Lab Assignment 02

Task 1: Write an algorithm for filing up a given tables of the size *P* x *Q* as per the description shown below :

A[0][0] to a[0][m-1] = 0

A[n-1][0] to a[n-1][m-1] = 0

A[0][0] to a[n-1][0] = 0

A[0][m-1] to a[n-1][m-1] = 0

A[1][1] to a[1][m-2] = 1 (when row exists and is not filled with 0)

A[n-2][1] to a[n-2][m-2] = 1 (when row exists and is not filled with 0)

A[1][1] to a[n-2][1] = 1 (when row exists and is not filled with 0)

A[1][m-2] to a[n-2][m-2] = 1 (when row exists and is not filled with 0)

A[2][2] to a[2][m-3] = 2 (when row exists and is not filled with 1)

A[n-3][2] to a[n-3][m-3] = 2 (when row exists and is not filled with 1)

A[2][2] to a[n-3][2] = 2 (when row exists and is not filled with 1)

A[2][m-3] to a[n-3][m-3] = 2 (when row exists and is not filled with 1)

.... and so on ....

Take the size of arrays and number of desired arrays from command line arguments .
Measure the running time complexity of your devised algorithm.

Sample Outputs:

```
4 x 3        5 x 7            6 x 7            7 x 7

0 0 0        0 0  0 0 0 0 0   0 0 0 0 0 0 0    0 0 0 0 0 0 0
0 1 0        0 1 1  1 1 1 0   0 1 1 1 1 1 0    0 1 1 1 1 1 0
0 1 0        0 1 2  2 2 1 0   0 1 2 2  2 1 0   0 1 2 2 2 1 0
0 0 0        0 1 1  1 1 1 0   0 1 2 2 2  1 0   0 1 2 3 2 1 0
             0 0 0  0 0 0 0   0 1 1 1 1  1 0   0 1 2 2 2 1 0
                              0 0 0 0  0 0 0   0 1 1 1 1 1 0
                                               0 0 0 0 0 0 0
```

Task 2: Implement a calculator ***(SmartCalci)*** which can store any *base-d* number (for any d up to a fixed maximum value, maxD) and perform few simple operations on them. Define an ADT(abstract data type) called Number that can store the digits at different positions of the number in the particular representation. The ADT must support ADD procedure and SUBTRACT procedure. For easiness of your implementation, assume that each digit is represented by a character. A separate lookup table needs to be created for storing the symbols (character) corresponding to the various digit values.
The program must take two command-line parameters: i) the first parameter is the name of a file containing the sequence of characters to be used for the digit values from 0 to maxD-1, and ii) the second parameter is the name of a file containing the instructions for the calculator.

The sequence of characters corresponding to digit values 0 to maxD-1 is given in the first line of the first input file, separated by spaces. Note that maxD to be used in the particular instance of execution is calculated by the number of entries in the first line. Only alphanumeric characters can be used for representing digits in the number system.

Consider a sample file:
File1: 0 1 2 3 4 5 6 7 8 9 a b c d e A B C D E
By reading this you should infer that maxD is 20 and the value of A is 15 and that of a is 10 and so on.
Every three lines of the second input file shall contain the operator, the first operand and the second operand respectively. For the lines containing the operands, the first field shall be the base of the number, d, followed by the digits of the number using the symbols defined in the first file. The fields in each line of the input file shall be separated by a single space.
Consider the following sample (second) file:
File2:

```
add
6 121
10 23
 subtract
 4 103
 3 12
 add
 12 b01a
 17 Aa0
subtract
 2 1011
 2 1010
```

Your program will have to read and interpret these as 4 operations to be performed in sequence. For instance, the first operation requires addition of $(121)_6$ and $(23)_{10}$

While adding or subtracting two numbers of different bases, the convention to follow shall be to convert the number with the smaller base to the higher base, and then perform the operation and provide the result in the higher base.

Create new type definitions: (Base.h) : Create a type definition for lookup table named *Base* , containing a list of symbols indexed by digit values. Note that since the base is immutable (i.e. won't change), the list can be allocated at time of definition.

Create new type definitions: (Number.h) :Create a type definition for a *Number* , containing the fields: the base of the number, a linked list of digits each of which is a character, and any other field that may be useful (e.g. number of digits).

Design Base ADT with the following operations (BaseOps.h)

a) int initializeBase(FILE *basefile) : reads the file containing the characters corresponding to each digit, populates the lookup table and returns the maximum base supported in the current execution. Note that to use the number system, this function has to be called first, before any other function.

b) ***int lookup(char c) :*** looks-up the character in the Base lookup table and returns the value associated with the character.

Design Number ADT with the following operations (***NumberOps.h***)

i) Number createNumber(char *number_format) : takes the number format as a string containing the base of the number, followed by a space, followed by the number using the custom characters(Most

Significant Digit first). This function then creates a Number using the values corresponding to the digits and returns it.

ii) Number add(Number a, Number b) : adds two numbers a and b and returns the result. If a and b are represented in the same base then the result should be in the same base. If not, the number in the smaller base should be converted to the larger of the two bases for addition and the result should be represented in the larger base.

iii) Number subtract(Number a, Number b) : subtracts b from a and returns the result. Follow the same convention as in addition vis-à-vis bases.

iv) void printNumber(Number n) : prints the Number in the base stored in the system.

Implement the above operations in the corresponding files: *NumberOps.c and BaseOps.c*

This will require implementation of the following helper function in ***NumberOps.c:***

Number convert(Number n, int to_base) : converts the number ***n*** to a number of base ***to_base*** and returns it. This is primarily a helper function used for adding and subtracting numbers of different bases. Note that since convert is a helper function this needs to be visible only inside NumberOps.c and hence should be declared static .

Main driver file. Create a simple driver file (***SmartCalci.c***) that can reads the two input files. The first input file containing the characters used in the number systems. The executable mycalc generates an output file which contains the result of (Operand1 *Operation Operand2*) for each triplet of lines in the input file.

For the above sample input files(File1 and File2), SmartCalci prints the following: (base of each number is indicated within parenthesis)

121(6) + 23(10) = 72(10)

b01a(12) + Aa0(17) = 4d77(17)

103(4) - 12(3) = 32(4)

1011(2) - 1010(2) = 1(2)

Points to Consider:

- Inputs to the program must be either (a) command line arguments (b) or read from a file (other than stdin). **Do Not Read** anything from stdin and **Do Not Use Prompts like** "Please Insert a number …".
- You are required to write the output to a file (other than stdout) and errors if any to a different output channel (stderr or another file).
- Use standard C coding conventions for multi-file programs. Major performance of each modules of your implementation using gprof utility.

How to use gprof tool to produce an execution profile of C

i) gcc –pg sampleProgram.c –o myProg //compile and generate myProg executable object

ii) ./myProg //Execute the program

iii) ***gmon.out*** // Check this new file generated automatically in working directory

iv) gprof myProg gmon.out > output.txt //profiling information present in output.txt