

Topics Covered

- 1. Basic Input/Output
- 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

Final - Review

3

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

• Be able to ... Write a piece of code to add a new node to a list.. To the front P 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 pPtr -> next -> member

• 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** S • Which represents a queue? **FIFO** o T/F Stacks can only be represented using linked-lists False Queues can only be represented using linked-lists

```
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

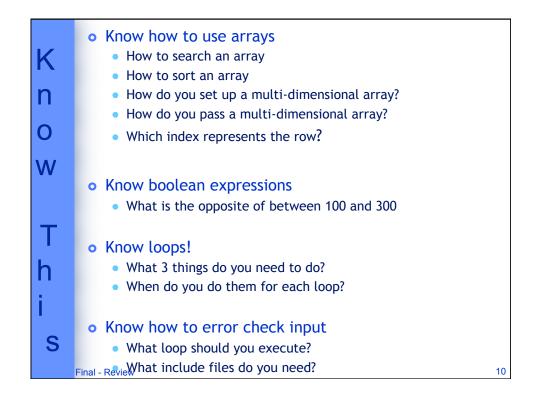
How do you access a member of a struct?

dot operator
What if you are using pointers
```

```
    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
u
          movies.
          DvdRec movies[100];
C
       • How would you output the title of the 10<sup>th</sup> element in
          your array?
          cout << movies[9].title;</pre>
 S
       • Be able to write a function that can read into an array
          of structs or output an array of structs.
```

 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 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 S ■ Be general Final - Review Accomplish 1 task



• 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

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? Chatic binding because at compile time, excepted disc.

Static binding happens at compile time - overloading dynamic binding happens at run time - overridding

• 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

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;
             GetID() const;
       int
       virtual void DisplayAll () const;
  private:
       string name;
       int
             idNum;
};
```

```
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,
                                                                  Why not
                                       int
                                              classesTaken,
                                                                  virtual?
                                       float currentGpa);
          void AddClasses(int numberOfNewClasses);
          void ChangeGPA (float newGpa);
                 GetClasses() const;
          int
          float GetGPA()
                              const;
          virtual void DisplayAll() const;
     private:
                 totalClasses;
          int
          float gpa;
 };
Final - Review
                                                                        15
```

```
o Implement the constructor and AddClasses methods

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

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

o 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);
```

```
o 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

           Student::DisplayAll() const
           Person::DisplayAll();
           cout << setprecision(2) << fixed;</pre>
           cout << setw(8) << totalClasses</pre>
                 << 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;</pre>
                                                                       17
```

```
o Implement the method SetInitalValues in the Person class
 void
        Person::SetInitialValues(string personsName,
                                        personsId)
                                   int
         ChangeName (personsName);
         idNum = personsId;
• Implement the method SetInitalValues in the Student Class
       Student::SetInitialValues(string studentName,
void
                                         studentId,
                                    int
                                    int
                                           classesTaken,
                                    float currentGpa)
        Person::SetInitialValues(studentName, studentId);
        totalClasses = classesTaken;
        ChangeGpa(currentGpa);
                                                                  18
```

 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;</pre>
```

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

From Student class

19

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);</pre>
 myFaculty = new Faculty("Mo Doe", 4321,"C.S.", 7);
 myPer = myFaculty;
                                     string DisplayPerson (Person
 cout << DisplayPerson(myPer);</pre>
                                        string outstr;
                                        outstr = myPer->DisplayAll()+'\n';
                                        return outstr;
                                                                               20
Final - Review
```

```
O Assume there is a non-default constructor that initializes all values of a
    person

O 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);

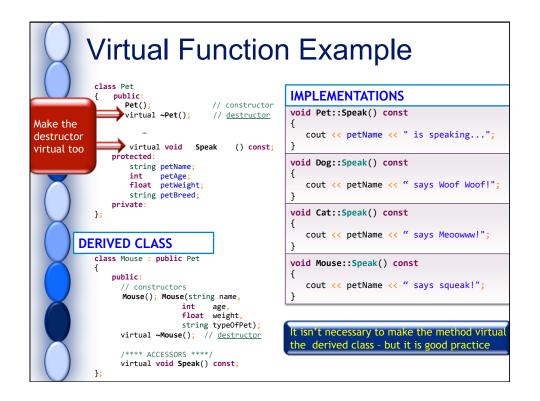
    cout << myPerson->DisplayAll();

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

Person *myPerson;
    Student *myStudent;

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

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



```
Example Main &
#include "OOP Header.h"
#include "pet.h"
#include "dog.h"
                                                    Output
#include "cat.h"
#include "mouse.h"
int main()
                                     These all need to be Pointers

→ for Dynamic binding
   Pet *buddy;
   Pet *bear;
   Pet *ben;
                                     This enables sub-type polymorphism
   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("<u>Ben</u>", 1, 0.5, "Mouse");
                                                                  OUTPUT
   buddy -> Speak();
                                                                  Buddy says Woof Woof!
   bear -> Speak();
                                                                  Bear says meoowww!
           -> Speak();
   ben
   fluffy -> Speak();
                                                                  Ben says squeak!
                                                                 Fluffy is speaking...
   fluffy-> Display();
   buddy -> Display();
                                                                             0
   bear -> Display();
ben -> Display();
                                                                 Buddy
                                                                             6
                                                                                  32.50
                                                                                             Dog
                                                                  Bear
                                                                             18 20.50
                                                                                              Cat
                                                                 Ben
                                                                            1 0.50
                                                                                             Mouse
   return 0;
```

```
Exercise

Which method is called? (assume classes below)

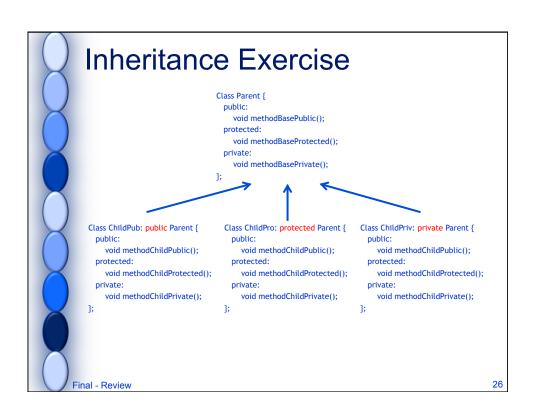
class Parent
{
    public:
        void method1();
        void method2();
        void method3();
        ...
};

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
                                         class Parent
I<sub>f</sub>/ create Parent object - Parent pointer
          ptrP1 = new Parent;
          ptrP2 = new Child;
                                         // create Childipbject - Parent pointer
          ptrC = new Child;
                                         // create Child objectoic hildtpointer;
                                                              virtual void method2();
          ptrP1 -> method1();
                                         // Parent class methodd()riethwoked;
          ptrP1 -> method2();
                                         // Parent class method2() is invoked
          ptrP1 -> method3();
                                        M Parent class method3() is invoked
          ptrP2 -> method1();
                                         // Parent class method1() is invoked
          ptrP2 -> method2();
                                        19 assic bilds pirblic metrod 2() is invoked
          ptrP2 -> method3();

√ Parent class method3() is invoked

          ptrC -> method1();
                                        // Child class method1() is invoked
                                        // Child class method2() is invoked (); // Parent class method3() is invoked
          ptrC -> method2();
          ptrC -> method3();
```



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

Final - Review

27

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?

Final - Review

• What is recursion? Process in which the result of each repetition is dependent upon R the result of the next repetition o T/F е Every Loop can be implemented Recursively C • What are the advantages and disadvantages of u recursion? r Advantages: Models certain algorithms most accurately; Results in shorter, simpler functions S Disadvantages: May not execute very efficiently • What is a base case? The simpler-to-solve problem is known as the base case 0 Recursive calls stop when the base case is reached n 29