# CMPSC 122 Assignments, Exercises, Projects, Conventions, & Policies

All C++ programs are to be named with the .cpp extension. If applicable, you must also submit any student-provided .h files, and if any input or output files (.txt files only) are used, samples of those may also be submitted, unless required. Program design documents may be submitted with .doc, .docx, .gif, .jpg, and/or .pdf extensions. Students should regularly review the contents of this document.

*Students should not print out this file* as it may be changed weekly.

## Program Documentation

All C++ programs must begin with a comment that specifies:

1. the author,
2. the date,
3. a brief description of the purpose of the program (*usually a few sentences*)

Every function (except the main function; see above) must be prefaced by an explanation of its purpose (usually a sentence or two). In addition, this preface must include comments on the purpose and use of all passed parameters and returned values.

Every local variable and all constants, must be declared with a comment explaining its purpose and use. *Global variables are NOT allowed (with one exception, Assignment 7). Global constants are allowed; structures and unions can also be* ***declared*** *globally.*

Comments may be interspersed within functions in order to explain the logic flow.

## Naming Conventions

**All programs, and the variables, constants, and functions within programs, as well as any other files submitted, must adhere to the following naming conventions.** Failure to do so will result in points taken off the grade. (*Do NOT submit any files with spaces in their names!*)

### Program File Names

All program files must have the extension .cpp, and the name of the file must refer to the assignment, exercise, or project, i.e., the

1. program for in-class assignment week 1 must be named assignment1.cpp;
2. program for exercise 3 must be named exercise3.cpp; and,
3. project number 2 must be named project2.cpp. Note: project 1 is a paper.

### Text File Names

Text files may have any name, but must end with the .txt or .dat extension unless otherwise specified.

### Include File Names

*Student-provided include files, if any,* may have any appropriate name, but must end with the .h extension.

### Function, Variable, Constant, and Class Names

Function names must begin with a lower case alpha, and all other words in the name are capitalized, as in: getTimeOfDay ( … ).

Variable names must begin with a lower case alpha, and all other words are capitalized, as in: timeOfDay.

Constant names must be all in upper case, as in: RADIUS.

Class names (not applicable in this course, except in Assignment 14) must begin with an upper-case alpha, as in: Point.

## Compilation and Portability

Most students will be using the MS Visual Studio compiler on Windows to produce runnable programs; some students will be using Xcode on the MAC. The instructor will compile and run student programs using the GNU tools common on all UNIX and Linux systems.

*Portable* programs, programs that can be compiled on *any* operating system using any *standards-compliant C++ compiler*, are an important goal of this course. However, not all compilers are standards-compliant; in particular, MS Visual Studio is not standards-compliant, but it is the one that Penn State has chosen to use.

Visual Studio also allows a program that uses the <cmath> library (with functions such as *pow( )* and *sqrt( )*) to be used without specifically explicitly including the <cmath> library. **Portable programs MUST explicitly include the <cmath> library** with:

#include <cmath>

Generally, a submitted program that fails to compile will merit only 50% credit. However, most students will not have access to all 3 types of compilers, so this rule may be relaxed at the instructor’s discretion. The instructor will provide direction in class and in comments on students’ programs that bear on the portability issue. Students should PAY ATTENTION to those comments and make every effort to comply with them.

Please note that in CMPSC 101/121/201 efficiency and elegance of students’ programs were NOT requirements; in those courses it was most important that the program solved the problem given, following the directions given. In this course efficiency and elegance WILL also be counted in grading programs.

The use of the keyword **goto** and labels are absolutely not allowed.

## Use of Header Files

**Students must use only C++ header files, not C-language header files**, although the compiler permits the use of C-language headers. As an example, use the C++ <cstdlib> header, not the C stdlib.h. **Do not put any spaces in between the angle brackets**. The use of the Windows conio.h header is expressly forbidden.

## Use of Classes

This is a course on *procedural* programming, NOT *object-oriented* programming. We will be discussing how to use classes in C++ programs at the end of this course. We will also be discussing *assembler* *languages*, but students will not be asked to write any programs in assembler. You will have one exercise simulating *assembler language* programs.

Object-oriented programming using Java is taught in CMPSC 221.

This course will emphasize and REQUIRE the use of the basic mechanisms of programming as a foundational preparation for future programming courses.

We will make extensive use of the following C++ header files in this course:

* <iostream>
* <iomanip>
* <fstream>
* <string>
* <cmath>
* <cstdlib>
* <ctime>
* <cctype>
* <cstring>

**Hence, the use of ANY classes in programs other than those above is expressly forbidden** (this includes ANY *user-defined* *classes*). In particular, the following is a partial (i.e., not exhaustive) list of functions, classes, and adaptors NOT usable in this course:

* any function in header file <algorithm>
* array class in header file <array>
* format class in the header file <format>
* random class in header file <random>
* complex class in the header file <complex>
* sstream class in the header file <sstream>
* deque class in header file <deque>
* limit class in the header <limit>
* iterator class in header <iterator>
* list class in header file <list>
* map class in header file <map>
* queue class in header file <queue>
* set class in header file <set>
* stack class in header file <stack>
* any function in the header file <utility>

**You are not allowed to submit ANY programs that define C++ classes** or create C++ structures that have functions inside of their definitions.

## Submissions Posted to Canvas

Submission of programs to Canvas must include all program files (.cpp), all header files (.h, *if any*), examples of input and output files (.txt or .dat, *if any*), and all design documents (.docx, .gif, .jpeg, or .pdf, *if required*). Any files submitted with other extensions will be ignored for grading purposes. **Do NOT submit the MS Visual Studio project, or the Xcode project.** Do NOT submit any files with spaces in their names! Follow the file naming conventions specified above.

## Policy on Late Submission of Assignments, Exercises, and Projects

The purpose of this policy is to give students who manage their time well the opportunity to correct their programs IF AND ONLY IF the programs are submitted by the first submittal due date. This policy also allows students who do not manage their time well some latitude in submitting their programs for grading by the final due date.

IF a student turns in an in-class assignment, exercise, or project by *the first submittal date*, THEN that student may resubmit a corrected assignment, exercise, or project by the *final due date* (except as noted below). All final due dates are set on Tuesdays at midnight, unless explicitly changed by the instructor. Officially due dates are at midnight, but your instructor has added 10 minutes to the time to accommodate students whose clocks may be slightly off, so the actual due date is 10 minutes after midnight *on the next day*. Students should never attempt to bulk load their submissions at times close to the due date/time, because Canvas does not always work as expected.

If students submit multiple versions of a programming problem only the LAST submittal will be considered for grading purposes. Submittals MUST be posted to Canvas in the SAME drop box as the original submittal (failure to do this will result in the re-submittal NOT being considered for a grade change).

The due dates for assignments, exercises and projects are below.

Students who fail to submit an assignment, exercise, or project by the first submittal date will be granted an automatic extension to the final submittal date (see above, and except as noted below), but will not be allowed for grading purposes to resubmit at any later time.

**Caution**: as the programming assignments for this course are demanding, past semesters have shown that students who want an A in this course WILL make all programming submissions by the first submittal due date and WILL resubmit it by the final due date.

Exception: no student assignments or program will be accepted for grading purposes after the last day of class. If there is a final exam, the day and time of the final will be the last day of class. If there is no final exam, then the last day of class will expire at midnight of the Tuesday during final exam week.

## Policy on the Use of Input and Output Files; Use of Structures and Unions; Recursion

Whenever a programming problem requires that the input be read from a file, the program MUST prompt the user for the name of the input file.

Whenever a programming problem is CAPABLE of producing output more than 100 lines the output MUST go in an output file; you can name it any thing you want, BUT you must inform the user of the name of the output file just before you exit the program.

As structures and unions will be used in this class, the html files used in my CMPSC 121 class have been posted to Canvas under Modules for students who need a review of the material covered in my CMPSC 121 class or for students who did not take my CMPSC 121 class.

Input and output files in C++ are covered in the file called in\_out.html, and structures and unions are covered in the file called comp\_types.html.

Students may want to review these HTML files, and may also consult: <https://www.tutorialspoint.com/cprogramming/c_pointers.htm>,

<https://www.tutorialspoint.com/cprogramming/c_structures.htm>,

<https://www.tutorialspoint.com/cprogramming/c_unions.htm>, and

<https://www.tutorialspoint.com/cprogramming/c_recursion.htm>

## Due Dates sorted by 1st submission date (EC = extra credit)

| **Problem** | **1st Submittal (due date)** | **Final Submittal (closed)** |
| --- | --- | --- |
| Assignment 1 | 1/13/2023 | 1/17/2023 |
| Assignment 2 | 1/20/2023 | 1/24/2023 |
| Exercise 1 | 1/24/2023 | 2/7/ 2023 |
| Assignment 3 | 1/27/2023 | 1/31/2023 |
| Assignment 4 | 2/3/2023 | 2/7/2023 |
| Exercise 2 | 2/7/2023 | 2/21/2023 |
| Assignment 5 | 2/10/2023 | 2/14/2023 |
| Project 2 | 2/14/2023 | 2/28/2023 |
| Assignment 6 | 2/17/2023 | 2/21/2023 |
| Exercise 3 | 2/21/2023 | 3/14/2023 |
| Assignment 7 | 2/24/2023 | 2/28/2023 |
| Project 3 | 2/28/2023 | 3/21/2023 |
| Assignment 8 | 3/3/2023 | 3/14/2023 |
| Exercise 4 | 3/14/2023 | 3/28/2023 |
| Assignment 9 | 3/17/2023 | 3/21/2023 |
| Assignment 10 | 3/24/2023 | 3/28/2023 |
| Exercise 5 | 3/28/2023 | 4/11/2023 |
| Assignment 11 | 3/31/2023 | 4/4/2023 |
| Project 4 | 4/4/2023 ? | 4/18/2023 |
| Assignment 12 | 4/7/2023 | 4/11/2023 |
| Exercise 6 | 4/11/2023 | 4/25/2023 |
| Project 1 (Paper) | N/A | 4/25/2023 |
| Assignment 13 | 4/14/2023 | 4/18/2023 |
| Assignment 13 | 4/14/2023 | 4/18/2023 |
| Project 5 (Group Project) | 4/21/2023 | 5/2/2023, 6:00 PM |
| Assignment 14 (EC) | 4/21/2023 | 4/25/2023 |
| Project 1 (Paper) | N/A | 4/25/2023 |
| Exercise 7 (EC) | 4/25/2023 | 5/2/2023, 6:00 PM |
| SRTE Extra Credit | N/A | 5/2/2023, 6:00 PM |

## Due Dates sorted by final submission dates

|  |  |
| --- | --- |
| **Problem** | **Finally Due** |
| Assignment 1 | 1/17/2023 |
| Assignment 2 | 1/24/2023 |
| Assignment 3 | 1/31/2023 |
| Exercise 1 | 2/7/2023 |
| Assignment 4 | 2/7/2023 |
| Assignment 5 | 2/14/2023 |
| Exercise 2 | 2/21/2023 |
| Assignment 6 | 2/21/2023 |
| Project 2 | 2/28/2023 |
| Assignment 7 | 2/28/2023 |
| Exercise 3 | 3/14/2023 |
| Assignment 8 | 3/14/2023 |
| Project 3 | 3/21/2023 |
| Assignment 9 | 3/21/2023 |
| Exercise 4 | 3/28/2023 |
| Assignment 10 | 3/28/2023 |
| Assignment 11 | 4/4/2023 |
| Assignment 12 | 4/11/2023 |
| Exercise 5 | 4/11/2023 |
| Assignment 13 | 4/18/2023 |
| Project 4 | 4/18/2023 |
| Assignment 14 (EC) | **4/25/2023** |
| Project 1 (Paper) | **4/25/2023** |
| Exercise 6 | **4/25/2023** |
| Exercise 7 (EC) | **5/2/2023, 6:00 PM** |
| Project 5 (Group Project) | **5/2/2023, 6:00 PM** |
| SRTE Extra Credit | **5/2/2023, 6:00 PM** |

### Assignments

**Assignment 1**: write a program, assignment1.cpp, which reads integers into a **linked list** using **pointers** to structures *instead of an array of structures*. The integers *must be read from an input file* determined by prompting the user. A structure, similar to the following, must be used:

struct entry {

int value; // the integer read in

struct entry \*nextPtr; // pointer to the next structure

};

You must also declare a struct entry \*firstPtr which points to the first entry in the linked list. You must also declare a struct entry \*lastPtr which points to the last entry in the linked list. You must NOT allow the lastPtr to point beyond the end of the list. All structures in the linked list MUST be allocated by using the **new** keyword.

After reading in the integers in the file print out (using pointers) the integers read in the order they were read with no more than 10 entries per line.

**Assignment 2**: write a *recursive program*, assignment2.cpp, which determines the GCD (greatest common divisor) of a pair of positive integers. Prompt the user for 2 integers and verify that they are both positive integers before proceeding. GCDs are used to reduce rational numbers (fractions) to lowest terms.

Use Euclid’s algorithm to find the GCD. Here is an example of finding the GCD of 24,310 and 1235:

Step 1: 24310 = 1235 \* 19 + 845 (note that 845 = 24310 % 1235)

Step 2: 1235 = 845 \* 1 + 390

Step 3: 845 = 390 \* 2 + 65

Step 4: 390 = 65 \* 6 + 0

The GCD is the last NONZERO remainder, in this case 65.

**Assignment 3**: (ASCII table) in the text problem 13.17 (in the 9th edition; see Canvas for problem). Make sure the output is neatly formatted!

**Assignment 4**: (Combinations) in the text problem 14.12 (in the 9th edition; see Canvas for problem). Include the numbers 0 and 1; just embed them in the words as a 0 and a 1. Example: one possibility for 5701679 would be: LR01OPW. Assume the user correctly enters only 7 digits (no ‘-‘, parentheses or spaces). Read the user input into a char array (NULL-terminated). Use a recursive function to build the combinations. The recursive function will HAVE TO HAVE a loop in it! Leave the output in a file and inform the user of the file name before exiting.

You will need some constants:

const int ROWSIZE = 10;

const int COLSIZE = 5;

const int NUMSIZE = 8;

You will need an array to read in the user’s 7-digit number (‘\0’-terminated), a 2-dimensional character array, and an array in which to build the number combinations:

char array[NUMSIZE]; // user input – do not go beyond 7 chars and a NULL byte at the end!

// 2-d array of letters: 9 x 5

char letters[ROWSIZE][COLSIZE] = { { '0', '\0', '\0', '\0', '\0' }, // row index 0

{ '1', '\0', '\0', '\0', '\0' },// row index 1

{ 'A', 'B', 'C', '\0', '\0' }, // row index 2

{ 'D', 'E', 'F', '\0', '\0' }, // row index 3

{ 'G', 'H', 'I', '\0', '\0' }, // row index 4

{ 'J', 'K', 'L', '\0', '\0' }, // row index 5

{ 'M', 'N', 'O', '\0', '\0' }, // row index 6

{ 'P', 'Q', 'R', 'S', '\0' }, // row index 7

{ 'T', 'U', 'V', '\0', '\0' }, // row index 8

{ 'W', 'X', 'Y', 'Z', '\0' } }; // row index 9

char buildArr[ROWSIZE]; // output array; **must be null-terminated!**

Your recursive routine should have five parameters:

void combine (char \*, // user input array

char [ROWSIZE][COLSIZE], // 2-d array of letters, above

char \*, // array in which to build the output

int, // index into output array; **base case when this = 7**

ofstream&); // reference to the open output file

When the base case is reached output the 7-character array.

Inside the recursive combine function with the header:

void combine (char \*input, char letters[ROWSIZE][COLSIZE],

char \*build, int index, ofstream& out) { … }

to convert a character digit (‘0’ – ‘9’) to a row index in the 2-dimensional array use:

int indexNum = (int)array[index] - (int)'0';

**Assignment 5**: (Pig Latin) A language game, such as Pig Latin, alters spoken words to make them incomprehensible to the untrained ear. Write a program to prompt the user for an English *word, phrase, or sentence* and then translate the words into Pig Latin, as follows: the initial consonants are moved from the front of the word to the end and the letters “ay” are then added to the end. Thus “meal” becomes “eal-may”, “scram” becomes “am-scray”, and “java” becomes “ava-jay”. If the word begins with a vowel, then just add “ay” to the end; ‘over’ becomes ‘over-ay’.

Recommended: read the string into a char array using getline ( ), and use strtok (see the reference page for strtok in C++: https://www.cplusplus.com/reference/cstring/strtok/) to parse the string. The Visual Studio C++ compiler may issue an error saying that strtok is insecure. You can eliminate this “error” by putting the following line after the ‘using namespace std;’ line:

#define \_CRT\_SECURE\_NO\_WARNINGS

**However, if this does not work you will have to use strtok\_r** (and #include <string>). It requires 3 parameters; in addition to the 2 parameters required for strtok, the third and last parameter is a char \*\* (i.e., a pointer to a char \*). The third parameter is used by strtok\_r to hold its internal state (required for the secure, thread-safe, and reentrant version). To use a char \*\*:

char state = 0;

char \*ptr = &state;

Then the first call to strtok\_r passes the address of ptr as shown here:

char line[201]; // leaves plenty of room for a long sentence

char \*tokenPtr;

. . .

tokenPtr = strtok\_r (line, " ", &ptr); // \*\*ptr

and all subsequent calls to strtok\_r are:

tokenPtr = strtok\_r (NULL, " ", &ptr);

If you look at the reference page for strtok\_r you will see:

**char \*strtok\_r(char \*restrict** *str***,**

**const char \*restrict** *delim***,**

**char \*\*restrict** *saveptr***);**

The ‘restrict’ is NOT a new type. It is intended to be a “hint” to the compiler that for the char \* only the pointer itself or a value directly derived from it (such as pointer + 1) will be used to access the object to which it points. It is used for compiler optimizations.

I had hoped not to have to cover this until CMPSC 311.

**Assignment 6**: (Machine integer representation) integers in most machine architectures are represented in **little endian** format: the least significant byte is stored in the lowest address; for instance, the decimal number 23456789 is 0165EC15 in hex. In little endian format the number is stored in 4 bytes of main memory as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Byte (in hex) | 15 | EC | 65 | 01 |
| Byte Address | a | a+1 | a+2 | a+3 |

In **big endian** format the 4-byte integer is stored with the most significant byte is stored in the least address:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Byte (in hex) | 01 | 65 | EC | 15 |
| Byte Address | a | a+1 | a+2 | a+3 |

On Intel CPUs little endian format is used. However, many other CPUs do use big endian format. In addition, all 2- and 4-byte integers as integers (as opposed to strings or character arrays) sent over the Internet must be in network standard format, which is big endian.

Write a program that prompts the user to enter an unsigned integer in base 10 (the maximum value of an unsigned 4-byte int is 232 -1 = 4,294,967,295) and reverses its format (from little to big endian, or vice versa). Print out the user-entered number in hexadecimal and binary, reverse the endianness, and print the reverse in hexadecimal and binary.

**For this exercise you may NOT use the C++ hex formatting capability of the extraction operator (<<).** *Write your own routines to print a number in hexadecimal and binary using bit operators: <<, >>, &, |, and/or the unary operator: ~.* You can copy the conversion from base 10 to binary used in the lecture!

It is recommended that you use the following constants, a **union** to represent 4-byte integers, and print prototypes as shown here:

const int INTSIZE = 4; // in bytes

const int BYTESIZE = 8; // in bits

const int NIBSIZE = 4; // nibble, in bits

union integer4 {

unsigned int intrep;

unsigned char byterep[INTSIZE];

};

void prHex (unsigned char);

void prBin (unsigned char);

You will find that is very important that you use unsigned chars and ints. In most machine architectures, negative integers are represented in 2s complement format with the most significant bit a 1. To convert a binary (byte) 1, or 00000001, to negative -1 in binary, use the following procedure:

1. find the 1s complement; that is, flip all the 1s and 0s: 11111110;
2. add 1 to the result: 11111110 + 00000001 = 11111111, or 0xFF (the overflow bit is discarded).

Hence a 0xFF as an unsigned char is 255 (decimal), and 0xFF as a (signed) char is -1 (decimal).

Here is a sample run (be sure to use the spaces for readability):

Enter an unsigned integer in base 10 => 23456789

In hex:

15 EC 65 01

In binary:

00010101 11101100 01100101 00000001

Reverse endian:

In hex:

01 65 EC 15

In binary:

00000001 01100101 11101100 00010101

**Assignment 7**: write a *stack-based* application, assignment7.cpp, which can be used to recognize palindromes (strings spelled identically both backwards and forwards). Your program should *ignore spaces and punctuation, as well as alpha cases;* so, for instance, the string “Madam I’m Adam” counts as a palindrome.

You must supply your own push and pop operations. For this exercise (**alone**) you can use global variables for the stack and the stack pointer. Be sure that your program does not let the user push more than 25 characters!

*You are expected to push (and pop) the* ***entire*** *string onto the stack.*

**Assignment 8**: write a *queue-based linked list* application that allows the user to enter a list of positive integers (up to 25), possibly with duplicates in the list. Your application should enqueue the integers in the order specified by the user, *removing any duplicates*, and allow the user to enter 0 or a negative integer to quit entering integers.

Then, print out the queue resulting from head to tail.

**Assignment 9**: create a hash table for names (first name and last name only, no telephone numbers), and use the following to generate the keys:

unsigned char key = 0;

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

key ^= fullName[i]; // ^ = bitwise XOR

Your instructor will be using an input file similar to the following list of names to test your program:

Uzoma Acholonu

Giuliana Asmad

Michael Atkins-Combs

Vishnu Bakthisaran

Christopher Blowars

William Bronson

Trevor Butcher

Tiffany Caceres-Bonilla

Dulce Castro

David Cifuentes

Daniel Coursen

Alexandra Davilla

Amanda Dewitt

Alfredo Diaz

Jan Espinosa

Anthony Farrisi

Seth Fenstermaker

Daniella Gabout

Regina Green

Kelly Gregory

Bryan Houser

Michael Jackamorris

Kevin Kim

Benjamin Lee

Kyle Lighting

Perlamassiel Lopez

Omran Losinno

Lisa Maine

Derrike Mason

Andrew McCalla

Alexander McConnell

Niral Modi

Aida Montanez

Christopher Price

Daniyal Raza

Morgan Rex

Andres Rivero

Thomas Rudnicki

Kyle Russell

Samantha Rutkowski

Jaskamal Saini

Kyle Schmidt

Taylor Schnappauf

Ryan Snyder

Heazel Souid

Jason Spiegel

Minh Tang

Tehillah Trauger

Alexis Tocci

Luis Virola

Dale Wenger

Malik White

Jaimie Williams

When done loading the hash table print out the contents of each *occupied* entry, and any other entries pointed to by an occupied entry in the following format (use tabs, the columns do not have to line up):

. . .

Slot 51 is occupied with

Alexandra Davilla Ryan Snyder

Slot 56 is occupied with

Derrike Mason William Bronson Taylor Schnappauf Jason Spiegel

. . .

You do NOT have to print out the statistics (as shown in the HashTables.pptx).

**Assignment 10**:

This assignment is NOT a programming assignment.

Part A: for the following tree, what is the result of:

1. inorder traversal;
2. preorder traversal; and,
3. postorder traversal.

A picture containing text, indoor, map, dirty

Description automatically generated

Part B 1): what is the tree for the following mathematical expression?

((x \* y) + z) / ((a % (b – c))^2) (which is in infix notation).

Rewrite this expression in:

1. prefix notation; and,
2. postfix notation.

Put your answers in a .doc file and post it to Canvas.

**Assignment 11**: matrix operations. Write a program, assignment11.cpp, that reads two matrices and: 1) adds them (if compatible) and prints their sum, 2) subtracts them (if compatible) and prints their difference, and 3) multiplies them (if compatible) and prints their product. Prompt the user for the names of two matrix input files (see format below) and *place the output in a file of the user’s choosing*.

The format of the input files should contain the number of rows, followed by the number of columns, followed by the matrix data; here is an example of a 2 by 3 matrix input file:

2 3

1.0 -1.0 0.0

2.0 0.0 1.0

Here is an example of the output expected (note that both matrices are 3 x 3):

Matrix A =

1.500 -1.400 0.300

2.000 1.100 1.900

0.000 -1.700 1.800

Matrix B =

1.000 -2.000 1.000

0.000 1.000 0.000

1.000 -1.000 1.000

A + B =

2.500 -3.400 1.300

2.000 2.100 1.900

1.000 -2.700 2.800

A - B =

0.500 0.600 -0.700

2.000 0.100 1.900

-1.000 -0.700 0.800

A \* B =

1.800 -4.700 1.800

3.900 -4.800 3.900

1.800 -3.500 1.800

The next assignment requires the ability to multiply two matrices correctly, so it pays to get this one right!

**Assignment 12**: write a program, assignment12.cpp, which reads an adjacency matrix, M, (which will be square) from a file, and computes powers of M: Mp, where p varies from 1 to 5. Print out all powers from 1 to 5. Leave the output in a file; prompt the user for the name of the output file *or* inform the user of the name of the output file before exiting.

Note: Mp shows the number of paths of length p between the nodes of the graph.

The power of a matrix is recursively defined:

M1 = M

Mn+1 = Mn  M

For testing purposes, form the adjacency matrix for the following sample graph, and put it in a file (in the appropriate format – these matrices have int elements, not double elements), and run your program with that input file. The input adjacency matrix will contain all the paths of length 0 and 1 (between 2 possible nodes: A, B, C, D, and E). Consider the following sample graph:

Diagram, engineering drawing

Description automatically generated

Using this graph and its associated adjacency matrix in a file you should get the following partial output:

. . .

Matrix to the power = 3

0 2 1 3 0

3 1 0 0 1

0 1 1 2 0

2 0 0 0 1

1 1 0 1 1

. . .

**Assignment 13**: Determine whether the string 1011 is described by each of the following regular expressions:

1. 10\*1\*
2. 0\*(10 | 11)\*
3. 1(01)\*1\*
4. 1\*01(0 | 1)
5. (10)\*(11)\*
6. 1(00)\*(11)\*
7. (10)\*1011
8. (1 | 00)(01 | 0)1\*

This is NOT a programming problem; submit your answers in a .docx or .pdf file.

**Assignment 14 (EC):** Using the Point.cpp file developed class (and also up on Canvas under Modules, Block 8), add a Line class AFTER the Point class declaration. The Line class should have a constructor that is passed TWO Point objects and a method that couts out the equation of the line in one of THREE possible forms:

* y = mx + b, or slope-intercept form,
* y = b, b a constant when the slope is 0, or
* x = c, c a constant when the *slope is undefined*.

Recall that the slope between P1 = (x1, y1) and P2 = ( x2 ,y2) is *defined* as:

( y2 – y1) / ( x2 – x1 ), and is *undefined* when x1 = x2 .

You can copy and paste the Point.cpp file on Canvas, BUT remember that all .cpp files are text files only. They should only be opened using either NotePad (Windows) or TextEdit (MACs).

You will also need to *modify the main function* to test your new Line class; it must prompt the user for two Points.

**SRTE Extra Credit** (1 point): IF the SRTE participation rate is at least 75%, then you can do the following exercise for extra credit: write a 3-5 page paper (Times New Roman font, 12-point, double-spaced). References (bibliography) are required; footnotes (citations) are not required. Topic: compare and contrast the design criteria of object-oriented programming in C++ and Java as conceived by the *original developers* of these languages. C++ was developed by Bjarne Stroustrup in the 1980s, and Java was developed by a team at Sun Microsystems in the early 1990s.

### Exercises

**Exercise (Block) 1:** (Machine language simulation) in the text 9th edition, problems 8.15 & 8.16 (also scanned and posted to Canvas). This should be a single program; do not implement parts a), b), & c) in the first exercise until you have done the second exercise. That is, do the second exercise first and test it using the sample programs given in the text. Then proceed to write and test the programs specified in a), b), and c) in the first exercise.

This exercise is your introduction to machine-language programming and simulates the entry of programs into early computers (before assembly language was even invented).

You do not have to follow the instructions exactly as to input (see middle p. 373 in the 9th edition) or output (bottom of page 374 in the 9th edition). See the additional requirements in the Exercise1.pptx file up on Canvas.

For input you should read the instructions from a file (after prompting the user for the name of the file to read).

Here is a sample of the output expected when the simulator runs to completion on a correct machine language program:

. . .

Execution halted normally

REGISTERS:

accumulator -1

instructionCounter 6

instructionRegister 4300

opcode 43

operand 0

MEMORY:

0 1 2 3 4 5 6 7 8 9

0 1007 1008 2007 3008 2109 1109 4300 4 -5 -1

10 0 0 0 0 0 0 0 0 0 0

. . .

90 0 0 0 0 0 0 0 0 0 0

**Exercise (Block) 2**: (File matching and merge) problems 14.6-14.8 (in the 9th edition text; see Canvas for the problem). Only turn in your program for the 14.8 problem (the other problems are meant to prepare you for 14.8). Be sure to prompt the user for the names of the input files, and inform the user of the name of the output file before exiting.

**Exercise (Block) 3**: (Finite state machines) in this assignment you will be given a list of strings separated by commas (NO spaces allowed), and terminated by a period. You are to write a program that identifies valid C++ identifiers and non-negative integers. The C++ compiler has a similar functionality. When parsing programs for syntactical correctness, the compiler distinguishes between possible identifiers and possible non-negative integers by looking at the first character. If the first character is an alpha, then the remainder of the string MAY be a valid C++ identifier; if the first character is a digit, then the remainder of the string MAY be a valid non-negative integer.

A **finite state machine** (FSM) consists of a set of states, a set of transition rules, and an input string (all finite, of course). In the diagram below the ovals represent the states, and the arrows represent the transition rules. The strings in this case can only have the following symbols: A-Z, a-z, 0-9, underbar (\_), period (.), and comma (,) – NO spaces. This diagram depicts the FSM you are to write. Your program: should validate the string: 95,K9,abc,A20,.

And should reject the string: 95,K0,0abc,A20,.

It should also reject any string with a character NOT in the list above, such as: space, !, minus (-), +, ….

When a string is rejected, show where the fault lies (print the offending string to the string end), with output: This string is invalid beginning at: 0abc,A20,. Note that the invalidity is DETECTED when the 0 is followed by a.

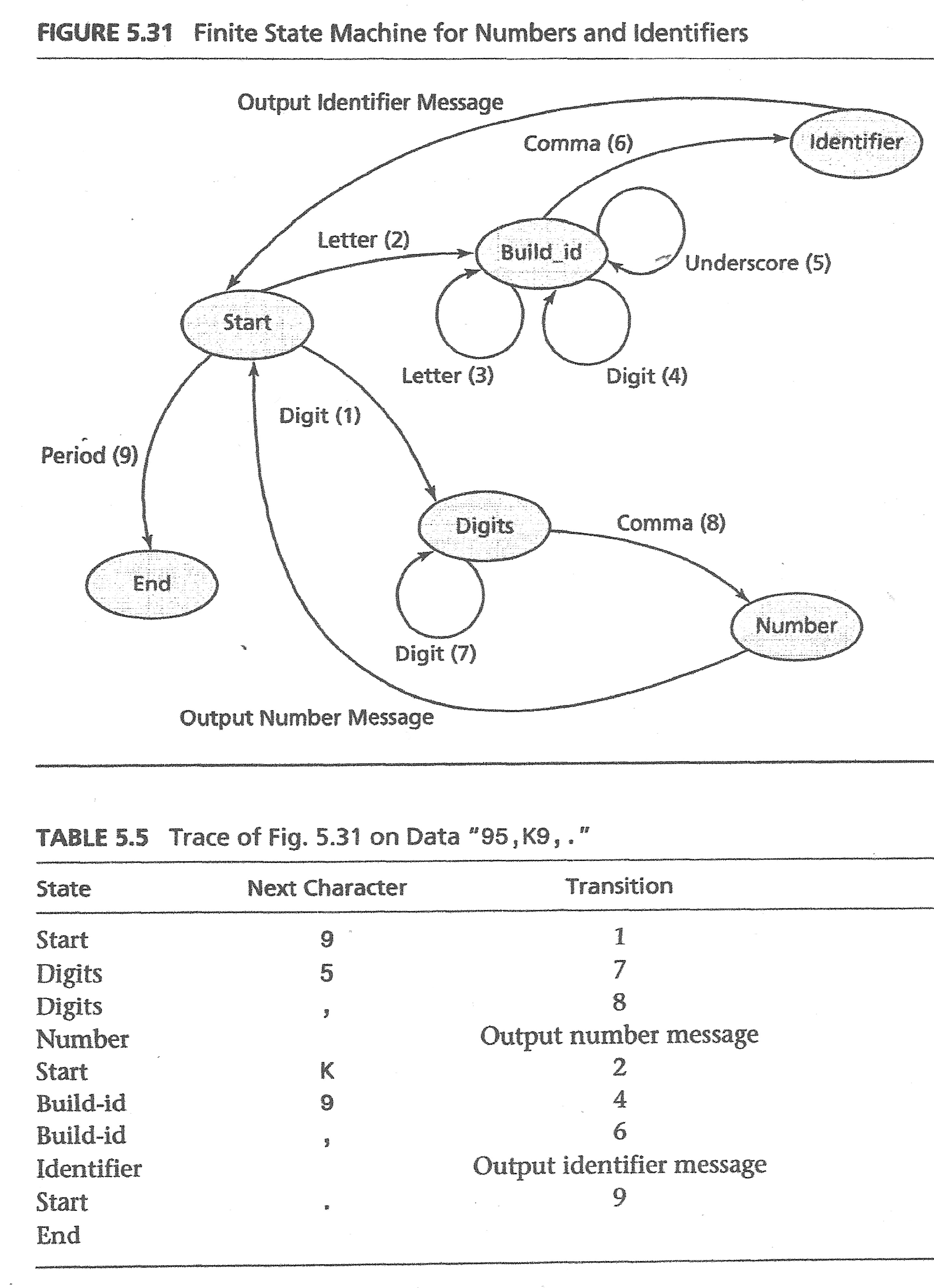
Write a program that *prompts the user for a string* (100 characters max) and either validates it or not according to this FSM. Every time it encounters a valid substring, your program should print it out along with its type (either an identifier or a positive integer). When it encounters an invalid string it should print out the rest of the string beginning with the string with the invalid character (and exit).

In the 95,K0,0abc,A20,. example, the output should be:

95 positive integer

K0 identifier

Invalid string beginning at: 0abc,A20,.



**Exercise (Block) 4**: implement a priority queue of maximum length 25. Use priorities ranging from 0 to 10 with the highest priority a 0. Prompt the user for a priority (first) and a job name (a string, second), and allow the user to enter a negative priority to quit entering jobs.

This exercise requires that you implement a function to insert a job anywhere in the queue, which *must be implemented as a linked list*. Do not remove any duplicates; that is, the same job can be entered multiple times with the same or different priorities. You may want to use the following structure:

struct pqEntry {

string job;

int priority;

struct pqEntry \*nextPtr;

};

After the user has quit entering priorities and jobs, print out the resulting queue from head to tail.

Sample output:

Enter priority => 1

Enter name of job => job1

Enter priority => 3

Enter name of job => job3

Enter priority => 0

Enter name of job => job0

Enter priority => 2

Enter name of job => job2

Enter priority => 2

Enter name of job => job4

Enter priority => 2

Enter name of job => job5

Enter priority => 2

Enter name of job => job2

Enter priority => -1

Priority: 0 Job: job0

Priority: 1 Job: job1

Priority: 2 Job: job2

Priority: 2 Job: job4

Priority: 2 Job: job5

Priority: 2 Job: job2

Priority: 3 Job: job3

**Exercise (Block) 5**: write a program that creates a tree for the following string data (stored in an input file – sample shown below), where:

1. the leftmost string goes into a node;
2. the middle string goes into the node’s left child; and,
3. the rightmost string goes into the node’s right child.

However, the string “null” in the middle position means the node has no left child, and the string “null” in the rightmost position means the node has no right child.

The first line contains the root node’s string in the leftmost position, and for every non-“null” string, there must be a line with that string in the leftmost position. Use the following data in the input file:

dog over brown

over red jumped

red The quick

The null null

quick null null

jumped fox null

fox null null

brown the lazy

the null null

lazy null null

This data produces the following tree:

Diagram

Description automatically generated

Note that the string “The” is not the same as the string “the”.

It is recommended that you **load** the tree using preorder searches while reading the input file. After the tree is loaded your program should print out the result of a postorder traversal with a space between each string and a newline at the end.

**Exercise (block) 6***:* write a program that generates random words from a training set as explained in the lectures on graphs. Do not hard-code the training set! Read it from a file with the following format for the input file is (using the example from the lectures):

6

a e m r s t

10

ate

eat

mate

meet

rate

seat

stream

tame

team

tear

Here are some suggestions for constants, array declarations, and helper functions (note that **use of global variables will be heavily penalized**):

#include <iostream>

#include <fstream>

#include <cstdlib>

#include <ctime>

#include <string>

using namespace std;

const int SIZE = 27; // 26 chars + 1

const int WORDSIZE = 25; // max word size

// read from input file

char cArray[SIZE]; // array of characters in set

char tArray[SIZE][SIZE]; // training array

//constructed by your program

int firstChars[SIZE]; // array of first character multiplicities

int transArray[SIZE][SIZE]; // transition array of multiplicities

char word[WORDSIZE]; // word being built

// helper functions

int getRandom(int); // get next random integer

int getIndex(char [SIZE], char); // get index of char in cArray

char getChar(char [SIZE], int); // get char at index in cArray

Be sure to seed the random number generator with the current time (ONCE) before using the srand() function:

srand(time(NULL)); // seed random number generator

*Prompt the user for the name of the input file and the number of iterations* (this program will run forever without this limit!). Either prompt the user for the name of the output file *or* inform the user of the name before exiting.

**Exercise 7 (Extra Credit - 2 possible points):** information gain; this extra credit exercise is explained in the lecture on Canvas under block 6. If you plan to do this exercise, you should meet with the instructor first!

### Projects

If a project deliverable involves writing source code, then in addition to the project source code you must also turn in a design. The design can be in pseudocode or UML. If in UML (see: <http://www.uml.org>) it can be handwritten and submitted as a .gif, .jpg, or .pdf image; if in pseudocode submit it in a .doc or .docx, and follow the basic directions and level of detail in: [http://www.unf.edu/~broggio/cop2221/2221pseu.htm](http://www.unf.edu/%7Ebroggio/cop2221/2221pseu.htm)**.**

For due dates see the table above.

**Project 1: Paper on History of Computing (due towards the end of the semester)**

Write a paper, project1.docx, covering the history of computer science spanning the early ages until the present; students will be assigned different time spans as shown below. Your report should cover hardware development, software development, systems development, important personalities that had an impact on the field of computing, and the influence of technology on various aspects of society at the time.

Start your research early.

Format: **5-10 pages**, Times New Roman font, 12-pt., double-spaced; references required. Citations in the APA or MLA format are NOT required. Include optional citations in the text using the ACM format.

Time Spans:

1. Early Development: before 1623
2. Mechanical Era: 1623 – 1936 (not including Alan Turing)
3. Alan Turing
4. John von Neumann
5. Early First Generation: 1936 – 1953 (not including John von Neumann)
6. Late First Generation: 1953 – 1959
7. Second Generation: 1959 – 1962
8. Early Third Generation: 1963 – 1969
9. Late Third Generation: 1969 – 1972
10. Fourth Generation: 1972 – 1984
11. Fifth Generation: 1984 – 1990
12. Early Sixth Generation: 1990 – 2004
13. Late Sixth Generation: 2004 – 2008
14. Early Seventh Generation: 2008 – 2012
15. Late Seventh Generation: 2012 – 2016

**Project 2: Permutations**

Write a **modularized C++** program that will **recursively** generate all the permutations of any given **n distinct objects** (in this case **use characters**).   For this project, use **n = 6** as the **maximum value** for **n**, but your program should have the ability to work for **any value of n** with minimal code modification. (Note: this program will require a recursive routine that has a loop in it.). For n = 6, the number of permutations of 6 characters is 6!, or 720, so an output file is required.

First prompt the user for *the number of characters to permute* (i.e., n); then prompt the user for the character data. At the least verify that the following are correctly permuted. *Inputs should have whitespaces between the characters.*

1. H I
2. 1 2 3
3. P S L V
4. @ # $ % &

Place the output in a file and inform the user the name of the file before exiting. Here is an example of the input and the output file:

Enter the number of characters you want to permute => 3

Enter 3 characters => 1 2 3

Sample output (in a file):

Permutation 1: 1 2 3

Permutation 2: 1 3 2

Permutation 3: 2 1 3

Permutation 4: 2 3 1

Permutation 5: 3 2 1

Permutation 6: 3 1 2

**Project 3: Modified Bisort**

Develop a highly modularized C++ program that will sort a set of numbers using a modified binary sort algorithm while maintaining a *doubly linked sorted list*.   Use pointers to build the double linked list.   Give the user the option to view every step of the sorting process (this **verbose option** will print each step to an output file and is a very useful debugging technique). Read the input from a file specified by the user. Leave the sorted output in a file (neatly formatted!), and inform the user the name of the file before exiting.

Here is a sample input file:

18 10 29 21 10 26 38 8 28 32 10 12 48 2 2 49 2 15 49 3 25 59 1 22

53 4 9 76 8 20 79 4 28 81 5 28 81 6 81 10 18 86 5 9 95 5 20

98 11 23 0 3 30 2 4 8 6 6 10 6 8 10 6 9 10 8 1 7 9 4 4

Do not remove any duplicates found.

A doubly linked list has pointers in both directions:

struct listMemb {

int value;

struct listMemb \*leftPtr;

struct listMemb \*rightPtr;

};

Consider a bisort of the following data: 44 55 12 42 94 18 67

Let h be a pointer to the first (head) element of the list.

Let t be a pointer to the last (tail) of the list.

Let m be a pointer to the approximate middle of the list.

Let x be the current input value.

Let c be a count of the elements in the list

Initially h => (points to) NULL, t => NULL, and m => NULL; c = 0 (a counter of the number of integers read)

*The following is a sample of what the verbose output should look* *like* (without the text in parentheses and the Note that’s)

Read x = 44

Element #: 1

Values: 44

c = 1

h => 44 (element 1)

t => 44 (element 1)

m => 44 (element 1)

Note that c/2 + 1 = 1 (integer division!)

Read x = 55

Element #: 1 2

Values: 44 55

c = 2

h => 44 (element 1)

t => 55 (element 2)

m => 44 (element 1)

Note that c/2 = 1

Read x = 12

Element #: 1 2 3

Values: 12 44 55

c = 3

h => 12 (element 1)

t => 55 (element 3)

m => 44 (element 2)

Read x = 42

Element #: 1 2 3 4

Values: 12 42 44 55

c = 4

h => 12 (element 1)

t => 55 (element 4)

m => 42 (element 2)

Read x = 94

Element #: 1 2 3 4 5

Values: 12 42 44 55 94

c = 5

h => 12 (element 1)

t => 94 (element 5)

m => 44 (element 3)

Read x = 18

Element #: 1 2 3 4 5 6

Values: 12 18 42 44 55 94

c = 6

h => 12 (element 1)

t => 94 (element 6)

m => 42 (element 3)

Read x = 67

Element #: 1 2 3 4 5 6 7

Values: 12 18 42 44 55 67 94

c = 7

h => 12 (element 1)

t => 94 (element 7)

m => 44 (element 4)

Algorithm (steps 2-6 in a loop):

Step 1: set h, t, and m to point to the first element read

Step 2: read next input: x; if x < h->value, then add x to the list prior to h, set h to point to the new element, and *adjust m*

Step 3: else if x > t->value, then add x to the end of the list, set t to point at the new element, and *adjust m*

Step 4: else if x < m->value, insert x somewhere between h and m (beginning at m), and *adjust m*

Step 5: else (if x > m->value), insert x somewhere between m and t (beginning at m), and *adjust m*

Step 6: go to step 2 and repeat the process until all data have been sorted

Note: if you encounter a duplicate x, add a new one.

The goal is to sort a list (with possible duplicates included in the input AND output) in ascending order. Here is some basic pseudocode:

read input

while more data to read

if input is <= the head pointer’s data, then

insert at head of queue

else if input is >= the tail pointer’s data then

insert at tail of queue

else

if input < the mid pointer’s data, then

insert somewhere in the first half of the queue

else

insert somewhere in the second half of the queue

end if

end if, else if, else

recompute m

read input

end while

To insert in the first half of the queue begin at the mid pointer and search left; to insert in the second half of the queue begin at the mid pointer and search right.

*To recompute the mid pointer* at the end of each read and insertion:

* keep track of the number of entries in the queue: n.
* if n is even, divide n by 2: n / 2 = m
* if n is odd, divide n by 2 (as integers) and add 1: c/2 + 1 = m
* **start at the head and find the mth entry in queue; put its address into the mid pointer.**

**Project 4: Tree Sort**

Develop a C++ program that will recursively alphabetize a set of strings in a user-specified file using the tree sort algorithm explained in the lectures while maintaining a balanced binary tree at all times.

The instructor will test your program with an input file containing:

Max Hank Jet Frisky Chata Richard Nan Sam Thomas Karen Gerri Ingrid Alan Dana

Because the string in C++ is a class, we can compare two strings with code like:

if (str1 > str2) ….

When done print out the contents of the tree with *inorder* traversal in a tabular format similar to the following.

This example used an input file with: Mark Susan Clarise Henry Ryan Jane Henry Tammy Fred Nan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NODE | LEFT | RIGHT | HEIGHT | BALANCE |
| Clarise | null | Fred | 2 | -1 |
| Fred | null | null | 1 | 0 |
| Henry | Clarice | Jane | 3 | 1 |
| Jane | null | null | 1 | 0 |
| Mark | Henry | Susan | 4 | 0 |
| Nan | null | null | 1 | 0 |
| Ryan | Nan | null | 2 | 1 |
| Susan | Ryan | Tammy | 3 | 1 |
| Tammy | null | null | 1 | 0 |

Extra Credit (1 point): *enhance* your program to allow for duplicates in the input data (but do NOT put duplicate entries in the tree). Expand the above table to contain a column for MULTIPLICITY, which keeps track of the number of duplicates encountered for each node in the tree. Turn in a SEPARATE program for this extra credit option, called project4\_ec.cpp.

Extra Credit (2 *more* points) *further* *enhance* your program, in a separate program called project4\_ec.cpp, as explained in the lectures so it will correctly alphabetize the following set of strings. If you do this extra credit option you do NOT need to do a separate program for the above 1-point extra credit option – this 2-point option should include the detection of multiplicities, for a possible 3 extra credit points.

Max Hank Jet Frisky Chata Juvy Winky Bridget Tiger Richard Thomas

Dixie Sewald Karinda Isabella Donato Hipolita SanJuan Kuko

Mariano Julio Rodriguez Salas Consuelo Irene Cruz Paredo

Marcos Paciencia Pitang Conrad Antonio Ding Josefa Nene

George Eamon Cappuccio McCluskey Carolina

**Project 5: Turing Machines (Group Project)**

This project is to be done as a group project (see the GroupsandPapers file on Canvas for group assignments). Design a program that simulates a Turing machine, project5.cpp; there should be no more than one executable for this project. **Each** of the exercises below requires that the program read a text file that contains the initial string for the Turing machine tape on the first line in the file, AND contains the state machine for that exercise in the remaining lines (i.e., 5-tuples with one 5-tupe per line). Hence the deliverables for this project are:

1. **the project design**;
2. **one project5.cpp file** that is to be used for ALL the exercises below; and,
3. **one input text file for each of 3 exercises below**.

Write a Turing machine to accomplish each of the following exercises. Use your project5 executable to simulate the Turing machine’s behavior. You must submit a sample text file for each exercise containing a sample input string for the tape, and a set of state machine transitions appropriate to that exercise.

As the Turing machine tape cannot be of infinite length, it is suggested that you limit the tape to 100 characters. The constants and the state transition structure could be:

const char BLANK = 'B';

const char ZERO = '0';

const char ONE = '1';

const char STAR = '\*';

const char LEFT = ‘L’;

const char RIGHT = ‘R’;

const int MAXTAPE = 100;

const int MAXSTATES = 25;

const short STARTPOS = 40;

const char STARTSTATE = ZERO;

struct stateTrans {

char state;

char readChar;

char nextState;

char writeChar;

char move;

};

Note that the initial string is in the MIDDLE of the tape (somewhere), and that the entire tape is first to be initialized to all blanks (all ‘B’).

Input and output should conform to the following. One example used in the lectures was a Turing machine that validated any string with all 1s, and rejected any string that had at least one 0. The example input file (with an input string of 5 1s) is:

11111

0101R

0B1BL

1111L

1B2BR

0030L

3141R

3B4BR

The output for this Turing machine simulation should print out the starting data as:

Tape starts at 40 length = 5 highlights position 40 tape =

B |1| 1 1 1 1 B

0 1 0 1 R

0 B 1 B L

1 1 1 1 L

1 B 2 B R

0 0 3 0 L

3 1 4 1 R

3 B 4 B R

Note that the initial string is shown flanked by Bs, and the starting *position (40) of the R/W head is highlighted inside two vertical bars*. The length of the initial string is also indicated.

At the end of the run the output should be:

Execution halts in state 2

Tape starts at 40 length = 5 highlights position 40 tape =

B |1| 1 1 1 1 B

**Note that the read/write head is left at the beginning of the validated string when successful.**

If we start instead with the input file:

11101

0101R

0B1BL

1111L

1B2BR

0030L

3141R

3B4BR

this changes the initial string to contain a 0, and the initial and final output should be:

Tape starts at 40 length = 5 highlights position 40 on tape =

B |1| 1 1 0 1 B

0 1 0 1 R

0 B 1 B L

1 1 1 1 L

1 B 2 B R

0 0 3 0 L

3 1 4 1 R

3 B 4 B R

. . .

Execution halts in state 4

Tape starts at 40 length = 5 highlights position 43 on tape =

B 1 1 1 |0| 1 B

**Note that the read/write head is left pointing to the invalid 0 embedded in the string.**

You MAY want to add a verbose option to help you with debugging (see Project 3, above).

**Project 5 Exercises**:

1. Create a Turing machine that validates all strings of alternating 0s and 1s; all of the following should be validated by the Turing machine:

|  |
| --- |
| Valid Strings |
| 0 |
| 1 |
| 01 |
| 10 |
| 010 |
| 101 |
| 0101 |
| 1010 |
| 01010 |
| 10101 |
| ... |

The string: 01001 should not be validated by the Turing machine. Note the regular expression to be validated (using symbolism explained in Week 13) is:

(01)\* | (10)\* | (01)\*0 | (10)\*1

1. Create a Turing machine to subtract two positive numbers in *unary 1s format*. Require that the second number be less than the first (so there are no negative results). In unary 1s format represent n as n+1 1s, m as m+1 1s, and the result n-m as (n-m)+1 1s. Example: 5 is to be represented as 111111, 2 is to be represented as 111, and 5-2, or 3, is to be represented as 1111. Be sure to verify that n – n is 0 (or a single 1 in unary 1s format). **Leave the R/W head over the left-most non-blank character**.
2. Create a Turing machine to find the 2s complement of an 8-bit **binary number** (*do NOT use unary 1s representation*). See also Assignment 6, above. For example: find the 2s complement of: 00010111, as an 8-bit binary number, which is done in 2 steps. First find the 1s complement (flipping all the 0s and 1s), which is: 11101000, and then add 1 to the result:

11101000

+ 1 =

11101001

Be careful to correctly handle any carry operations when adding 1 to the result!

Note that the sum of the original 8-bit number and its 2s complement (which is the negative of the original 8-bit number) is the 8-bit number: 00000000.

00010111 +

11101001 =

00000000

with a carry of 1 out of the most significant bit of the result. **Leave the R/W head over the left-most non-blank character**.

Be sure to verify that your Turing machine correctly determines that:

|  |  |
| --- | --- |
| 0000 0000 is the 2s complement of | 0000 0000 |
| 1111 1111 is the 2s complement of | 0000 0001 |
| 0000 0001 is the 2s complement of | 1111 1111 |