CptS 121 - Program Design and Development

Final Exam Review Guide

This document will serve as a guide to help you prepare for the final written exam in CptS 121. You will find information about the exam format and topics you are expected to review within this guide.

What to Bring?

- Your WSU ID
- Two sharp pencils
- A "cheat sheet" (see below)
- Calculators and other notes may **not** be used during the exam!

The "Cheat Sheet"

The exam will be closed-book, but you will be allowed a “cheat sheet": **one side** of a page whose dimensions may not exceed 8-1/2" by 11" (i.e., standard sheet of notebook paper). This is twice the size of the “cheat sheets” allowed for the midterm exams. You must present your cheat sheet to your instructor at check-in, so that he can verify that it meets regulations. If you use a cheat sheet that exceeds the allowable dimensions, or that has writing on both sides of the page, you run the risk of its being confiscated prior to the exam. This policy will be strictly enforced.

Exam Timeframe

The final exam is scheduled for:
Friday, June 14, from 10:00 – 12:00 pm in Sloan 38

Note that, when you hand in your exam, you will be required to present your WSU ID and your cheat sheet to the exam proctor.

Exam Format

Expect the final exam to look a lot like the two midterms, except that it will be longer, because **you will have a full two hours** to take the exam, rather than one hour. There will be a mix of true-false, fill-in-the-blank, and short answer questions that test your knowledge of key concepts. Expect also to supply short code fragments, and to trace through C code fragments and specify their output.

Exam Coverage

The exam is comprehensive, covering all the material we have explored in this course. However, the emphasis will be on the final 4 weeks of the course (chapters 5 - 10, 12, 13, Appendix C, and pointers of the Hanly and Koffman text).

Topics that were covered in Midtern:

- See the midterm exam review

The following is a list of exam topics covered in the final 4 weeks of the course. These are the topics that will be emphasized on the final exam, so you would do well to devote the majority of your study to these topics:

Chapter 5: Iteration, Iteration, Iteration…

- Define what is *repetition* in programs
  - Allows for a group of operations to be performed multiple times
- Apply and implement the 3 looping constructs supported in C
  - for ( ), while ( ), do-while ( )
  - Recall these loop constructs may be transformed into one another
- Identify, apply, and implement the 5 major loop patterns
  - Counting or counter-controlled loop, used with for ( ) and while ( )
    - Indicates number of loop repetitions required are known before loop execution, i.e. while (count < 10)
  - Sentinel-controlled loop, used with for ( ) and while ( )
    - Indicates loop until a special value is encountered, i.e. while (array[i] != '\0')
  - Endfile-controlled loop, best suited to be used with for ( ) and while ( )
    - Indicates loop until the end of a file is reached, i.e. while (status != EOF) or while (!feof (infile))
  - Input validation loop, used with do-while ( )
    - Indicates loop until a value within valid range is entered by user, i.e. do {//something} while (input != valid)
  - General conditional loop, used with for ( ) and while ( )
    - Indicates processing of data until a condition is met
- Define what is one *iteration*
  - Discuss how loops are defined based on iterations
- Apply and implement nested loops
  - Walking through a 2-dimensional array requires the use of nested loops
- What is an *infinite* loop?
  - A loop which will execute "forever"
- Describe and apply compound assignment operators
  - These include: +=, -=, *=, /=, %=
- Describe and apply the increment and decrement operators
  - Post-increment (i++), pre-increment (++i), post-decrement (i--), and pre-decrement (--i)
- Provide an example of an *off-by-one* loop error

Chapter 6: Modular Programming and Pointers

- Define what is a *pointer*
  - A variable that contains the address of another variable
  - Used as *output* parameters which send back results from functions
- Distinguish between *output* and *input* parameters
- Declare and apply pointers
- Distinguish between the multiple usages of the * operator with pointers
  - int *ptr – indicates the declaration of a pointer
  - *result = i + j – indicates the dereferencing of a pointer (the operator is called the *dereference* or indirection operator)
- Define what is a *direct* value
- Define what is an *indirect* value

- Accessed via the dereference operator – meaning "follow the pointer"
- Apply logical memory diagrams of pointers and how they relate to their indirect values

Chapter 7: Simple Data Types, Enumerated Types, and Arrays

- Identify the integer types in C
  - short, unsigned short, int, unsigned, long, unsigned long
    - signed indicates negative and positive numbers are supported, this is the default type
- Identify the floating-point types in C
  - float, double, long double
- Discuss problems with applying floating-point numbers to loop conditions
- Declare and apply enumerated types in C
  - One example includes a Boolean type, where FALSE and TRUE may be assigned a variable of this type
- Describe what is an *array*
  - A collection of contiguous or adjacent memory cells associated with one variable name and one type
  - An array is considered a data structure
    - A *data structure* is a way of storing and organizing information; a composite of related data items
- Define what is a *subscript* and *index*
  - Recall array indexing in C starts at 0. Why?
- Declare and apply single and 2-dimentional arrays
  - Use an initializer list to initialize each item in an array
  - Use loops to traverse through arrays
- Write functions which accept arrays (single and 2-dimentional) as parameters
- What happens when an array is passed to a function?
  - The address of the 0th element, only, is copied and passed into the function
- How are arrays and pointers related?
  - Is array notation and pointer notation interchangeable?
    - Pointer notation may always be used with arrays; array notation may replace pointer notation only if the pointer points to the start of an array
- Declare and apply parallel arrays
  - Parallel arrays may be replaced by an array of structs

Chapter 8: Strings

- Define what is a *string* in C
  - A character array which contains alphabetic, numeric, and special characters, and is terminated by the null character ('\0')
- Declare and apply strings
- Declare and apply an array of strings
  - An array of strings is simply a 2-dimensional array of characters where each row represents a string and the max length of each string is determined by the max number of columns
- Apply string library functions <string.h>
  - strlen ( ) – returns the length of a string, not including the null character
  - strcpy ( ) – makes a fresh character-by-character copy of a string
  - strcat ( ) – appends one string to the end of another string
  - strcmp ( ) – performs a character-by-character comparison based on ASCII values
    - returns 0 if the strings are equal (case does matter), < 0 if string1 is less than string2, or > 0 if string1 > string2

- Write functions which mimic the four listed string library functions above, without calling the string library functions
  - Apply array and/or pointer notation to these functions
- Declare and apply arrays of pointers, i.e. char *array_ptrs[10]
- Distinguish between scanf ( ) and gets ( ) related to strings
- Apply the character operations found in <ctype.h>
  - These include: isalpha ( ), isdigit ( ), islower ( ), isupper ( ), toupper ( ), tolower ( ), ispunct ( ), isspace ( ), isalnum ( )

Chapter 10: Structs

- Define what is a *structure* in C
  - A collection of related fields or variables under one name
  - May be used to describe real world objects
- Define what is *encapsulation*
- Declare and apply structs
  - Declare, initialize, and print out struct information
- Declare and apply arrays of structs
- What is the . operator and what is the -> operator?
- Can the assignment (=) operator be applied to structs?
  - Yes, the assignment operator copies one struct to another

Other Topics

- Apply pointer arithmetic
  - For example, ptr++, ptr + index, --ptr, etc.

Chapter 9: Recursion

- Define what is *recursion*
- Define what is a *base case* and what is a *recursive step*
- Define what is a *recursive function*
- Define what is a function call *stack*
- Diagram the function and return *sequence* of a particular recursive solution
- Convert between iterative and recursive solutions and vice versa
- Apply and implement recursive functions to solve a problem

Appendix C: Bit Manipulation

- Define what is a *bit*
  - Recall a bit is a single binary digit – 0 or 1
- Define what is a *nibble* and *byte*
  - A nibble is a sequence of 4 bits and a byte is a sequence of 8 bits
- Convert between binary and decimal numbers and vice versa
- How can we determine if a number is even or odd without using the mod (%) operator?
  - If the least significant bit (lsb) is 1 the number is odd, if its 0 the number is even
- How do we multiply and divide by powers of 2 without using the multiplication (*) and division (/) operators?
  - Recall multiplication and division is expensive and resource intensive
  - Shift all bits in a number to the *left* by 1 to multiply by 2
  - Shift all bits in a number to the *right* by 1 to divide by 2
- Apply bitwise operations to decimal numbers

- These include: one's complement or negation (~), left shift (<<), right shift (>>), AND (&), OR (|), and XOR (^)

Chapter 12: Programming in the Large

- Define what is a command line argument
- Modify main ( ) to accept command line arguments
    - The arguments are *argc* and *argv*
- Extract strings from the command line arguments
- Convert strings to integers using atoi ( )

Chapter 13: Dynamic Data Structures

- Define what is dynamic memory
    - Memory allocated at runtime from a memory area called the *heap* or *memory store*
- Apply malloc ( ) to allocate memory
- Apply free ( ) to de-allocate memory
- Apply sizeof ( )
    - Recall this returns the number of bytes allocated for a type or variable

Recommended Strategy for Preparing for the Exam

I recommend that you use the following activities and materials to prepare for the exam:

Review quizzes and lab exercises: These may well be your best resource. An excellent learning activity would be to retake the quizzes and review the lab exercises.
Lecture slides and example code: Study the lecture slides and example code. Continue to complete extra coding examples on your own time.
Read the textbook: Read or re-read chapters 1 -10, 12, 13, & Appendix C in your textbook. Solve the end-of-chapter exercises.