|  |
| --- |
| CS 6301 : Implementation of Advanced Data Structures and Algorithms |
| Project 4 (Type 1) |
| Arithmetic with Large Numbers |

#### ARJUN GOPAL - axg145630

Abstract: This project is an application of linked list data structures to represent very large numbers, the maximum number which can be stored in any data type depends on the bit size of the operating system and the processor , what if we want to represent a bigger number then this ?, this is what exactly this project does .

Problem statement: To demonstrate some elementary arithmetic operations on very large numbers, numbers which cannot be either represented by int or long data types making efficient usage of linked list data structure.

**IMPLEMENTATION DETAILS**

The project is implemented in Java and the details of the implementation are as follows.

Classes: Source code consist of 3 classes

1. LinkedNode : To represent how each node in a linked list looks like.
2. Operand: To maintain the structure of the operands.
3. Expression: To represent the structure of the expression evaluated.
4. The main class(AXG145630\_Project4) which includes the driver program and the linked list operations.

Methods employed:The code is written to handle a number in any base, and consists of the following methods

1. StrToNum: This function takes as input any arbitrary string representing a big number and converts it to number such that each linked list node has exactly one number.
2. NumToStr : This function takes as parameter a result list and converts it to a String for display
3. addTwoLinkedList : This function takes as parameter 2 lists performs addition on them and returns the result list
4. subtractTwoLinkedLists: Similar to the above function takes 2 lists as parameter but returns the difference between them, if difference is less than zero, zero is returned.
5. multiplyTwoLinkedLists: Takes 2 lists as input multiplies one by the other and returns the result,simple high school multiplication algorithm is employed at each step partial product is calculated and finally it is added to get the product.
6. PowerOfTwoLinkedLists: Takes 2 lists as parameters and calculates **List1 power list2** we have made use of simple repetitive multiplication algorithm to calculate the power.
7. createExpression: This function creates a expression which will be evaluated to get the result, all the operands will be stored in a map for future reference (to implement Go To functionality), it also stores the expression in a list. It takes as input a splitted arithmetic expression consisting of operands (large numbers) and operators (=, +,-, ^,\*) and returns an expression object.
8. evaluateExpression: Takes as input an expression object and evaluates it.

## Eligibility for extra credits:

The challenge for getting extra credit was to do the linked list operations in any base. The input and output will be always in base 10 but if another base (a long value which is limited by the number of bits that the system can take for one operation) is provided, then it should be used to save the number to the linked list. And all the operations in the linked list are carried out in base x. Conversion from base 10 to x and reverse is achieved by a simple logic which is explained below.

Converting from base 10 to base x

Let’s say the input number is a string and is in the base 10.

1. Initialize a linked list with one node and the data to be 0
2. Read the string from left to right and read integer n. For each n, initialize variable carry=n
3. Insert the element to the linked list starting from head
   1. Take the first node as current node.
   2. Multiply the data over there with 10
   3. Add carry to this. Divide this value with x and put the remainder in the current code. Update carry as the quotient and go to next node. Create new node if there is a carry and next node is null.
   4. Repeat step b till there is no carry and all the nodes in the list are traversed.

The Java code is shown below.

***private******static*** *LinkedNode StrToNum(String string) {*

*LinkedNode headNode =* ***new*** *LinkedNode(0);*

***for*** *(****int*** *i = 0; i < string.length(); i++) {*

***long*** *carry = string.charAt(i) - '0';*

*LinkedNode node = headNode;*

***while*** *(node !=* ***null****) {*

***long*** *tmp = node.data \* 10 + carry;*

*node.data = tmp % base;*

*carry = tmp / base;*

***if*** *(node.next ==* ***null*** *&& carry > 0) {*

*node.next =* ***new*** *LinkedNode(0);*

*}*

*node = node.next;*

*}*

*}*

***return*** *headNode;*

*}*

Converting from base x to base 10

We have to do the same procedure explained above, in a reverse order. Read the linked list from right to left and get each element. Convert it to base 10 and store it in a new linked list. Algorithm is same as explained below.

### Platform:

The problem was coded on a 64 bit windows 8 machine. Dual Core I7 Processor.

### I/O Format:

The program can be compiled and executed as follows.

>javac AXG145630\_Project4.java

>java AXG145630\_Project4 <base>

<base> is an optional argument which will be a long value(limited by system configuration). If <base> is provided, then numbers will be stored in linked list with that base. Otherwise the default value will be 10.

The main function is a driver method which reads expressions in a loop from the standard input till the string “1001” is read. Input is given in the following format, <line number > < Expression>. Expression can take one of the following forms [<expression = <operand>],[<expression> = <operand> <operator><operand>],[<expression> = <expression>], [<expression>]. Output will be printed only after the loop is terminated by giving the string “1001”.

### Test Results:

These are results of few sample test cases executed

#### Input

1 x=999

2 y=8

3 z=x+y

4 z

5 a=x^y

6 a

#### Output

1007

992027944069944027992001

#### Input

1 x=10

2 p=1

3 n=1

4 p=p\*x

5 x=x-n

6 x? 4

7 p

#### Output

3628800

## Conclusion

The linked list representation makes it possible to do arithmetic on big numbers. It gives an extra advantage if the number can be represented in a different base, say a larger one. This will optimize the computation speed also as it needs to consider only less number of elements if the base is larger. The project was initially implemented to support only base 10 and later the feature was added and considerable improvement in the performance was noted.

The Project was successfully implemented with an equal contribution from all the group members and tested properly. It took 4 days for the implementation and 1 day for all the testing. Report was built along with the implementation.