**Lecture 15 - Chapter 7: Pointers – Wed Oct 11 or Thurs Oct 12**

**Announcements**

Reading:

* Chapter 7

Assignments:

* Assignment #5 graded
* Due: Assignment #6- due on **Oct 18** (MW class) or **Oct 19** (TR class) **(no late assignments accepted)**

Exam #2:

* Monday Oct 23 (MW class) – Tuesday Oct 24 (TR class) (chapters 1 – 7)

**Today’s Goals**

1. Assignment #5 Feedback
2. Passing Arguments to Function by Reference

**Assignment #5 Feedback 5 min**

**Pseudocode**

* A lot of people wrote next to nothing in terms of pseudocode!
* Pseudocode is part of each assignment.
* The purpose of pseudocode is to:
  + Think through your design without getting caught up in syntax and coding
  + Determine where there are problem areas before you even code them
  + To reduce the amount of time it takes to code
  + To reduce the number of bugs
* This is **NOT** pseudocode

**Function displayConcertMenu**

print different concert options

Says nothing about range of values, process for validating input

* This is pseudocode

**Function: displayConcertMenu()**

Display concert menu for the 3 groups (Rolling Stones, Imagine Dragons and U2)

Prompt user to select a group 1, 2 or 3

while the user input is not valid

Prompt the user to enter a valid number between 1 and 3

Return selected group

**Commenting**

* Functions should have a comment above the function explaining purpose!
* Functions should have comments within the function as well!

// Displays the sections and prices based on the concert customer wants to see.

// Prompts user for desired section and returns selected section

**int** **getSection** (**int** whichConcert) {

// Function code goes here

// This code should also be commented!

}

**Clean Up Code**

* Remove debugging statements
* Lots and lots of broken windows!
* Clean up the warnings, your code should be warning free!
* Lots of warning regarding returns on functions
  + “control reaches end of non-void function - indicates
    - there is a return type but did not include a return statement in function
    - there is a return type but its inside if statement or loop

**int** **displayConcertMenu**(**void**) {

**int** whichConcert;

bool processing = true;

**while** (processing == true) {

**printf** ("Which concert would you like to attend?\n");

**scanf**("%d", &whichConcert);

**if** (whichConcert == ROLLING\_STONES) {

processing = false;

**return** whichConcert;

}

**else** **if** (whichConcert == IMAGINE\_DRAGONS) {

processing = false;

**return** whichConcert;

}

**else** **if** (whichConcert == U2) {

processing = false;

**return** whichConcert;

}

}

}

* This code violates structured programming approach because:
  + It contains **return** statements in while loop and inside if statements
  + This causes early exit from these structures
* This code produces a warning because compiler thinks there is a possibility that a **return statement** will never be executed.
* If you write code that does **not** violate structured programming – the warning goes away and the code is cleaner, easier to understand, contains no surprises.

**int** **displayConcertMenu**(**void**) {

**int** whichConcert;

bool processing = true;

**while** (processing) {

**printf** ("Which concert would you like to attend?\n");

**scanf**("%d", &whichConcert);

**if** (whichConcert == ROLLING\_STONES ||

whichConcert == IMAGINE\_DRAGONS ||

whichConcert == U2) {

processing = false;

}

} // while

**return** whichConcert;

}

**Functions**

* Confusion about passing values into methods and using those values
* **Incorrect approach:**

**int** **main**(**void**) {

**int** whichConcert = displayConcertMenu();

int numSeats; You should prompt user inside

**printf** ("How many seats for the concert? "); the function **getNumberOfSeats**

getNumberOfSeats(whichConcert); Prompting for the number of seats,

and checking if valid value is

} the task of the **getNumberOfSeats** function! This removes that

functionality from main and puts

it in one location to be used over

and over!

**int** **getNumberOfSeats** (**int** whichConcert) {

**int** numSeats = 0;

**scanf** ("%d", &numSeats);

**return** numSeats;

} // getNumberOfSeats

* Correct approach

**int** **main**(**void**) {

**int** whichConcert = displayConcertMenu();

// Next find out how many seats customer wants

**int** numSeats = getNumberOfSeats(whichConcert);

}

// Prompts the user for the number of seats they want for the selected concert

// The user can select between 1 and 6 seats.

**int** **getNumberOfSeats** (**int** whichConcert) {

**int** numSeats = 0;

**printf** ("How many seats for the concert? ");

**scanf** ("%d", &numSeats);

// Use a while loop to do seat validation

**while** (numSeats < 1 || numSeats > 6) {

**printf** ("%d seats is not valid. Select between 1 and 6 seats. ", numSeats);

**scanf** ("%d", &numSeats);

}

**return** numSeats;

} // getNumberOfSeats

**Random Numbers**

* Should not use while loop until random number generator generates a value in a specific range.
* Instead should write code to force number from random number generator to be in desired range.

**if** (section == FLOOR) {

// Seats on the floor are letters A- F

// Generate a random number between 65 and 70 to represent sections A - F

sectionNumber = 65 + (**rand**() % (70-65));

}

**Today’s Terminology 5 min**

**Terminology**

* Pointer
  + Variable whose value is the address of another variable.
* Indirection
  + When you reference a value through a pointer
* Pass by Value
  + When a copy of an argument is sent to the function
* Pass by Reference
  + When the address of an argument is sent to a function
* Least Privilege Principle
  + Provide enough access to data but no more
* Function Pointer
  + Variable whose value is the address of a function!

**Passing Arguments to Function by Reference**

**Pass by Reference**

* C by default is pass by value – all values are passed by value
* But we can implement pass by reference with
  + Address operator (&) and
  + Indirection operator (\*)

**Example**

* Modification of example in book Fig 7.6 and Fig 7.7

**#include** <stdio.h>

**#include** <stdlib.h>

**int** **cubeByValue** (**int** number);

**void** **cubeByReference** (**int** \*numberPtr);

**int** **main**(void) {

**puts** ("Pass by value vs Pass by reference");

**int** number = 5;

**puts** ("Before cubeByValue");

**printf** (" number = %d\n\n", number);

**int** result = cubeByValue(number);

**puts** ("After cubeByValue");

**printf** (" number = %d\n", number);

**printf** (" Result = %d\n", result);

**puts** ("");

**puts** ("Before cubeByReference"); Showing that argument &number

**printf** (" number = %d\n", number); is being **passed by value**

**printf** (" &number = %p\n", &number);

cubeByReference(&number); // Sending the address of number

**puts** ("After cubeByReference");

**printf** (" number = %d\n", number);

**printf** (" &number = %p\n", &number);

} // main

**int** **cubeByValue** (**int** number) {

**int** cube = number \* number \* number;

// I can make changes to number but it will not affect the

// value that was passed in once control is returns to main

number = number + 5;

**return** cube;

} // cubeByValue

Note function is **receivin**g address This says pointer to integer

**void** **cubeByReference** (**int** \*numberPtr) {

**puts** ("In cubeByReference");

**printf** (" \*numberPtr = %d\n", \*numberPtr);

**printf** (" numberPtr = %p\n", numberPtr);

**int** cube = \*numberPtr \* \*numberPtr \* \*numberPtr; // All the \*'s makes this

// look confusing!

cube = (\*numberPtr) \* (\*numberPtr) \* (\*numberPtr); // This is a bit clearer

\*numberPtr = cube;

**printf** (" \*numberPtr = %d\n\n", \*numberPtr);

// I can make changes to numberPtr but it will not affect the

// value that was passed in once control is returns to main

numberPtr = numberPtr + 5;

**puts** (" Modifying numberPtr!");

**printf** (" numberPtr = %p\n\n", numberPtr);

} // cubeByReference

**Displays**

Pass by value vs Pass by reference

**Before cubeByValue call**

number = 5

**After cubeByValue**

number = 5

Result = 125

**Before cubeByReference**

number = 5 Want to show that **address to number**

&number = **0028FF10** is not changed – this is passed by value!

**In cubeByReference**

\*numberPtr = 5 In function we can see the value in **&number**

numberPtr = **0028FF10** was placed in **numberPtr**

\*numberPtr = 125

Modifying numberPtr!

numberPtr = **0028FF24** **numberPtr** is modified in function

**After cubeByReference**

number = 125 Modification to **numberPtr** in function did not

&number = **0028FF10** affect **number** outside function

**Visually**

|  |  |  |
| --- | --- | --- |
|  | Location | Value |
|  |  |  |
| number | **0028FF10** | 5 |
|  |  |  |
| Result | 0028FF14 |  |
|  |  |  |
|  |  |  |
|  |  |  |

…….. **\*numberPtr**

……..

……..

|  |  |  |
| --- | --- | --- |
| numberPtr | 0028FF1C | **0028FF10** |
|  |  |  |
|  |  |  |

**&numberPtr passed by value**

The value 0028FF10 can be

changed in function but those

changes don’t affect 0028FF10

in main.

But since 0028FF10 is a

memory address, what is at

that address (5) can be

modified! It’s not a copy, it’s

the real value -> 5!

* **Review Notes:**
  + Every variable has a memory address
  + How do I access a variables’ memory address?
    - Address operator address -> &variable
  + What can I do with a variable’s address?
    - Assign address to pointer variable -> int \*variablePtr = &variable;
  + What does not have a memory address?
    - Constants
    - Literals
    - Preprocessor defines
    - Expression (RHS)
  + Things you can do with a memory address (i.e. pointer)
  + What can I do with a pointer?
    - Access or modify the addressed variable
  + How do I access or modify an addressed variable?
    - Deference operator -> \*variablePtr
  + Use pointer parameter to receive an address

**void** **cubeByReference** (**int** \*numberPtr); // Function prototype

cubeByReference(&number); // Function invocation

**void** **cubeByReference** (**int** \*numberPtr) { // Function definition

// Code for function

}

cubeByReference receives the address of an integer value

* + The compiler does not differentiate between a function that receives a pointer and one that receives a one-dimensional array.

**void** **getSeats** (**int** seats[], **char** line[]);

***int seats[ ]*** is converted to ***int \*seats***

* + For two-dimensional array it’s more complicated –
    - Decay principle comes into play – complicated topic

***int seats[][]*** is not converted to ***int \*\*seats!***