EMERTXE TRAINING PROJECT DOCUMENTATION FRAMEWORK

REQUIREMENTS & DESIGN DOCUMENT

Module – Data Structures

Arbitrary Precision Calculator

EMERTXE

Contents

1 Abstract

Arbitrary-precision arithmetic, also called bignum arithmetic, multiple precision arithmetic, or sometimes infinite-precision arithmetic, indicates that calculations are performed on numbers whose digits of precision are limited only by the available memory of the host system. This contrasts with the faster fixed-precision arithmetic found in most arithmetic logic unit (ALU) hardware, which typically offers between 8 and 64 bits of precision.

Applications of APC

- A common application is public-key cryptography, whose algorithms commonly employ arithmetic with integers having hundreds of digits.
- Arbitrary precision arithmetic is also used to compute fundamental mathematical constants such as π to millions or more digits.

2 Requirements

Operations to be implemented

- Addition (+)
  - Subtraction (-)
  - Multiplication (*)
  - Division (/)
  - Modulus (%)
  - Power (^)

NOTE :

- All operations should work for integer numbers and also for numbers with decimal point.
  - Slice the numbers according to sizeof(int) (Should be portable). Maintain Double Linked List

Points to be taken care

Addition

1. If any of the numbers are zero, your algorithm should be smart enough to reduce the work.
   1. If Num1 = 0 and Num2 = x

      Then directly print Num2 as output.
   2. If Num1 = x and Num2 = 0

      Then directly print Num1 as output.
   3. If Num1 = Num2 = 0

      Then directly print 0 as output.
2. If operation in any nodes results a carry. Don't forget to propagate this carry to next node

Incorrect	Correct
9900 1100 1234 9999 + 0012 0100 0023 9999	1 9900 1100 1234 9999 + 0012 0100 0023 9999
Res: 9912 1200 1257 9998	Res: 9912 1200 1258 9998

3. Output to be taken care if any node has zero value or less number of digits.

Incorrect	Correct
9900 0012 0000 9999 + 0012 0000 0000 0000	9900 0012 0000 9999 + 0012 0000 0000 0000
Res: 9912 12 0 9999	Res: 9912 0012 0000 9999

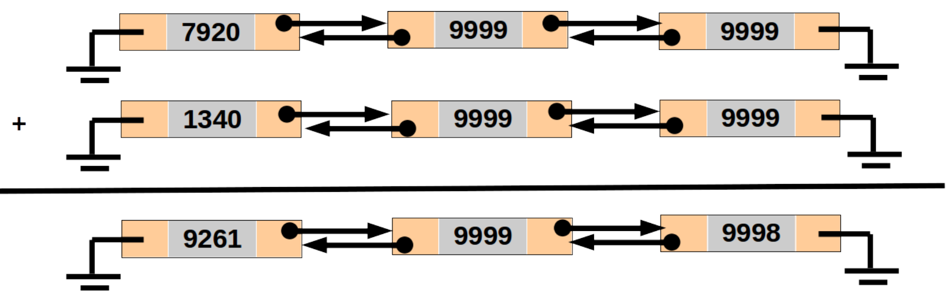
4. If number of nodes for numbers are different after slicing. Should not stop addition when nodes of one smaller number gets over if last addition results a carry.

Incorrect	Correct
1 1121 9900 9999 0000 9999 + 9999 0001	1 1 1 1121 9900 9999 0000 9999 + 9999 0001
Res: 1121 9900 9999 0000 0000	Res: 1121 9901 9999 0000 0000

NOTE :

There are many such situations in other operations where it is possible to make algorithm optimal. Think about such scenarios !!!!

Slicing the numbers



Example Operations

```
9839018013073709710749047104719074109 + 074101740107404748484848400000
9839018087175449818153795589567474109
```

Fig 1: Addition

```
258085582922292020220222737 - 464648499994940000000000004848494048393939
-46464849999493741914417082556473828171202
```

Fig 2: Subtraction

```
1844800000819911010101001018227373 * 1098401841133133131313335669999900
2026331717422995763707108078552662837274783168010809031193087262700
```

Fig 3: Multiplication

```
723489682365828683261324790262851045214441399101 / 801038018301830183030138381811
903190193019301391
```

Fig 4: Division

```
933738391391839183913193819389 % 9810308310831083
5314376719615855
```

Fig 5: Modulo

```
123456 ^ 123
18030210630404480750814092786593857280734268863855968048844015985795
85023608137325021978269698632257308716304364197947589320743503803676
97649814626542926602664707275874269201777743912313197516323690221274
71384589545774873530948433719137325552792827178520638296799898433048
21053509422299706770549408382109369523039394016567561276077785996672
43702814072746219431942293005416411635076021296045493305133645615566
59073596565258793429042547382771993501287009357598778943181804701340
46917957731704057646146460549492988461846782968136255953333116113852
51735244505448443050050547161779229749134489643622579100908331839817
426366854332416
```

Fig 6: Modulo

3 Prerequisites

- Pointers, Structures and Dynamic Memory Handling
- Double Linked List

4 Design

Required Structure

```
typdeef int data_t;
typedef struct node
{
        struct node *prev;
        data_t value;
        struct node *next;
} Dlist;
```

Function Prototypes

Operation	Addition
Prototype	int addition(Dlist **head1, Dlist **tail1, Dlist **head2, Dlist **tail2, Dlist **headR);
Input Parameters	• head1: Pointer to the first node of the first double linked list. • tail1: Pointer to the last node of the first double linked list. • head2: Pointer to the first node of the second double linked list. • tail2: Pointer to the last node of the second double linked list. • headR: Pointer to the first node of the resultant double linked list.
Return Value	Status (SUCCESS / FAILURE)

Operation	Subtraction
Prototype	int subtraction(Dlist **head1, Dlist **tail1, Dlist **head2, Dlist **tail2, Dlist **headR);
Input Parameters	• head1: Pointer to the first node of the first double linked list. • tail1: Pointer to the last node of the first double linked list. • head2: Pointer to the first node of the second double linked list. • tail2: Pointer to the last node of the second double linked list. • headR: Pointer to the first node of the resultant double linked list.
Return Value	Status (SUCCESS / FAILURE)

Operation	Multiplication
Prototype	int multiplication(Dlist **head1, Dlist **tail1, Dlist **head2, Dlist **tail2, Dlist **headR);
Input Parameters	• head1: Pointer to the first node of the first double linked list. • tail1: Pointer to the last node of the first double linked list. • head2: Pointer to the first node of the second double linked list. • tail2: Pointer to the last node of the second double linked list. • headR: Pointer to the first node of the resultant double linked list.
Return Value	Status (SUCCESS / FAILURE)

Operation	Division
Prototype	int division(Dlist **head1, Dlist **tail1, Dlist **head2, Dlist **tail2, Dlist **headR);
Input Parameters	• head1: Pointer to the first node of the first double linked list. • tail1: Pointer to the last node of the first double linked list. • head2: Pointer to the first node of the second double linked list. • tail2: Pointer to the last node of the second double linked list. • headR: Pointer to the first node of the resultant double linked list.
Return Value	Status (SUCCESS / FAILURE)

5 Sample Output

```
user@emertxe] ./apc 76234567891123234455522 + 119835485936279949364
76246551439716862440486
user@emertxe] ./apc 76234567891123234455522 - 119835485936279949364
762225843425296064506158
user@emertxe] ./apc 76234567891123234455522 * 119835485936279949364
91356064883750773594387361915825144310188008
user@emertxe] ./apc 76234567891123234455522 / 119835485936279949364
6361.60209936974722456957
user@emertxe]
```

Fig 5 1: Expected Output

6 Artifacts

6.1 Skeleton Code

- www.emertxe.com/content/data-structures/code/arbitraryprecisioncalculator_src.zip

6.2 References

- https://en.wikipedia.org/wiki/Arbitrary-precision_arithmetic
- http://rosettacode.org/wiki/Arbitrary-precision_integers_(included)
- http://www.sciencedirect.com/science/article/pii/S1567832604000748
- http://bt.pa.msu.edu/pub/papers/HICOSYMSU08/HICOSYMSU08.pdf