CQUPT – University at Albany
Computer Science – International College

ICSI 403 --- Design and Analysis of Algorithms
Project 1 --- Spring 2025

Assigned: Tuesday, March 4th, 2025.
Due Date: Your co-instructor will inform you the date this exercise is to be submitted.

Purpose of the Project

To refresh your C++ programming skills.

To Do

You are to write two C++ programs: (1) a Consumer, and (2) a Producer. Both programs will assume the use of ASCII codes with odd bit parity. The Consumer program will send encoded data representing a sequence of algebraic expressions to the Producer whenever a result for the computation is needed. The Producer will decode the received data, perform the requested computations, encode the result, and make the encoded result available to the Consumer. The Consumer will decode the result received from the Producer and display it to the user. All messages shared between both Consumer and Producer programs are to be encoded according to the format described as follows:

1. The Consumer

This program will

1) *read* the text file *project1Text.inpf* supplied by the user,
2) *encode* its contents and store it in file *project1Bin.binf.* It will also
3) *decode* the file *project1Result.outf,* received from the Producer. It will also
4) *store* the decoded information in the file *project1Result.done* to be available to the user. Table **project1Result.done** below illustrates the format to be used for reporting the processed results to the user for the input date provided by table **inputData1**.

All communication between Consumer and Producer will be done by means of a set of files as illustrated in the table **File Characteristics** below.

File Characteristics			
File Name	Contents	Created by	Accessed by
project1Text.inpf	any ASCII character	user	consumer
project1Bin.binf	(0 and 1) ASCII characters	consumer	producer
project1Result.outf	(0 and 1) ASCII characters	producer	consumer
project1Result.done	any ASCII character	consumer	user

All Consumer service requests will be available as text files containing algebraic expressions. The Consumer program will create a file of bits (‘0’ and ‘1’ characters) for each input file supplied. The data stored in each file will be structured as a set of data blocks. Each data block will consist of 2 SYN characters, ASCII 22, and a control character indicating the length of the message followed

by a maximum of 16 data characters. Every data block must contain 8 message characters except, possibly, the case where the remainder of the file cannot fill the 16 data character's block structure.

Each character will consist of 7 information bits with the addition of one parity bit which will be the most significant bit. A total of 8 bits ('0' or '1' character) will be used for the encoding of each input character. Consider, for instance, the binary encoding of character A which is 100 0001. The odd parity representation of this encoding is 1100 0001.

The table below illustrates the encoding of the "ABC ?" character string. The actual contents of this block must be encoded with characters '0' and '1' to simulate the binary representation of the characters used.

Character	Decimal	7-Bit Binary	8-Bit Odd Parity Binary
A	65	100 0001	1100 0001
B	66	100 0010	1100 0010
C	67	100 0011	0100 0011
space	32	010 0000	0010 0000
?	63	011 1111	1011 1111

The data block representation of the "ABC ?" character string is provided as follows where the two SYNC characters and the length of the data block are still encoded as decimal numbers. Blank spaces are used in the example below to help the understanding of the message encoding. As can be seen the actual contents of this block are encoded with characters '0' and '1' to simulate the binary representation of the characters used.

2222	5	1100 0001 1100 0010 0100 0011 0010 0000 1011 1111

The table that follows illustrates the complete encoding of the "ABC ?" string where the left most column is the two 22 SYNC characters, the middle column is the length of the data block, and the right most column is the message to be processed.

0001011000010110	10000101	11000001110000100100001100100000010111111

The actual message after the encoding will be a sequence of "0" and "1" characters with no formatting information as illustrated by the table below.

00010110000101101000010111000001110000100100001100100000010111111

Consider now the message "ABC ?A" and assume the maximum size of the data block to be 5. A second data block will be required in this case with the following contents:

0001011000010110	00000001	11000001

The information to be shared will be the concatenation of the two messages as illustrated below:

0001011000010110100001011100000111000010010000110010000001011111100010110000101100000000111000001

2. The Producer.

This program should take the files created by the Consumer as input, decode the stream of bits, perform the operations stated in the received file, and encode the result produced. Operations to be processed are either *initialization*, *arithmetic,* or *end of input* statements. The code that follows illustrates these statements:

inputData1
1) id_1 = 40
2) var_2 = 50
3) id_1 += 30
4) var_2 -= 20
5) id_1 *= 5
6) k_27 = 80
7) k_27 /= 20
8) </end/>

project1Result.done
id_1 = 350
k_27 = 4
var_2 = 30

Line numbers are not part of the statements; they have been included for reference only. Lines 1, 2 and 6 are examples of *initialization* statements. For example, line 1 defines *id_1* as an identifier and assigns it an initial value of *40*. The *</end/>* statement in line 8 is an *end of input* statement and it determines that no more statements are to be processed. All other statements are *arithmetic* statements. Line 3, for example, determines that the value of the identifier *id_1* is to be added by *30*. Your solution must be able to process any number of identifiers.

Your producer program must be composed of three states: *input*, *processing*, and *output* states. Behavior associated with each of these states are described as follows:

1. *Input:* your producer will decode the received file to obtain the original character contents of it and it will use a *Binary Search Tree* data structure to perform the operations associated with each expression. Your producer program then moves to the next state.
2. *Processing:* your producer will perform the computation as indicated by the operation associated with each instance of each identifier.
3. *Output*: your producer must sort the results obtained in alphabetic sequence such that the identifiers that are lexicographically smaller are placed first and encode the sorted sequency in the file *project1Result.outf.*

Your solution must use a *Binary Search Tree* and must also include a class *Symbols* with the following characteristics:

(a) a *string* to represent the identifier,
(b) an *integer* to represent the identifier's value,
(c) a method *performOperation(String* operation, *int* value *)* to carry out operations on the current value of the identifier,
(d) a *constructor* to create an object with a specified string and value, and
(e) a method *toString()* to give the string and its value.

Documentation.

Your program should be developed using the C++ programming language. It should be layered, modularized, and well commented on. The following is a tentative marking scheme and what is expected to be submitted for this assignment:

1. External Documentation (as many pages necessary to fulfill the requirements listed below.) including the following:
    a. Title page.
    b. A table of contents.
    c. [5% total] System documentation.
        i. [1%] A high-level data flow diagram for the system.
        ii. [2%] A list of routines and their brief descriptions.
        iii. [2%] Implementation details must provide evidence a Binary Search Tree is used for processing the arithmetic expressions.
    d. [14% total] Test documentation.
        i. [2%] How you tested your program.
        ii. [12%] Testing outputs. You may use the example presented in this document as well as the file below for testing your initial prototype. However, you must provide two additional test sets. In essence a total of four test sets are to be included.
        iii. *project1-test1-input.txt*

```
id_1 = 40
var_2 = 50
id_1 += 30
var_2 -= 20
id_1 *= 5
k_27 = 80
k_27 /= 20
f_32 = 40
l_20 = 20
z_11 = 21
p_52 = 52
ls_20 = 22
f_32 += 10
f_32 /= 5
</end/>
```

    e. [2%] User documentation.
        i. How to run your program.
        ii. Describe parameter (if any).

2. Source Code
    a. [75%] Correctness (Does your solution produce correct results?).
    b. [4%] Programming style.
        i. Layering.
        ii. Readability.
        iii. Comments.

iv. Efficiency.

How to Submit your Work.

- Your solution must be submitted according to the instructions provided by your co-instructor.
- Your files for *Project-1* must include the source code for all modules used for the exercise as well as an executable file to allow your solution to be tested. You must include all your source files in a Compressed (zipped) folder (.zip). Your (.zip) folder as well as the subject of your message must follow the format: *Project-1 Your Name*. Marks will be deducted if you do not follow this requirement.