Final Review

9 Pointers

9.1

Getting the Address

Address operator & returns address of variable

This operator also passes by reference in a function

Int main()

{

Int x = 5;

Reference(x);

Cout << x << endl;

}

Void reference(&x)

{

X + 1;

}

Console out prints 6 since passed by reference, the void function actually changes the variable.

9.2

Pointer Variables

Holds an address

Defining

Int \*ptr;

Int\* ptr;

Indirection

Indirection operator \*, will dereference a pointer after it’s created, allowing you to access the data pointed to.

9.3

Arrays and Pointers

Array names can be used as constant pointers

Pointers can be used as array names

* Adding actually adds the value specified times the size of the data type being referenced by the pointer
* \*(number + 1) adds one to address of number, the derfs
* \*number + 1 adds one to contents of first element in array

9.4

Pointer Arithmetic

++ and –

Integer + and – or += and -=

Pointer on point + and –

9.5

Initializing Pointers

Int myValue;

Int \*pint = &myValue;

Int ages[20];

Int \*pint = ages; // legal since ages in a array

Float myFloat;

Int \*pint = &myFloat //illegal since pint holds a int

9.6

Comparing Pointers

If one address comes before another in memory it is said to be less than.

* < == != >= <=

9.7

Pointers as Function Parameters

Using a pointer as a parameter works much like pass by reference since it gives access to the original argument

Pointer to constant

* Constant variable
* Changeable address
* Const double \*rates
* If we pass constant data into a regular pointer then error

Constant Pointer

* Constant address
* Changeable data
* Int \* const ptr
* When using a paramter, point is initialized with value passed in
* We cannot change the address after this point, but we can dereference and change the value
* We can successfully recall the function, and the constant pointer will point to different addresses

Constant Pointer to Constant

* Constant address
* Constant data
* Const int \* const ptr
* When used as a parameter we can recall the function using different arguments
* Other than that there is no way to change

9.8

Dynamic Memory Allocation

Variables may be created and destroyed while the program is running

Dynamically allocate – while program is running, create variables on the fly

Iptr = new int;

Create a pointer variable big enough to hold a integer

Iptr = new int[100]

Create a pointer variable big enough to hold a integer array of 100

What if we don’t create a pointer big enough?

Throwing an exception

Delete iptr;

Delete [] iptr;

Releases the memory

9.9

Returning Pointers

Should only be returned if

* They were passed into the function to begin with
* Are dynamically allocated

9.10

Smart Pointers

* Ability to automatically delete allocated memory that is no longer in use
* #include <memory>
* Unique\_ptr<int> ptr ( new int ) ;

11 Structured Data

11.1

Abstract Data Types

Defined by the programmer

Have their own range of data and operations

Composed of one or more primitive data types

11.2

Structures

struct PayRoll //uppercasefirst letter to distinguish between primitives

{

Variable declarations

};

11.3

Accessing members

Dot operator

structureVariable.memberVariable = something;

struct PayRoll

{

Int empNumber;

};

PayRoll employee;

Cin >> employee.empNumber;

Can only compare structures by members individually

11.4

Initializing a structure

Employee = {1120};

Since employee only has one member we only need to provide one

You can initialize as many or as few as you like, they have to be in order though

11.5

Arrays of structures

Struct BookInfo

{

String title;

String author;

};

BookInfo booklist[20];

booklist[5].title = ArtofWar;

Initializing

BookInfo booklist[20] = {

{ArtofWar, Sun Tzu},

{Jonathon Livingston Seagul},

{Harry Potter, JK Rowling}

};

11.6

Nested Structures

Double dot

Struct Costs

{

double wholesale;

double retail;

};

struct Item

{

string partNum;

string description;

Costs pricing;

};

Item widget;

widget.pricing.wholesale = 100.0;

widget.pricing.retail = 150.0;

strucutreVariable.memberVariable.nestedMemberVariable

11.7

Structures as Function Arguments

Can pass individual members

Can pass entire array

11.8

Returning a structure

Necessary to have a local structure variable to accept the data

11.9

Pointers to structures

Circle \*cirPtr = nullptr;

Circle myCircle = {10.0, 20.0, 314.159}

CirPtr = &myCircle;

Indirect access

Dot operator > \*

(\*cirPtr).radius = 10;

Or structure pointer notation

cirPtr->radius = 10;

Dynamic allocation

cirPtr = new Circle

cirPtr->radius = 10;

11.10

Notation

* Dereference a pointer to a structure

\*Dereference a pointer (possibly inside a structure)

11.1

Unions

All members occupy same memory area so they can only be accessed one at a time

Entire variable takes up only as much space as the largest data type used in the union

Only one member can hold a value at a time

Anonymous Unions

No tag

Members can be accessed without dot operator

**11.12**

Enumerated Data Types

Programmer defined data type

Consists of integer constants

enum TypeName {enumerations };

Cannot directly assign integers to them

workday = static\_cast<Day>(3);

Can compare enumerators

FRIDAY > THURSDAY

Anonymous

No name, cannot use data type to define a variable of this enum type

Math ops

Day day1, day2;

Day1 = TUESDAY;

Day2 = static\_cast<Day>(day1 + 1);

12 Advanced File I/O

12.1

Data Types

Ifstream – input file stream, used to read data from files to memory

* If doesn’t exist open fails

Ofstream – output file stream, used to create files and write data to them

* Truncates by default

Fstream – file stream. Used to create files, write and read from them

* Requires two arguments
* First, string containing name
* Second, file access flag to indicate the mode
* Fstream dataFile;
* dataFile.open(“info.txt”, iso::out);
* dataFile.open(“info.txt”, iso::in);
* File access flags
* App (append) preserve contents, write to end, create if doesn’t exist
* Ate, if files exists program goes to end, output may be written anywhere
* Binary, data written or read will be in pure binary
* In, data read from
* Out, data written to
* Trunc, if file exists contents will be truncated (deleted), default mode of ios::out
* Flags can be used together with | operator
* Ios:out alone will delete
* Ios:out | ios:in will preserve

End of file marker – always non printable character, most use control-z

* Automatically written when file is closed

12.2

File Output Formatting

Fixed – fixed-point notation

Setprecision(#) - # of decimal places

12.3

Passing Fstream to functions

Always pass by reference (parameter is the exact same)

Not pass by value (which is a copy)

12.4

Error Testing

Ios bits

* Ios::eofbit end of stream input encountered
* Ios::failbit attempted operation fails
* Ios::hardfail unrecoverable error
* Ios::badbit invalid operation attempted
* Ios::goodbit all above flags not set, indicates good operation

Member function

-eof() returns true of eofbit set

-fail() retruns true is failbit or hardfail set

-bad() returns true is badbit set

-good() returns true if goodbit set

-clear() when called with no arguments, clears all flags

Can also clear specific flag

-fstream file

-void showState (fsteam &file)

{

Cout << “file status” << endl;

Cout << “eof bit: “ << file.eof() << endl;

File.clear();

}

12.5

Reading and Writing Member functions

Getline(dataFile, str, ‘\n’);

* dataFile, name of file stream object
* str, string to read data into
* ‘\n’, delimiter of choice (can be excluded, n is default)

12.5

Get

* Reads a single character
* Streamobj.get(ch);

Put

* Writes a single character to file
* Streamobj.put(ch);

12.6

It is possible to have more than one file open at a time

Called a filter

12.7

Binary mode

* File.open(“stuff.dat” , ios::out | ios:binary);
* By default, files are open in text mode.

Write and read member functions

* fileObject.write(address, size);
* best to use sizeof(variable) for second argument
* writes the contexts of memory
* fileobject.read(address,size);
* reads information back into memory

Writing data other than char

* read write expect pointer to char
* must type cast when writing other types
* int x = 65;
* char \*ptr = nullptr;
* ptr = reinterpret\_cast<char \*>(&x), sizeof(x));

12.8

Creating Records with Structures

Field - individual piece of data

Records -made up of fields, complete set of data about a single item

Const int NAME\_SIZE = 51, ADDR\_SIZE 51, PHONE\_SIZE = 14;

Struct Info

{

Char name [NAME\_SIZE];

Int age;

Char address1[ADDR\_SIZE];

Char address2[ADDR \_SIZE];

Char phone[PHONE\_SIZE];

};

Info Person;

File.write(reinterpret\_cast<char \*>(&person), sizeof(person));

\*\*note because structures can use a mixture of data types, we should always use the \*\*ios::binary mode when opening a file to store them

12.9

Random Access Files

Seekp and seekg

* Seek put
* Output
* Seek get
* Input
* File.seekp(20L, iso::beg);
* 20L forces the compiler to treat the number as a long integer
* This statement moves the compiler to the position of byte number 20
* This is actually the 21 position since all numbering starts at 0
* The second argument is the call mode
* Designates where to calculate the offset from
* Iso::beg means the beginning of the file
* Beg, end, curr

Tellp and tellg

* Return, as a long integer, the current byte number of a file’s read and write position

Rewinding

-dataFile.clear();

-dataFile.seekg(0L, ios::beg);

12.10

Opening for input and output

* Files contents preserved
* Read/write initially placed at beginning
* File dne, then it’s created

13 Classes

13.1 Procedural and object oriented

Procedural

Steps

Object Oriented

Manipulating data with functions

Object

Attributes / member functions

Data / functions (methods)

Encapsulation

Combination of data and functions

Data hiding

Protecting data from code outside of the object

13.2 Intro to Classes

class Classname

{

Declaration

};

Private by default, meaning code outside the class cannot access

Const with member functions

* Specifies the function will not change data stored in the calling object
* Double getWidth() const;

Member functions

* Prototype in class
* Class Rectangle
* {
* Public:
* Void setWidth(double);
* };
* Definition elsewhere with scope resolution operator
* Void Rectangle::setWidth(double w)
* {
* Width = w;
* }
* ReturnType ClassName :: functionName (Parameter List)

Accessor and Mutator

* Accessor, gets a value from another class but does not change
* Mutator, set the value

13.3 Instance of a class

ClassName objectName;

* Rectangle box;
* box.setWidth(12.7);
* we used the dot operator with a setter function to change a variable indirectly

Sale data

* dependent on other data that could possibly change without updating resulting in stale data

Points to objects

* Rectangle myRectangle;
* Rectangle \*rectPtr = nullptr;
* rectPtr = &myRectangle;
* rectPtr->setWidth(12.5);
* delete rectPtr;

**Smart objects**

13.4

13.5

13.6

13.7

Constructors

- Automatically called when a class object is created

- Member function with same name as class

- class Demo

- {

- public:

- Demo();

- };

- Demo::Demo()

- {

- cout << “Welcome to the constructor!\n”;

- }

Default constructor

No parameters

Parameters with default arguments are allowed.

* (double cost = 0.0) vs (double cost)

Dynamic allocation

Rectangle \*rectPtr = nullptr; // constructor not called yet

rectPtr = new Rectangle; // at this point constructor is called

13.8

13.9

Destructors

- class Demo

- {

- public:

- Demo();

- ~Demo();

- };

- Demo::Demo()

- {

- cout << “Welcome to the constructor!\n”;

- }

- Demo::~Demo()

- {

- cout << “The destructor is running!\n”;

- }

13.10

13.11

13.12

13.13

13.14

13.15

13.16

13.17

15

15.1

Inheritance

New class based on old class

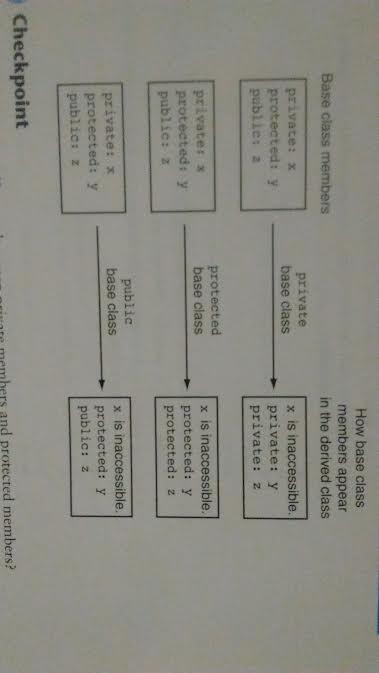
Gets all info besides constructors and destructor

- class Derived ClassName :: (base access specification) BaseClassName

- class FinalExam :: public GradedActivity

15.2

Protected members can be accessed by the base class, private members cannot



15.3

Constructor

Base class called first

Destructor

Base class called last

Arguments to Constructor of derived class

In class prototype

* Derived Class () : Base Class ()

Definition

DerviedClassName::DerivedClassName(parameter list): BaseClassName (argument list)

15.4

Redefining

- A derived class with a same function name as the base class

Overloading

* Same function name with different parameters

15.5

15.6

15.7

15.8

16

19