

Test #3 Overview

Material covered

Chapters 7, 9, 10, and 13 in Programming in C++ by D’Orazio
Homework Assignments 5 - 8

Format

Open-books, open-notes, no computers

Test may contain T/F, multiple-choice, and short answer problems.

Test will deal with terminology, proper syntax in instructions, determining the results of C++ instructions or programs, and writing programs or instructions to accomplish specified tasks.

Chapter 7 – Functions

Library-defined functions versus user-defined functions

Functions may:

- return no value
- return one value (through the return statement)
- return two or more values (using reference parameters)

Three parts to using/defining functions:

- function declaration (or prototype)
- function call
- function definition

Dummy arguments

- variable names do not need to match between the function call and the function definition
- the number, order, and type of the variables must match

Function arguments – may be value parameters or reference parameters

- Value parameters (or input parameters or copy parameters) – input arguments
- Reference parameters – follow type by & - used for outputs mainly, but can be inputs also

Overloading functions

- two or more versions of the same functions can be defined with different numbers or arguments and/or different types
- C++ determines which version of the function to use

Default inputs – if default values are assigned to arguments in the prototype, the arguments can be omitted in the function call and the default values will be used.

Scope – local versus global variables. Do not use global variables to avoid passing arguments!

Chapter 9 – 1D Arrays

Arrays are also called subscripted variables or indexed variables

Declaring arrays

- use type, variable name, and brackets []
- value inside brackets must be an integer or const integer variable or expression with a defined value

Memory is allocated when the array is declared.

C++ does not check to see if you exceed array bounds, so you can crash the computer by writing beyond the bounds of an array.

Array indices begin at 0 (so `int A[8]` defines 8 variables: `A[0]` to `A[7]`)

Initializing arrays with lists

- List consist of a set of braces { } containing values separated by commas
- Recall that if less values are listed than are in the array then the remaining elements are initialized to zero. So `int A[100] = {0}` initializes all 100 values to zero.
- An uninitialized array contains junk, not zeros!
- If the array size is omitted, the array is sized to fit the list.

Printing arrays – the number of items printed per line is controlled by the loop

Reading values from arrays into data files.

Reading until the EOF marker is found.

Functions and arrays

- arrays are always treated as reference parameters, so no `&` required.
- typically dimension 1D arrays in the main program and pass the array and the array size to functions

Chapter 10 – Multi-dimensional Arrays

Arrays may have multiple dimensions: 1D, 2D, 3D, 4D, etc;

A 2D array is often called a matrix.

Loading arrays with lists.

- 2D arrays are loaded by row
- for 2D or higher, they are loaded by varying the indices, beginning with the rightmost index

Multidimensional arrays and functions

- Only the leftmost set of brackets can be empty in the function declaration and definition.
- The size of the leftmost set of brackets is typically passed as an argument.

Chapter 13 – C++ strings

Comparison to C-style character arrays primarily for reference.

Using **class string**, so be sure to use **#include <string>**

Typical class usage: dot notation, member functions.

Declaring and initializing strings

Concatenation using `+` and `+=` operators

String comparison using relational operators – based on ASCII values and lexicographical ordering

Accessing elements of a string using brackets `[]` – similar to using an array

Member functions in class string: **find**, **rfind**, **length**, **substr**, etc – refer to table in text or notes

Reading strings using `cin` (or `InData`, etc., with a data file) – reads one word at a time

getline – can be used to read one line at a time or to read until a certain character is encountered

- **getline(cin,S1)** or **getline(cin,S1,'\n')** – reads one line from keyboard into string S1
- **getline(cin,S1, '*')** - reads all characters up to and including * from keyboard into string S1

ignore (Example: **indata.ignore(50,'*')** – ignore up to 50 characters until * is encountered)

- useful for ignoring string input until a certain character is encountered
- Particularly helpful when using `getline` to read a string from a file after reading numbers (int, double, etc) as a `\n` character may be left in the file.

Strings in functions – strings can be used like any other variable as function inputs, returns, etc.

String arrays – arrays of strings are similar to arrays of other variables.