1ST Class – Monday, May 26, 2014 (Closed for Holiday: Memorial Day)

1ST Class – Wednesday, May 28, 2014

Text: C++ How To Program, 7th Edition by P.J. Deitel, H.M. Deitel

<http://www.deitel.com/Books/C/CHowtoProgram7e/tabid/3472/Default.aspx>

<http://www.amazon.com/C-How-Program-7th-Edition/dp/0136117260/ref=sr_1_1?ie=UTF8&qid=1401300699&sr=8-1&keywords=0136117260>

Instructor: Shams Al-Ajrawi – [sal-ajrawi@coleman.edu](mailto:sal-ajrawi@coleman.edu)

Assignments on WebClass

**PowerPoint – “C++ (COM203) Chapters 2&3.ppt”**

<http://www.thinkingparallel.com/2007/03/06/c-vs-c-a-checklist-from-a-c-programmers-point-of-view/>

New Project – Other Language – C++ - Empty Project – Add – New Item – C++ File (.cpp)

Multiple data items can be input by concatenating stream insertion (<<) operators.<<

std::cout << "Hello" << “Welcome” << \n;

Precedence: () \* / % + -

2ND Class – Thursday, May 29, 2014

Text:

Three stages of Compilation:

1. Preprocessing – preprocessor directive (strips white space), only in memory.

2. Compilation – into object code (machine code), saves to disk. Object code – machine language for hardware dependent and

3. Linking combines the object code – executable file. Others come from the Run-Time Library

Can use #include <<stdio.h>> instead of the use namespace std;

**::** – used to define functions inside a class, e.g. std::cout << “”Hello”;

Class – user defined type. Has data and methods.

Define the object with classname objectname (compared to C#, there is no instantitation with new )

Use dot notation or arrow notation (chapter 8). Only single inheritance.

Accessors: public, private, protected

Member functions of a class can be written in the class or outside the class as void classname :: int area();

Anatomy of a C++ Program:  
Program header comment, preprocessor directives (if any)

Object Initialization

1. By assignment – the worst way because it only works on public data members.   
   No control over the operations of the data members:

#include <iostream.h>

class circle

{

public:

double radius;

};

Int main()

{

Circle c1; // Declare an instance of the class circle

C1, radius = 5; // Initialize by assignment

}

1. By a Public Member Function using set and get:
2. By constructor: The name of the constructor is the same name as the class. Three ways:
3. Default constructor (no args), void constructor
4. Copy constructor,
5. constructor with parameters.constructor (type arg)

Multiple Files – organized into modules

Header – .h files, contain the prototype of the function, e.g. void are (int ) //no parm just a type

Functions – .cpp files

UML of Example:

GradeBook

+ displayMessage(string coursename)

Repetition statements

for (initial value; condition; increment)

while

do while

Week1

Type Examples:

**fig02\_01, fig02\_03, fig02\_04, fig02\_05, fig02\_13**

~~ex02\_05~~

Type Examples:

**fig03\_01, fig03\_03,** **fig03\_05, fig03\_07**

Week2

Type Examples:

**fig03\_07, fig03\_09, fig 03\_10, fig03\_11, fig 03\_12, fig 03\_13, fig03\_15, fig 03\_17**

Type Examples:

**fig04\_10**

3rd Class – Wednesday, June 4, 2014

**Functions** – Chapter 6 (Page 207), methods and procedures in other languages.

: Signature of the function:

Return type – Name – Arguments

: User-defined functions are programmer-defined functions.

: Passed by reference and passed by value:

& - by reference, not by value, e.g. void duplicate ( int &a, int &b, int &c ) (page 241)

Fig 6.19 (page 242) Passing arguments by value and by reference.

: e.g. func ( int x, int y=2). func(12)… 12/2=6, func(12,6)… 12/6=3

: Function overloading – using different parameters by either different types or quantity.

: Template (Page 251) – To make the overloading more generalized.

template < class T >

T sum ( T a, T b )

{

T result; // define

result = a + b;

return result;

}

Void main()

{

Int x=1, y=2, h; // define

double c=2.0, d=0.5, k;

h = sum<int> (x, y);

k = sum<double> (c, d);

cout << h << endl;

cout << k << endl;

}

: Comparing the scope, inside a function, of updating global variables to declared local variables.

: Namespace name {…}, using namespace name;

: Recursion – a factorial function

// Factorial calculator

#include <iostream>

Using namespace std;

Long factorial (int a) // function

{

If (a>1)

Return (a\*factorial(a-1));

Else return 1;

}

int main()

{

int number =9;

Cout << number << “!=” << factorial(number);

return 0;

}

**Pointers** – Chapter 8 (Page 345), methods and procedures in other languages.

4th Class – Monday, June 9, 2014

**Arrays and Vectors** – Chapter 7 (Page 282), 3 ways:

Declare an Array with it’s size as a constant integer to initial the Array: const int size=5;

int Array[5]

and using a ‘for’ loop to initialize the Array’s elements.

for (int i=0; i<size; i++)

cin >> a[i];

int Array[5] = {1, 2, 3, 4, 5}

: Automatic –

#include

void StaticArray (void);

void AutomaticArray (void);

cont int size=3;

void main()

{

Cout “First call to each function”;

StaticArray();

AutomaticArray();

}

void StaticArray (void);

{

Static int Array[size];

Cout << “ values entering static function”;

for (int i=0; i<size; i++)

cout << “Array [” << i << “]=” << Array[i];

cout << “ values exiting static function”;

for (int j=0; j<size; j++)

cout >> “Array [” << j << “]=” << Array[j];

}

void AutomaticArray (void);

{

int Array[3]={1,2,3};

Static int Array[size];

cout << “ values entering automatic function”;

for (int i=0; i<size; i++)

cout << “Array [” << i << “]=” << Array[i];

Cout << “ values exiting automatic function”;

for (int j=0; j<size; j++)

cout >> “Array [” << j << “]=” << Array[j];

}

*Output of 1st call:*

*0 0 0*

*5 5 5*

*1 2 3*

*6 7 8*

*Output of 2nd call:*

*5 5 5*

*10 10 10*

*1 2 3*

*6 7 8*

*Output of 3rd call:*

*10 10 10*

*15 15 15*

*12 3*

*6 7 8*

Accessing values from an Array

: for… total += Array[i];

Passing Arrays to Functions

void PrintArray ( const int [], int); // function prototype has no names of the parameters

void PrintArray ( const int a[], int length); // function

{

for (int i=0; i<length; i++)

cout << a[i]; << endl;

}

void main()

{

int FirstArray[]= {1,2,3}

int SecondArray[]= {4,5,6,7}

PrintArray(FirstArray,3);

PrintArray(SecondArray,4);

}

: Linear Search

int LinearSearch (const int [], int, int) // function prototype passes, array name, length, key to find

int LinearSearch (const intArray [], int size, int) // function passes, array name, length, key to find

const int size=5;

void main()

{

intArray[size]= {5, 10, 3, 16, 2}

for (int i=0; i<length; i++)

{

cout << Array[i];

cout << “Enter Key search”;

cin >> keySearch;

}

int index = LinearSearch(Array, Size, keySearch);

if (return > -1)

cout << “Found the Key in the index: ” << index;

else

cout << “Key not found. ”;

}

// function passes, array name, length, key to find

int LinearSearch (const int A[], int s, int k) {

for (int j=0; j<length; j++)

{

cout << Array[i];

cout << “Enter Key search”;

cin >> keySearch;

}

int index = LinearSearch(Array, Size, keySearch);

if (a[j] == k)

return j;

else

return -1;

}

: Insertion Sort. Sort in increasing order {7,2,9,6,}

The first number is compared to each number, starting at the first index and places the largest in the hole.

The next number is compared to the each number, starting at the second index and places the largest in the hole.

Repeat…

int value;

int a[5];

for (int i=1; i<5; i++)

value = a[i];

holes = i;

while ( (holes>0) && (a[holes-1] > value) )

{

a[holes] = a[holes – 1];

holes --;

}

a[holes] = value;

: Bubble Sort

As elements are sorted they gradually “bubble” (or rise) to their proper location. Two elements are compared, swapped if out of order, one pair at a time.

93057 – original to sort into descending order…

93570 – 1st pass

95730 – 2nd pass

97530 – 3rd pass

97530 – 4th pass

for (int i=0; i<length; i++) *// for all the elements of the array*

{

for (int j=0; j<length; j++) *// for the element that is the rising bubble*

{

*// Swamp elements out of order with a third, temp holder element*

int temp;

If ( a[j] > a[j+1] )

{

temp = a[j];

a[j] = a[j+1];

a[j+1] = temp;

}

cout << a[j];

}

}

const int row = 4;

const int column = 3;

int a[row][column];

for (int r=0; r<row; r++) *// for rows*

{

for (int c=0; c<column; c++) *// for the element columns*

{

cout << a[r][c];

cout << a[r][c];

}

cout << endl;

}

(Missed) Check On Learning – Quizzes on Webclass

The std::endl stream manipulator\_\_\_\_\_\_\_\_.

The correct answer is: outputs a newline and flushes the output buffer.

Calling a member function of an object requires which item?

The correct answer is: The dot operator

Which of the following data types can be used to represent integers?

The correct answer is: char, long, short; all of the above

Consider the execution of the following for loop  
    for (int x = 1; x < 5; increment )  
       cout << x + 1 << endl;

If the last value printed is 5, which of the following might have been used for increment?

Select one:

a. x++

b. x += 1

c. ++x

d. Any of the above

### Feedback

The correct answer is: Any of the above

5th Class – Wednesday, June 11, 2014

**Pointers** – Chapter 8 (Page 345), methods and procedures in other languages.

Reference operator ‘&’: can obtain the address of a variable by preceding the name with the ampersand ‘&’.

y = &x, z= y, z = 3

// another example …

y – 1775

a – 1975

…

int y = 3, a = 5;

p1 = &y;

p2 = &a;

p1 = 10;

cout << a, y, p1, p2;

- - - - -

a+5

y=3

p1=~~1775~~ 1975

p2=1975

Assign a variable’s value to **the pointer’s ‘content’ value:**

\*myPointer = 5;

Assign a **pointer’s ‘address’ to the variable’s value:**

myPointer = &secondValue ; // Address ‘1ff5’

\* the asterisks is an overloaded operator, for two uses:

1) a ‘de-reference operator’ for content

2) to declare the pointer of a variable, to obtain the address later with the &.

// another example …

firstValue – 1f23

secondValue – 1ff5

…

int firstValue, secondValue;

int \*myPointer;

int pointer;

myPointer = &firstValue ; // Address ‘1f23’

\*myPointer = 5; **// set the pointer’s content value**

pointer = myPointer; // Address ‘1f23’

myPointer = &secondValue ; // Address ‘1ff5’

\*myPointer = 10; **// set the pointer’s content value**

Cout << “firstValue: ” << firstValue; // 5

Cout << “secondValue: ” << secondValue; // 10

Cout << “myPointer: ” << myPointer; // Address ‘1ff5’

Cout << “pointer: ” << pointer; // Address ‘1f23’

// another example …

myVar = 25 – 1776

…

int \*a;

int myVar = 25;

a = &myVar; // 1776

\*a = 10;

Cout << a << \*a << myVar: // 1776, 10, 10

// another example …

firstValue – 1f2f2

secondValue – 1ff9

…

int firstValue, secondValue;

int \*myPointer;

myPointer = &firstValue ; // Address ‘1ff2’

\*myPointer = 10; **// set the pointer’s content value**

myPointer = &secondValue ; // Address ‘1ff9’

\*myPointer = 20; **// set the pointer’s content value**

Cout << “firstValue: ” << firstValue; // 5

Cout << “secondValue: ” << secondValue; // 10

With the \* you refer to the content. ‘Dereference’ to present the content.

\*p1 = \*p2;

Without the \* you refer to the address

p1 = p2;

// another example …

myArray[0] – 0001

myArray[1] – 0002

myArray[2] – 0003

myArray[3] – 0004

myArray[4] – 0005

…

int myArray[5];

int \*myPointer;

myPointer = myArray;

{

int numbers[5];

int \*p;

p = number; // 0001, \*p = 10

p++;

\*p = 20; // 20

p = &number[2] // 0003, \*p = 30

p = number + 3 // 0005, \*p = 40

p = number];

\*( p + 4 ) = 50 // 0001 + 0004 = 0005 = 50

}

**: Pointer Initialization:**

1776 – x

…

int \*x = &y; // 1776

int \*z = &x; // z is a (deep copy) of x (both are pointing to the same thing)

**: Pointer arithmetic:**

char mychar;

short mychar;

long mychar;

…

++mychar; // increments 1 bytes

++mychar; // increments 2 bytes

++mychar; // increments 4 bytes

…

\*p++ // increment pointer, and dereference un-incremented address.

\*++p // increment pointer, and dereference incremented address.

++\*p // dereference pointer, and increment the value it points to.

(\*p)++ // dereference pointer, and post-increment the value it points to.

0001 – x

…

0003– y

…

Int x;

Int y = 10;

Const int \*p = &y; // 0003

X = \*p;

\*p = x; // Error when we try to modify the content value because we declared the pointer as a constant

6th Class – Thursday, June 11, 2014 (Lab)

**Fetick\_Project2-TicTacToe**

– References:

1. Webclass – Week 4 – Project2B TicTacToe Specification
2. Student Manual (Lesson 5: Tic- Tac- Toe, page 12; Project 1, page 34; , page 74)

**Project 1 (a)** 2 POINTS

Write an algorithm for the game TicTacToe that describes ***in detail*** the steps as to how the game is played (follow the instruction provided by your instructor).

**Pseudocode**

**Pseudocode** (or "fake" code) is an artificial and informal language that helps a programmer to develop algorithms without having to worry about the strict details of C++ language syntax. The pseudocode we present here is particularly useful for developing algorithms that will be converted to structured portions of C++ programs. Pseudocode is similar to everyday English; it is convenient and user friendly, although it is not an actual computer programming language.

Pseudocode does not execute on computers. Rather, it helps the programmer "think out" a program before attempting to write it in a programming language, such as C++.

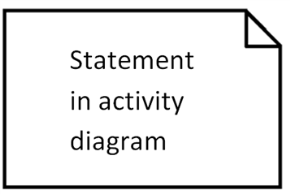
The style of pseudocode we present consists purely of characters, so programmers can type pseudocode conveniently, using any editor program. The computer can produce a freshly printed copy of a pseudocode program on demand. A carefully prepared pseudocode program can easily be converted to a corresponding C++ program. In many cases, this simply requires replacing pseudocode statements with C++ equivalents.

Pseudocode normally describes only **executable statements**, which cause specific actions to occur after a programmer converts a program from pseudocode to C++ and the program is run on a computer. Declarations (that do not have initializers or do not involve constructor calls) are not executable statements. We typically do not include variable declarations in our pseudocode. However, some programmers choose to list variables and mention their purposes at the beginning of pseudocode programs.

We now look at an example of pseudocode that may be written to help a programmer create the addition program.

1. Prompt the user to enter the first integer
2. Input the first integer
3. Prompt the user to enter the second integer
4. Input the second integer
5. Add first integer and second integer, store result
6. Display result

Activity Diagram for the above pseudocode:



The sequence structure is built into C++. Unless directed otherwise, the computer executes C++ statements one after the other in the order in which they are written that is, in sequence.

The Unified Modeling Language (UML) **activity diagram** above illustrates a typical sequence structure in which calculations are performed in order. C++ allows us to have as many actions as we want in a sequence structure.

Activity diagrams are part of the UML. An activity diagram models the **workflow** (also called the activity) of a portion of a software system. Such workflows may include a portion of an algorithm, such as the sequence structure. Activity diagrams are composed of special-purpose symbols, such as **action state symbols** (an elongated circle), **diamonds** (used usually for decisions/condition) and **small circles** (known as start and end states); these symbols are connected by **transition arrows**, which represent the flow of the activity.

Like pseudocode, activity diagrams help programmers develop and represent algorithms, although many programmers prefer pseudocode. Activity diagrams clearly show how control structures operate.

Consider the above activity diagram. It contains several **action states** that represent actions to perform. Each action state contains an **action expression** e.g., "prompt for an integer" or "display result" that specifies a particular action to perform. Other actions might include calculations or input/output operations. The arrows in the activity diagram are called transition arrows. These arrows represent **transitions**, which indicate the order in which the actions represented by the action states occur the program that implements the activities.

The **solid circle** located at the top of the activity diagram represents the activity's **initial state** (or the start state)the beginning of the workflow before the program performs the modeled activities. The solid circle surrounded by a hollow circle that appears at the bottom of the activity diagram represents the **final state** (or the end state)the end of the workflow after the program performs its activities.

The diagram also includes rectangles with the upper-right corners folded over. These are called **notes** in the UML. Notes are explanatory remarks that describe the purpose of symbols in the diagram. Notes can be used in any UML diagram not just activity diagrams. A **dotted line** connects each note with the element that the note describes. Activity diagrams normally do not show the C++ code that implements the activity. We use notes for this purpose here to illustrate how the diagram relates to C++ code. For more information on the UML visit [www.uml.org](http://www.uml.org).

if ( grade >= 60 )

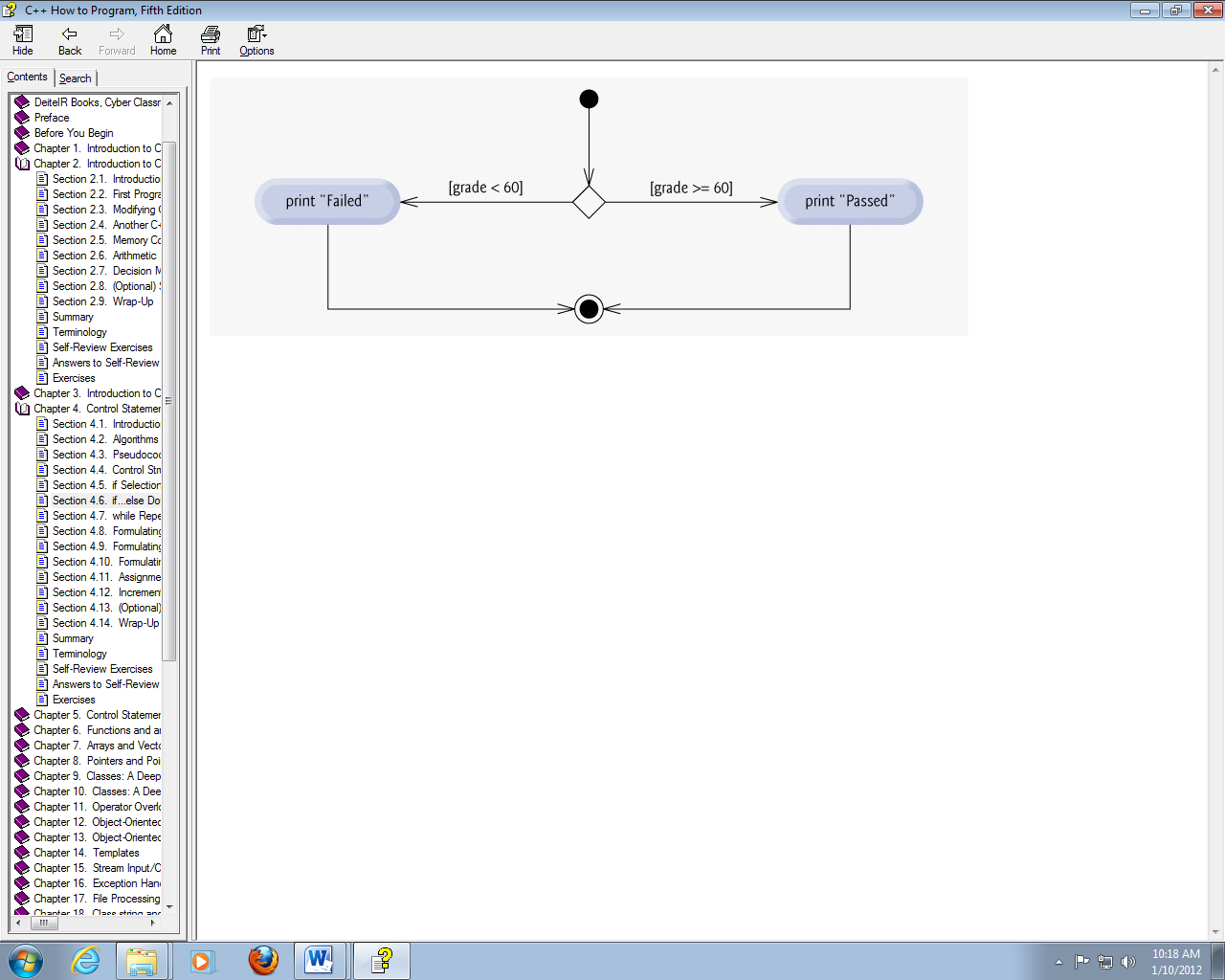
Statement in activity diagram

Statement in activity diagram

cout << "Passed";

else

cout << "Failed";



The following diagram illustrates the single-selection if statement. It contains what is perhaps the most important symbol in an activity diagram the diamond or **decision symbol**, which indicates that a decision is to be made. A decision symbol indicates that the workflow will continue along a path determined by the symbol's associated **guard conditions**, which can be true or false. Each transition arrow emerging from a decision symbol has a guard condition (specified in square brackets above or next to the transition arrow). If a particular guard condition is true, the workflow enters the action state to which that transition arrow points. In the above diagram, if the grade is greater than or equal to 60, the program prints "Passed" to the screen, then transitions to the final state of this activity. If the grade is less than 60, the program immediately transitions to the final state without displaying a message

…

*Refer to Pages 124 for the pseudocode and to Pages 125 – 128 in the textbook for the program*

…

On page 74:

**Project 1 (b) 3 POINTS**

Using the algorithm given above write a fully developed pseudo code for the **run** function for the TicTacToe game based on the description given above. Check if the description is sufficient. If not, what else needs to be done?

7th Class – Monday, June 16, 2014

Sorting – Chapter 19, page 784.

: Selection Sort

3 5 2 1 4

1 5 2 3 4

1 2 5 3 4

1 2 3 5 4

1 2 3 4 5

Start a search at the 1st location for the smallest value and switch it with the 1st location.

Start a search at the 2nd location for the smallest value and switch it with the 2nd location.

Start a search at the 3rd location for the smallest value and switch it with the 3rd location.

void selectionSort (int \*const, int \*const);

void swap (int \*cont, int \*const)

void main()

{

Const int arraySize;

int const a[arraySize] = {3,5,2,1,4};

for (int i = 0; i < arraySize; i++);

cout << a[i];

selectionSort (a, arraySize);

for (int j = 0; j < arraySize; i++);

cout << a[j];

}

void selectionSort ( int \*const array, int \*const arraySize);

{

int smallest;

for (int i = 0; i < arraySize; i++);

smallest = i;

for (int index = i+1; index < arraySize; index++);

if (array[index] < array[smallest])

smallest = index;

swap ( array[smallest], array[i] );

}

void swap ( int \*const element1Ptr, int \*const element2Ptr);

{

int hold = \*element1Ptr;

\*element1Ptr = \*element2Ptr;

\*element2Ptr = hold;

}

s elements are sorted they gradually “bubble” (or rise) to their proper location. Two elements are compared, swapped if out of order, one pair at a time.

**Vectors** – Chapter 7 (Page 323, 788, 934)

5 5 5 5

vector <int> first (4, 5)

vector <int> third (second.begin(), second.end() );

vector <int> first (4, 5)

and using a ‘for’ loop to initi

we want this to copy the contents of the 2nd vector to the 3rd vector:

Use ‘push\_back’ to add an element at the end, which changes its size during runtime.

Use ‘pop\_back’ to minus an element at the end, which changes its size during runtime.

Why is a vector better than an array:

1) Can change the size of the vecor at runtime with push\_back and pop\_back.

2) If we have two vectors, we can assign properties of one vactor to another.

Fig. 7-25.

8th Class – Wednesday, June 18, 2014

“A Deeper Look (Part 2)”

In the header file Gradebook.h

#ifndef GRADE\_BOOK\_H // an arbitrary label GRADE\_BOOK\_H is used to define the header

#define GRADE\_BOOK\_H

…

#endif

…

: Destructor (Pages 416-418 figs 9.11, 9.12, and 9.13)

CreatreAndDestroy.h

#ifndef CREATE\_H // an arbitrary label GRADE\_BOOK\_H is used to define the header

#define CREATE\_H

…

CreatreAndDestroy( int, string ); // constructor

~CreatreAndDestroy(); // destructor

#endif

…

CreatreAndDestroy.cpp

…

CreatreAndDestroy::CreatreAndDestroy( int ID, string messageString); // constructor

…

CreatreAndDestroy::~CreatreAndDestroy(); // destructor

…

Fig09\_13.cpp

…

void Creatre ( void ); // prototype

CreatreAndDestroy first( 1, “(global before main)” ); // global object

int main

{

CreatreAndDestroy second( 2, “(local automatic in main)” );

static CreatreAndDestroy third( 3, “(local static in main)” );

create(); // call to create objects

CreatreAndDestroy fourth( 4, “(local automatic in main)” );

} // end main

…

void create( void )

{

CreatreAndDestroy fifth( 5, “(local automatic in create)” );

static CreatreAndDestroy sixth( 6, “(local static in create)” );

CreatreAndDestroy seventh( 7, “(local automatic in create)” );

}

Objects:

1 constructor runs (global before main)

**main function begins**

2 constructor runs (local automatic in main)

3 constructor runs (local static in main)

**create function begins**

5 constructor runs (local automatic in create)

6 constructor runs (local static in create)

7 constructor runs (local automatic in create)

**create function ends**

7 destructor runs (local automatic in create)

5 destructor runs (local automatic in create)

**main function resumes**

4 constructor runs (local automatic in main)

**main function ends**

4 destructor runs (local automatic in main)

2 destructor runs (local automatic in main)

6 destructor runs (local static in create)

3 destructor runs (local static in main)

1 destructor runs (global before main)

Webclass quiz was reset, do from 9:45-12:00.

9th Class – Monday, June 23, 2014

“A Deeper Look (Part 2)”

Constant and constant member function

Page 486 - Two kinds of constants…

- Constant function: void print() const;

- Constant object: const int increment;

Object: Function:

Ok – const obj

Ok – obj func

**Error** – const non const

Ok – const const

Ok – non const obj const func

Ok – non const obj non-const func

Page 430-434. The ‘friend’ function

The prototype of the friend function

A friend declaration in the class allows functions outside the class,   
to access the private member variables inside the class.

1st should be a reference to the class

Page 445-448. The ‘this’ pointer ….

3 ways to access the variable

X = 12

this->x = 12

(\*this).x = 12

Page 449-453. Using the ‘this’ pointer ….

Cascading function calls:

t.setHour( 18 ). setMinute( 30 ). setSeconds( 22 );

…

t.setTime( 20, 20, 20 );

Composition:

Need to declare 2 headers, 2 classes, and need to implement

That contains objects of the other classes.

Static class members:

Static data member.

Data abstraction and information hiding:

10th Class – Monday, June 23, 2014

Composition: Player – Weapon - Enemy

“A Deeper Look (Part 2)”

Presentation – “Operator Overloading”

Chapter 11.

11th Class – Monday, June 25, 2014

Midterm Quiz

Midterm Project

In the header file Gradebook.h

#ifndef GRADE\_BOOK\_H // an arbitrary label GRADE\_BOOK\_H is used to define the header

#define GRADE\_BOOK\_H

…

#endif

12th Class – Monday, June 30, 2014

**Inheritance Presentation** (Chapter 12)

Moving from the general to the specific.

4 types of Inheritance:

**Direct**

Base <- derived

**Indirect**

Base <- derived1 <- derived2

**Single**

Base <- derived

**Multiple**

Base1, Base2 <- derived

…

is-a

has-a

…

note: members are declared private by default.

…

base

**base-class**

public

private

protected

**derived-class**

public, protected

private , private

protected, protected

“multiple indirect inheritance” - problem

Class assignment – creature, monster (groups me with Mitch and Steven.)

13th Class – Wednesday, July 02, 2014

**Inheritance** (Review)

Inheritance is a way to form new classes from classes that have already been defined.

…

**Polymorphism** (Chapter 13) (lab3.pptx)

Polymorphism is a way to perform different operations on the same message.

…

Override inherited functions in subclass, i.e. lion::sleep() of animal::sleep()

…

Pointers to derived classes.

…

Virtual Functions are declared in the base-class and redefined (overridden) in the derived-class.

…

Base ob(10), \*p;

derived d\_ob(10);

…

1. declare objects for the base class

2. create pointer to the object

3. derived obj

4. derived \*ptr

Can:

base obj.baseFunc

derived obj.derivedFunc

base ptr -> base func

base ptr -> base obj

derived ptr -> derived obj base ptr = & obj base ptr

derived ptr -> base func derived ptr = & obj derived ptr

Dynamic Binding – Use the pointer operator =>

– done during runtime, using object with .dot operator to static function

Static Binding – Use the .dot operator

14th Class – Monday, July 07, 2014

**Data Structures**

Code was put on the R drive under COM203/Shams Data Structure/LinkedList.

Learning Project1

linear and non-linear

…

Static – 3 types: arrays, multi-dimensional arrays, structs

Dynamic – 4 types: linked-lists, queues, stacks, trees

Linked-lists: Data, next node

Class node // contains pointers of the sane type

n – new node, for traversing

h – head node

t – tail

The disadvantage of an array versus a linked-list are:

- have to allocate memory of array size for storage

- inefficiency of excessive memory allocation and runtime

- more difficult to insert and delete nodes

“Self-referential class”

4 steps:

1. n = new node; // make a new node

2. n->data = 2; // set the value in the data location

3. t->next = n; // set the connection for the next node

(break a connection and set new connections)

4. t = t->next; // move the next node pointer to the tail

cout << n->data << "--->";

Project 2

1st class: create the node

2nd class: to create the link-list

Stacks

Project 3

…

Insertion to and removal from, only the top

With the compiler it has a program counter ‘PC’

Puts it on the stack with

‘push PC;’ to retain the address in the main() for the program to return to

…

e.g. push 5; push 10; pop 10;

~~10~~ push, pop

5 push

PC

15th Class – Wednesday, July 09, 2014

**Data Structures – Queues and binary search trees**

Section 20.6 Queues – Wait in line – FIFO – Dynamic data structure

Insert – push – enqueue (front)

Delete – pop – dequeue (rear) – delete from the head

To implement a queue with a Linkedlist

- Define two classes

– one for Data

– one for Pointer

To implement a stack with a Linkedlist

- push = insert to the top

- pop = remove from the top (can’t use dequeue from the rear)

Looked at the code in Project4(Queues).sln

Section 20.7 Binary Trees

Nodes contain two links (none, one, both which may be null)

Root node is the first node.

Value of left child node < “less than” node

Value of right child node > “greater than” root node

Example:

30 10 20 40 50

Traverse the binary search tree

- 1. In-order. L-Root-R, from the root, start from the left and go to the right:

10 20 30 40 50

- 2. Pre-order Root-L-R, from the root:

30 10 20 40 50

- 3. Post-order L-R-Root,

20 10 50 40 30

Did Chapter 20 Data Structure, question 5, page 844.

Looked at the code in Project5(BinarySearch).sln

16th Class – Monday, July 14, 2014

**Project 4 Hangman**

**Files**

Code – Random Access Files

1 - Bit

8 bits - Byte, char

Chars – fields

Fields – Record

Record – files

Chapter 15. Fig. 15-2 Stream I/O

Out File stream – ofstream

ofstream objName (“ abc.tpf”, ios::out);

ofstream objName (“ abc.tpf”, ); // ios default is out

In File stream – ifstream

ifstream objName (“ abc.tpf”, ios::in);

ifstream objName (“ abc.tpf”, ); // ios default is in

(filename, file – open mode)

1. Create file, write to file.

a. include the library

b. declare using namespace

using std::cerr;

using std::cin;

using std::cout;

using std::endl;

using std::ios;

need sizeod, fixed length

of – specific stream

f – general stream

point to position of data in a file, use:

seek put-> ofstream

seek get -> ifstream

// converts int to char

outPlayer.write( reinterpret\_cast< const char \* >( &player ),

inPlayer.read( reinterpret\_cast< char \* >( &player ),

Need sequential file in Hangman project.

Create file and read file.

? 65 shams 98

? 87 gfdg 9

?

toupper();

17th Class – Wednesday, July 16, 2014

**Chapter 16. Exception Handling**

Exception, Exception Handler,

Base Class Exception – Library <Stdexception>

cerr<< e.what() << endl; // virtual function what is from the base class exception

…

ExceptionHandling Program…

...

class DivideByZeroException: public runtime\_error

…

Enter 2 numbers(enter ctrl+z to exit):

100 7

Quotient is: 14.2857

There are not exceptions, This should not print if an exception occurs

Enter 2 numbers(enter ctrl+z to exit):

100 0

Exception Occurred: Attempted to divide by Zero

Enter 2 numbers(enter ctrl+z to exit):

Convert the result to double from dividing integers:

return static\_cast<double>(numerator)/denominator;

Page 704 – Fig 16.10 Standard Library exception classes

Exception –

runtime\_error, logic\_error –

…

18th Class – Monday, July 21, 2014

Ogre Project – Code

…

Pong, Ball

I have a problem…

I installed the OgreAL library but the Pong program has errors not finding it.