Final - Overview

FORMAT

- Bring a scantron
- Some T/F
- Some Mult Choice
- Some Problem Solving
- NOT open notes/book

Topics Covered

1. Basic Input/Output
2. Arithmetic in C++
3. Selection
4. Repetition
5. Functions
6. Arrays
7. Searching and sorting
8. Structs
9. Pointers
10. Linked-Lists
11. ADT's
12. Object oriented programming
13. Recursion

Some tips – avoiding test anxiety

- Get a good nights rest
  - I know this is tough, but you don't think as well without sleep
- Don't skip a meal before an exam
  - Your brain needs protein → try not to eat a high carb meal
- Don't Cram! Pace your studying
  - Try not to put it off until the last minute
  - If you pace yourself → you will be prepared
- Study with classmates so you can compare notes
  - don't discuss the exam just before coming in
  - their anxiety may impact you
- Take deep breaths → relax yourself
  - Thing positive thoughts → remind yourself that you are prepared
- Don't get bogged down on a question
  - answer the questions you know quickly → go back to the others
- Ask Questions
  - Calm yourself before you come in…
- Avoid being late

POINTERS

- Be able to …
  - Write a piece of code to add a new node to a list..
    - To the front
    - To the tail
    - To the middle
- Be able to remove a node
  - From the front
  - From the tail
  - From the middle
- Know how to search a linked list
  - What type of loop should you use?
    - a while loop
  - What should your loop check for?
    - That you haven't reached the end of the linked-list
    - That it hasn't been found
- Know how to access members of a struct using pointers.
  - pPtr -> *member*
- What if you want to access what the pointer member is pointing to? pPtr -> next -> *member*

5

ADTs

- What is an ADT?
  - Abstract Data Type
- What is a stack?
  - Items are added to the front, removed from the front
- What is a queue?
  - Items are added to the rear, removed from the front
- What is LIFO?
  - Last in First Out
- What is FIFO?
  - First in, First Out
- Which represents a stack?
  - LIFO
- Which represents a queue?
  - FIFO
- T/F
  - Stacks can only be represented using linked-lists
    - False
  - Queues can only be represented using linked-lists
    - False

6

Structs

- What is the advantage of using structs?
  easier to organize related data items.
- What is a member?
  a field within the structure
- Are aggregate operations allowed on structs?
  only assignments
- Can you pass structs by value or reference?
  yes
- Can structs be a return type?
  yes
- How do you access a member of a struct?
  dot operator
- What if you are using pointers
  ->

7

Structs

- Define a struct called DvdRec, that contains the title, genre, and running time.

```
struct DvdRec
{
    string title;
    string genre;
    int    time;
};
```

- Declare an array 100 elements of that struct called movies.
  DvdRec movies[100];
- How would you output the title of the 10th element in your array?
  cout << movies[9].title;
- Be able to write a function that can read into an array of structs or output an array of structs.

8

Know This

- How to format input and output
- How to use input/output files
- How to search/sort using arrays
  - Review last notes on arrays
- Know how to use functions
  - The difference between passing by value/reference.
  - What type of function you should use if there is one return value
    - Multiple return values
    - No return value
  - Know how to pass…
    - Arrays
    - Linked lists
    - structs
  - Know how to write a function so that it is reusable
    - Be general
    - Accomplish 1 task

Know This

- Know how to use arrays
  - How to search an array
  - How to sort an array
  - How do you set up a multi-dimensional array?
  - How do you pass a multi-dimensional array?
  - Which index represents the row?
- Know boolean expressions
  - What is the opposite of between 100 and 300
- Know loops!
  - What 3 things do you need to do?
  - When do you do them for each loop?
- Know how to error check input
  - What loop should you execute?
  - What include files do you need?

OOP

- What is the difference between Unstructured and Structured programming?
  - Unstructured – everything is in the main
  - Structured – Data is passed to functions – some data is managed by main – some by functions
- What is the difference between Structured and OOP?
  - Structured – focus is on functionality and data is passed through functions
  - OOP – focus is on data and methods are used to access data
- What is information hiding?
  - Keeping the data private and how methods are implemented hidden so that changes won't effect the code using the objects
- What is the difference between a mutator and an accessor?
  - mutators are methods that modify attributes –
  - accessors don't modify attributes

11

OOP

- What is a constructor?
  - A special method that is called upon when an instance of an object is declared that instantiates the object
- What is a destructor?
  - A special method that allows an instance of the class to be terminated
- What does it mean to say a class is composed of another class?
  - Attributes in one class are class types
- What is inheritance?
  - When a new class (the derived class) is created from another class (base class)
- What is polymorphism?
  - The ability to associate many meanings to one function name by means of using virtual functions or late binding (dynamic binding).
- What is a virtual function?
  - A function that can be used before it is defined.
- What is the difference between static and dynamic binding?
  - Static binding happens at compile time – overloading
  - dynamic binding happens at run time - overridding

12

OOP

- What is overloading?
  - Defining two methods of the same name in the same class differentiating them by their signatures
  - Functions can also be overloaded - so can operators
  - Static-binding (resolved at compile time)
- What is overriding?
  - Redefining a method that has already been defined in the parent class (using the same signature)
  - Dynamic binding (resolved at run time)

13

OOP

- Define a class called person that manages a students name and ID.
  - Declare some basic methods for name and ID
  - This class will be a parent class - include a virtual method to Display

```
#include <string>
#include <iostream>
#include <iomanip>
using namespace std;
class Person
{
   public:
        Person ();
        virtual ~Person();
        void   SetInitialValues(string personsName,
                                int    id);
        void   ChangeName(string personsName);
        string GetName()const;
        int    GetID()  const;
        virtual void   DisplayAll ()  const;
   private:
        string name;
        int    idNum;
};
```

14

OOP

- Define a class for a student that is derived from Person. Extend with the total number of classes taken, and GPA.
  The programmer using this should be able to add a number of classes, change the age, and change the GPA. They should also be able to Display all attributes.

```
#include "Person.h"
class Student : public Person
{
   public:
        Student ();
        virtual ~Student ();

        void   SetInitialValues(string studentName,
                                int    studentIds,
                                int    classesTaken,
                                float  currentGpa);
        void   AddClasses(int numberOfNewClasses);
        void   ChangeGPA (float newGpa);

        int    GetClasses() const;
        float  GetGPA()     const;

        virtual void DisplayAll() const;
   private:
        int    totalClasses;
        float  gpa;
};
```

Why not virtual?

OOP

- Implement the constructor and AddClasses methods

```
Student::Student()
{
        totalClasses = 0;
        gpa          = 0.0;
}

void Student::AddClasses(int numberOfNewClasses)
{
        totalClasses += numberOfNewClasses;
}
```

- Also be able to write code to use this class. Declare an instance (object) of Student called Mary and call the methods AddClasses to add 5 classes to Mary

```
Student mary;
mary.AddClasses(5);
```

OOP

- Implement the method DisplayAll in the Person class

```
void    Person::DisplayAll ()  const
{
        cout << left
             << setw(25) << name
             << right
             << setw(6) << idNum;
}
```

- Implement the method DisplayAll in the Student Class

```
void    Student::DisplayAll() const
{
        Person::DisplayAll();
        cout << setprecision(2) << fixed;

        cout << setw(8) << totalClasses
             << setw(4) << gpa;
}
```

- Declare an instance (object) of Student called Joe and the call for the methods SetInitializeValues and DisplayAll

```
Student joe;
joe.SetInitialValues("Joe Doe", 1234, 12, 2.75);
joe.DisplayAll();
cout << endl;
```

17

OOP

- Implement the method SetInitalValues in the Person class

```
void    Person::SetInitialValues(string personsName,
                                 int    personsId)
{
        ChangeName(personsName);
        idNum = personsId;
}
```

- Implement the method SetInitalValues in the Student Class

```
void    Student::SetInitialValues(string studentName,
                                  int    studentId,
                                  int    classesTaken,
                                  float  currentGpa)
{
        Person::SetInitialValues(studentName, studentId);

        totalClasses = classesTaken;
        ChangeGpa(currentGpa);
}
```

18

OOP

- Create a dynamic instance (object) of Student using a parent pointer (Person), assume that we have constructor in the Student class that allows to initialize all attributes in the Person and Student classes, invoke this constructor when creating this instance. Call the method DisplayAll

```
Person *newStudent;

newStudent = new Student("Joe Doe", 1234, 12, 2.75);

newStudent->DisplayAll();
cout << endl;
```

- Which DisplayAll() method was executed in the previous program, from the Person class or Student class?

  From Student class

19

OOP

Assume you have implemented Faculty which will also be inherited from Person.

- Each DisplayAll is implemented differently to handle the extended attributes of the different classes.
- You want to create one function to execute DisplayAll for any Person.
- Write the code segment in main that declares an instance of each of the classes derived from Person and calls DisplayPerson function
- Then write the function.

```
Person  *myPer;
Student *myStud;
Faculty *myFaculty;
Admin   *myAdmin;

myStud = new Student("Joe Doe", 1234, 5, 3.75);
myPer = myStud;

cout << DisplayPerson(myPer);

myFaculty = new Faculty("Mo Doe", 4321,"C.S.", 7);
myPer = myFaculty;

cout << DisplayPerson(myPer);
```

```
string DisplayPerson(Person  *myPer)
{
   string outstr;

   outstr = myPer->DisplayAll()+'\n';

   return outstr;
}
```


20

OOP

- Assume there is a non-default constructor that initializes all values of a person
- Declare a pointer of the base class, create an instance of the derived class - call the non-default constructor, and invoke DisplayAll

```
Person *myPerson;

myPerson = new student ("Joe Doe", 1234);

cout << myPerson->DisplayAll();
```

- What is wrong with this code? Slicing Problem
- Fix it!

```
Person  *myPerson;
Student *myStudent;

myStudent = new Student ("Joe Doe", 1234, 5, 3.75);

myPerson  = myStudent;
cout << myPerson->DisplayAll();
```

21

Virtual Function Example

Make the destructor virtual too

```
class Pet
{   public:
       Pet();              // constructor
       virtual ~Pet();     // destructor
       …
       virtual void   Speak    () const;
    protected:
       string petName;
       int    petAge;
       float  petWeight;
       string petBreed;
    private:
};
```

DERIVED CLASS

```
class Mouse : public Pet
{
    public:
      // constructors
      Mouse(); Mouse(string name,
                     int    age,
                     float  weight,
                     string typeOfPet);
      virtual ~Mouse();  // destructor

      /**** ACCESSORS ****/
      virtual void Speak() const;
};
```

IMPLEMENTATIONS

```
void Pet::Speak() const
{
   cout << petName << " is speaking...";
}
```

```
void Dog::Speak() const
{
   cout << petName << " says Woof Woof!";
}
```

```
void Cat::Speak() const
{
   cout << petName << " says Meoowww!";
}
```

```
void Mouse::Speak() const
{
   cout << petName << " says squeak!";
}
```

It isn't necessary to make the method virtual the derived class - but it is good practice

Example Main & Output

```
#include "OOP Header.h"
#include "pet.h"
#include "dog.h"
#include "cat.h"
#include "mouse.h"
int main()
{
    Pet *buddy;
    Pet *bear;
    Pet *ben;
    Pet *fluffy;

    fluffy = new Pet  ("Fluffy");
    buddy  = new Dog  ("Buddy",  6, 32.5, "Dog");
    bear   = new Cat  ("Bear" , 18, 20.5, "Cat");
    ben    = new Mouse("Ben"  ,  1,  0.5, "Mouse");

    buddy  -> Speak();
    bear   -> Speak();
    ben    -> Speak();
    fluffy -> Speak();

    fluffy-> Display();
    buddy -> Display();
    bear  -> Display();
    ben   -> Display();

    return 0;
}
```

These all need to be Pointers
→ for Dynamic binding

This enables sub-type polymorphism

OUTPUT

```
Buddy says Woof Woof!
Bear says meoowww!
Ben says squeak!
Fluffy is speaking...

Fluffy    0    0.00
Buddy     6   32.50   Dog
Bear     18   20.50   Cat
Ben       1    0.50   Mouse
```

Exercise

Which method is called? (assume classes below)

```
class Parent
{
        public:
                void method1();
                virtual void method2();
                void method3();
                …
};

class Child: public Parent
{
        public:
                void method1();
                virtual void method2();
                …
};
```

Which Method will be called?

```
…
int main()                                  // handling dynamic objects and inheritance
{
        Parent *ptrP1;                      // create Parent pointer
        Parent *ptrP2;                      // create another Parent pointer
        Child *ptrC;                        // create Child pointer
        …
        ptrP1 = new Parent;                 // create Parent object - Parent pointer
        ptrP2 = new Child;                  // create Child object - Parent pointer
        ptrC  = new Child;                  // create Child object - Child pointer

        ptrP1 -> method1();                 // Parent class method1() is invoked
        ptrP1 -> method2();                 // Parent class method2() is invoked
        ptrP1 -> method3();                 // Parent class method3() is invoked
        ptrP2 -> method1();                 // Parent class method1() is invoked
        ptrP2 -> method2();                 // Child class virtual method2() is invoked
        ptrP2 -> method3();                 // Parent class method3() is invoked
        ptrC  -> method1();                 // Child class method1() is invoked
        ptrC  -> method2();                 // Child class method2() is invoked
        ptrC  -> method3();                 // Parent class method3() is invoked
        …
}
```

```
class Parent
{
public:
        void method1();
        virtual void method2();
        void method3();
};

class Child: public Parent
{
public:
        void method1();
        virtual void method2();
        …
};
```

Inheritance Exercise

```
Class Parent {
  public:
     void methodBasePublic();
  protected:
     void methodBaseProtected();
  private:
     void methodBasePrivate();
};
```

```
Class ChildPub: public Parent {
  public:
     void methodChildPublic();
  protected:
     void methodChildProtected();
  private:
     void methodChildPrivate();
};
```

```
Class ChildPro: protected Parent {
  public:
     void methodChildPublic();
  protected:
     void methodChildProtected();
  private:
     void methodChildPrivate();
};
```

```
Class ChildPriv: private Parent {
  public:
     void methodChildPublic();
  protected:
     void methodChildProtected();
  private:
     void methodChildPrivate();
};
```

Inheritance Exercise

- List all Methods that can be accessed by:
  - ChildPub
  - ChildPro
  - ChildPriv
- List all Methods that can be accessed by objects of the following classes:
  - ChildPub
  - ChildPro
  - ChildPriv

Inheritance Exercise

- Let's assume now that we derive a new class from the Child classes, we will call them GrandChild
  - Class GrandChild: public ChildPub {};
  - Class GrandChild: public ChildPro {};
  - Class GrandChild: public ChildPriv {};
- Which Methods from the Child and Parent classes can be accessed by these classes?
- Which Methods from the Child and Parent classes can be accessed by objects of these classes?

Recursion

- What is recursion?
  Process in which the result of each repetition is dependent upon the result of the next repetition
- T/F
  Every Loop can be implemented Recursively
  True
- What are the advantages and disadvantages of recursion?
  Advantages: Models certain algorithms most accurately; Results in shorter, simpler functions
  Disadvantages: May not execute very efficiently
- What is a base case?
  The simpler-to-solve problem is known as the base case
  Recursive calls stop when the base case is reached

29