one that requires very large numbers; one example could be for a very large data base, where locations in a data base may be transformed into a number in order to store the location of a set of a files or other data at the location.

(2) What is the meaning of R33 in the following code?

```
typedef   float R33[3][3];
R33 x;
```

The above syntax in c++ refers to using the typedef to give another alias to a data type or variable type; so above, typdef is applied to R33 to describe an alias for creating a 2D float array with dimensions [3][3]; in doing this, it is easier to declare a 2D float array of that size by simply using the defined alias: R33.

(3) What is the purpose of sizeof( ) function ?

In memory, all locations contain a 32-bit address (or 64-bit depending on the architecture; I learned about architecture in my CIS 310 course and of course about memory in 200!). Starting at a given memory location, a certain block size (in bytes) is taken up depending on the type of data/data type stored there. So the sizeof() function allows for the retrieval of memory blocks taken up by a given data type. For example, sizeof(char) would return 1 (as in 1 byte). This can be applied to functions, classes, and data types to retrieve the size that is taken up in memory for the allocation of the given type.

(4) What is the difference between & and && as an operator?

In short, the & symbol is used as a bitwise operator to compare bit-by-bit the bits of two given memory locations. It will give a certain output by comparing each bit of two binary values to one another and producing the output of the new set of bits resulting from each bit-by-bit comparison. The && operator is a logical operator that compares two values as a whole (all bits must match to get a certain output); for example 1 && 1 is 1 (true).

(5) When should you use operators: ‘<<” or ‘>>’?

We use the output and input operators to send data on a stream from a variable using “<<”, or send data to a variable using “>>”. So for example you could send data to a file or the computer screen using cout (or filestream) << var_X; or, you could send data to a variable from a stream such as the keyboard or an input file: cin >> x.

(6) Under the Integer Category, we have data types: char, short, int. What criterion should you follow when picking one of them ?

When selecting a data type, it should always be considered the task that the program will perform so that you can most efficiently select a data type that will make your program most efficient and run the smoothest. For example, you would use char data type, which uses the least memory, to store data describing the color of pixel on a screen which typically uses RGB color schemes (red green blue). It would be a waste to use anything greater than 1 byte of memory, otherwise you might have a photo or video created that is unnecessarily large. In the same manner, short is a 16-bit (2-byte) integer data type, while int is a 32-bit (4-byte) data type; and as with any data type, you want to choose the best data type for you applications; so for example if you our working with arrays and smaller integers, it is best to use a short in order to save memory.

(7) Why do we sometime use ‘unsigned char” instead of ‘char’ ?

Unsigned char and char take up exactly the same amount of space in memory; we use ‘unsigned’ for any data type in order to use all bits that the data type occupies to specify only positive values. So instead of splitting the values evenly between positive and negative values (char), we use unsigned char so we get double the range but all going towards the positive value range (unsigned char) and sacrificing the ability to use a negative value for an unsigned variable. If you are writing a program that involves a large range but with only positive values, you may be able to save memory by stepping down to a smaller data type and using the unsinged version of it.

(8) Explain the keyword ‘register’

Using the keyword ‘register’ allows for a program to store a variable directly into a CPU data register for easy and much faster access, since CPU register memory is not only closer to or on the CPU but the memory is designed much faster to be processed right away (learned this from 310 too!). Thus; we use this keyword on a variable very sparingly and making absolute certain that it is a necessary and efficient use. We typically will use this keyword in situations where a variable or data type will be used very frequently throughout the program and that would make it well worth it to greatly increase the speed of the program.

(9) Explain the keyword ‘volatile’

We use this keyword on a variable in order to prevent the compiler from doing its normal code optimization algorithms on the particular variable at compile time. We use this because some variables or code situations might be transformed incorrectly when at higher levels (such as GUI interfaces) and cause the program to have a logic error.

(10) Explain the keyword ‘static’ in two cases: (a) a local variable in a function and (b) a data member in a class.

This keyword is used on a function or on a variable in order to tell the compiler not to destroy or create a new copy of the function or variable whenever they are called on. It is used in order to make a function and all its members on the stack be shared by all calls to it, or for a particular variable that needs to be made history-sensitive. For example, a function inside of a class may need to track how many times that particular function was called and use that value for some purpose; in order to do this, you might create a static variable inside the function to be used as a static counter. Without using the ‘static’ keyword, then all functions and variables called and placed on the stack will be destroyed and reset every time they are called on.

Question 2 (10 marks) Struct and Union

(1) Can you have a member function in a struct ? If yes, give a coding example.

Yes, you are allowed to have a member function inside of a struct; although usually the default public aspect of struct is used to store data variables that can be accessed directly without a function.

Here is one example:

```
struct Example{

int x;
int y;
int sumXY(void){return x + y;} //use this to easily return the sum of data members x and y

};
```

(2) Give a definition of a struct (called Student), which has three data members: (i) name (data type: string), (ii) GPA (data type: float), and (iii) age (data type; int).

```
struct Student{

string name;
float GPA;
int age;

};
```

(3) Declare a static array of the Student struct with a length of 10 elements.

```
static Student student_Struct[10];
```

(4) Declare a pointer variable p1, which is points to this Student struct.

```
Student* p1 = &student_Struct;
```

(5) Explain the similarity and difference between struct and union.

Structs and unions are the same in that by default the data members are public and thus can be accessed without having to right set or get functions. The major difference between the two is that structs store the memory of each of its data members in contiguous memory locations, while unions store the memory locations of all data members in a stacked method – starting at the same start address memory location. This means that while unions save memory, you can only use one of the data members of the union at one time; while structs give you the capability of using all of its data members simultaneously.

For instance, if you declare a union with data members of type char and float, when you use the char, it will input junk values into the float (since char and float binary data representations are different, inputting a 'g' character into char will be represented differently by a float). So, for a union, you must reinitialize each data member each time you want to switch from using one to the other because the memory locations overlap (start at same location).

Question 3 (10 marks) Class

(1) What is the main difference between protected and private data members in a class?

Protected data members can be accessed only by the base class, and any classes derived from the base class. Private data members may only be accessed by the base class (with the exception of using friends).

(2) Does C++ support multiple inheritance ? If yes, give a piece of code to demonstrate it.

Yes, c++ supports multiple inheritance; here is an example:

```
class Engine{

protected:
        int num_Pistons;
        int horse_Power;
};


class Wheels{

protected:
        int num_Wheels;
        float diameter_Wheels;
};

class Vehicle : public Engine, public Wheels  {

private:
        string color;
        float mpg;
};
```

(3) What is the purpose of using 'friend' in C++? Suppose A and B are two classes. How do you code to let A be a friend of B?

The purpose of using 'friend' is to allow one class to be able to easily access all the variables of another class, including private data members. Friend relationships are one way; one class may allow friend access to its own class, but that doesn't mean it can access the other class's members; for two-way relationship both classes must declare each other friends.

Here is an example:

```
class A{

private:
        int x;
};

class B{

private:
        int ;
public:
        friend class A; //now A is a friend of B
};
```

(4) What is a virtual function?

A virtual function is a function used in the concept of polymorphism with inheritance; it allows a function with the exact same name in the base class and derived class to take many forms (polymorphism) depending on if the function is rewritten for any derived classes of the base class. If a virtual function is not rewritten for a derived class, the derived class will default to use the virtual function from the parent class.

(5) What is polymorphism?

Polymorphism is a special and unique and particularly useful concept in object-oriented programming. It allows for functions or classes to take on many forms, hence that is why the concept is called polymorphism. Some examples of polymorphism include: virtual functions (functions with the same name in the base and derived class but rewritten with its own set of code), overloaded functions (functions with the same name but a different set of parameters), and template classes or functions (same code but incorporated with the specified data type).

(6) What is a pure virtual function?

A pure virtual takes on the form virtual int function_name(void) = 0; it means that the function is not written for the base class and thus cannot be called from the base class; if a pure virtual function is written for the base class, it means that the class is an abstract class. It is used to specify that the function will be written only for the derived classes as needed.

(7) What is an abstract class ?

An abstract class is a class which contains at least one function that is a pure virtual function. It serves as a base-class to help define many derived classes that will actually be used, or abstracted themselves. An abstract class can only be a base class for other classes; objects of an abstract class cannot be created; if a derived class inherits from an abstract class and does not define all pure virtual functions inherited, then it also becomes an abstract class.

(8) Can we have a nested class ? If yes, can we access the private data member of an outer class from an inner class?

Yes, it is possible to have a nested class; and as far as access goes, it is much like base and derived classes: The base class cannot access members of derived classes, but derived classes can access the base class members (provided members of base were protected).

To answer the question now: YES, the inner class can access private data members of the outer class since the inner class is just like any other data member/function of the outer class, and we know all functions of a class have access to all data members including private data members; however, the outer class cannot access the inner classes private members as the normal rules apply for the inner class in terms of private protected and public members.

(9) What are the three cornerstones of object-oriented programming ?

The three cornerstones are: **Data abstraction** (hiding data), **Inheritance** (hierarchical structures among class relationships that can assist in data abstraction), and **Polymorphism** (the ability of functions and classes to take on many forms by using properties of data abstraction, inheritance, and templates).

(10) What things cannot be inherited from a base class to a derived class?

Things that cannot be inherited from a base class are: **private data members**.

Question 4 (10 marks) Pointer and Reference

(1) Given the following variables, show the way to print out the beginning memory addresses of these variables:

int x=10; float y = 3.14; char z[] = {'a', 'b', 'c'}; Student w; // where // Student is a user-defined class

```
cout << &x << endl;
cout << &y << endl;
cout << z << endl;   //for array and function, the name represents the start address
cout << y << endl;
```

(2) Write a swap function that utilizes pointer variables as a way for pass-by-reference.

```
void Swap(int* x, int* y){

int holder = *x;
*x = *y;
*y = holder;

}
```

(3) Write a segment of code to dynamically allocate a two-dimensional float array and to deallocate this array. The size of this array is determined at run-time based on user input from keyboard.

```
int Row;
int Col;

cin >> Row >> Col; //user determines size of the 2D array at runtime
float** floatArr2D = new float*[Row]; //allocate memory for first dimension

//allocate memory below for the 2nd dimension:

for (int i = 0; i < Row; i++){

        floatArr2D[i] = new float[Col];

}

//deallocate memory below for 2nd dimension:

for (int j = 0; j < Row; j++){ //go to each element in the row and delete the float arrays

        delete[] floatArr2D[j];

}

delete[] floatArr2D; //delete first dimension (array of float pointers)
```

(4) Write a piece of code to construct a linked list that contains three nodes. Nodes 1 through 3 have a value of “Good”, “Morning”, and “America,” respectively.

```
struct LL{

string value;
LL* next;

};

LL stringList; //create a linked list; start address stored here
LL* ptr = &stringList; //create a ptr to point through list and create nodes
ptr→value = “Good”; //set first value
ptr→next = new LL; //allocate memory for next node
ptr = ptr→next; //move to next node
ptr→value = “Morning”; //set second value
ptr→next = new LL; //allocate memory for next node
ptr = ptr→next; //move to next node
ptr→value = “America”; //set third node value
ptr→next = nullptr; //set end of LL to null
```

(5) Explain in which cases we should use void *.

We use the void* pointer to point to any data type since all memory address internally are the same: 32-bit addresses. One case where we use the void* pointer is when we need to use one pointer to point to several data types, and then cast the pointer as the specified data type to which it is pointing in order to manipulate the data stored at the memory address. The only reason for this that I can think of is that we might be trying to save memory, or we may use it in situations with template classes or for functions that we want to be able to accept any data type. We also might use a void* pointer to point to a set of similar functions. Void* is often used with the type_casting function.

(6) Define a function pointer that points to a category of functions, which take two input float arguments and return a float number.

void*

(7) Write a while loop to traverse a linked list from its head to its tail. Assume that the head of the linked list is represented by Node *head;

while(head != nullptr){ head = head→next;}

(8) Write a function to do a deep copy of a linked list. Assume that the head of a source linked list is Node *head1;

Node* ptr = head1; //create a ptr and set it to the head; will use this to move through list
Node* newList;

while(ptr != nullptr){ //iterate through linked list until nullptr (end of list)

newList = new Node; //allocate new memory for the new element in the new list
newList→value = ptr→value; //set values equal; part of deep copy; copy values but not address
newList→next = nullptr; //set next equal to nullptr in case we have reached end of list
ptr = ptr→next; //move to next node to be deep copied

}

Question 5 (10 marks) Stack and Queue

(1) What is the main characteristics of Stack and Queue?

Stack implements First In Last Out, or Last in First Out (FILO, or LIFO), while a Queue implements First in First Out, or Last In Last Out (FIFO, or LILO) in terms of how the elements enter and exit the data structures.

For stacks, elements are added (pushed) and taken off (popped) the stack from one end, while for the Queue, elements are added (enqueued) from the back and removed (dequeued) from the front.

(2) Given the following sequence of statements, draw a stack after the completion of these statements:

Push('a');
Pop()
Push('b')
Push('c')
Push('t')
Pop()

Below, is the current stack after the above statements, with c being at the top of the stack:

(3) Given the following sequence of statements, draw a queue after the completion of these statements:

Enqueue('a');
Dequeue()
Enqueue('b')
Enqueue('c')
Enqueue('t')
Dequeue()

Below, is the current queue after the above statements, with c being at the front of the queue (next in line for dequeue):

Question 6 (10 marks) Sorting and Search

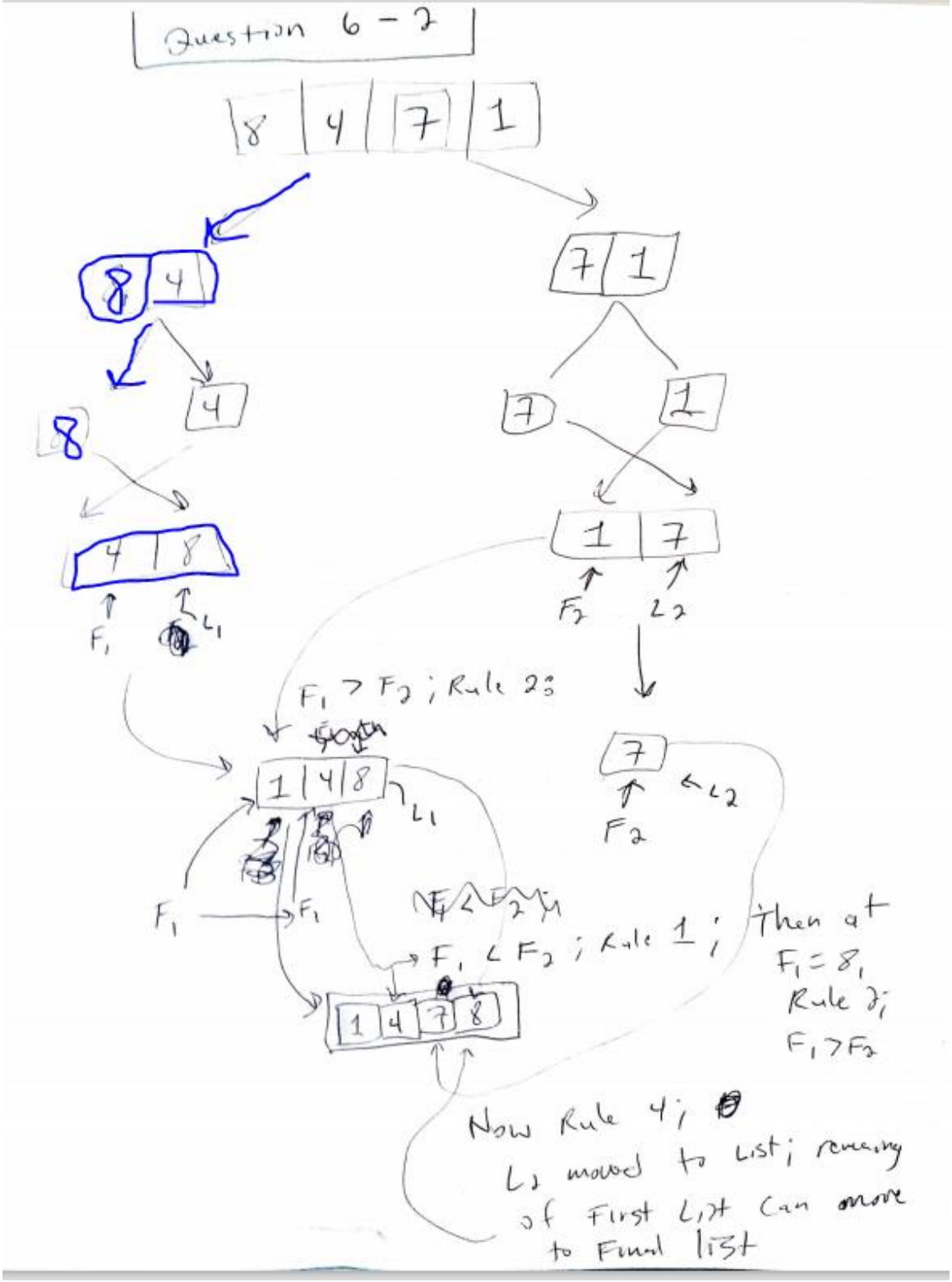
(1) What is the Big-O notation for time complexity of Bubble Sort, Insertion Sort and Selection Sort. Assume n is the number of elements in an input list.

The time complexity of the **insertion** sort is dependent on the scenario of the situation for when the function is called: for average and worst case, the time complexity is O(N^2), and for best case it is O(N). It is better to use this algorithm for smaller arrays.

The time complexity for **bubble** sort for average and best-case scenario is O(N^2).

The time complexity for **selection** sort is O(N^2).

(2) Given an unsorted input list: [8, 4, 7, 1], show the step of merge sort for sorting this list.



(3) Given an input array: char z[] = {‘a’, ‘t’, ‘c’, ‘e’, ‘d’};
write a code to demonstrate how to use qsort() for sorting this array.

```
char splitVal = z[5/2]; // splitVal is assigned to ‘c’
char* ptr = z;

while(*ptr < splitVal){



}
```

(4) Explain the basic idea of binary search. What is the time complexity of binary search and linear search ?

The time complexity of a binary search is equal to O(logN). This is because of how a binary search algorithm works: they can only be performed on binary search trees or sorted arrays, and the idea is that for each node, the left child is always less than the parent node, and the right child is always greater than the parent node. Thusly, the binary search will have to search through at most the height of the tree, which is determined by a maximum of log_2(n), where n is number of elements in the tree or array. For sorted arrays, it is just like a tree, except for we know that for each element, the left element is less, and the right element is greater (or depending on if the array is descending or ascending order). So when you search, you simply start at the center of the list, and check if the search value is greater or equal, and thus you can eliminate half of the elements to search each time you compare with the new center element, since all elements are sorted.

(5) If you have two sorted sublists, each of which has a length of *n* elements. Design an efficient way to combine these two sorted sublists into one single sorted list. What is the time complexity of your approach?

If I have two sorted sublists, I will use the approach of the merge sort just as we were asked to do in question 6-2, since it has a time complexity of O(NlogN) and since I am given two lists that are already sorted and that need to be merged.

Question 7 (10 marks) Software Engineering, File I/O, and Tree

(1) Explain CRC cards and Software Modifiability

CRC cards are Class, Responsibility, Collaboration cards that can be used to help with designing a program. They are used to break down the classes that will be used, and how they relate to and depend on each other. This ties directly into Software Modifiability; you need to know how classes collaborate in order to more easily spot designs in your code that make it more difficult to modify or if it is an efficient, easily modified implementation, where changes can be made to the code without so much difficulty or destroying or undermining the logic of the overall program.

(2) What is the difference between general trees and binary trees in computer science

A general tree in computer science could be a variety of different kinds of trees where the nodes are not sorted using the binary search tree method but where there can still only be up to 1 node for the left child and 1 node for the right child. For binary trees, each node has up to 1 left and right child, but the left child is always less than the parent, while the right child is always greater than the parent.

(3) Given an input sequence: "E", "C", "T", "B", "D", "Q" and "W", construct a binary search tree that corresponds to this input sequence. Draw this tree.

(4) If we traverse this tree in Inorder, what is the sequence to visit the nodes?.

BCDEQTW

(5) If we delete ‘E’ from this binary search tree, what is the resulting tree.

(6) Write the definition of a struct for representing a node of a binary search tree:

```
struct Int_BST{ //example of a struct for a binary tree containing ints

int value; //value that the node stores
Int_BST* right; //right child
Int_BST* left; //left child

};
```

(7) Write a piece of code to demonstrate how to use ‘get’ and ‘put’ pointers in file stream operations:

```
int oldVal = 2;
int newVal = 7;
int x  = 0;
ofstream outputFile;
ifstream inputFile;
outputFile.seekp(ios::beg, 2); //move from start of file 2 bytes forward
outputFile << oldVal; //file reads _ 2
outputFile.seekp(ios::cur, -2); //move seekp pointer back two bytes in file@curpos
outputFile << newVal; //now file reads 7 2
        inputFile.seekg(ios::beg, 2);
        inputFile >> x; //now x has value 2 instead of 0
```