CptS 121 - Program Design and Development

Exam 2 Review Guide

This document will serve as a guide to help you prepare for the second 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 5-1/2" (i.e., one-half of a standard sheet of notebook paper). 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

Please be aware that, because you will be taking the exam during a normal lecture period, **time will be extremely tight** for the exam. *If you show up late to class, you will have less time to take the exam*. 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 exam to look like an hour version of quizzes, with a few more involved problems that are more in the spirit of a lab exercise. Roughly 50% of the exam will be dedicated to concepts, with the other 50% being dedicated to C programming problems. For the concept section of the exam, expect true-false, fill-in-the-blank, multiple-choice, and/or short-answer questions similar to those found on the quizzes. For the programming problem section of the exam, I will present you with programming problems, and you will be expected to write syntactically-correct C code solutions that exercise good design. Note that your solutions do not have to be documented!

Exam Coverage

The exam covers the second five weeks of the semester, chapters 5, 6, 7, 8, 10.1 – 10.5 of the Hanly and Koffman textbook.

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 0^{th} 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.

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 5, 6, 7, 8, 10.1 – 10.5 in your textbook. Solve the end-of-chapter exercises.