INTAL - INTegers of Arbitrary Length

Project Description

The maximum limit of Unsigned Long Int in C/C++ is 18446744073709551615, a 20 digit number. While languages like C++/Java support classes of BigIntegers (100 digit numbers).
C by default has no such support. This project aims to bring that support to the C language along with basic arithmetic operations like Comparison, Addition, Subtraction, and Multiplication along with applications such as Factorial, Fibonacci, etc.

Author

Soumabha Banerjee

Project Language(s)

C Programming Language

Difficulty

Intermediate

Duration

12 hours

Prerequisite(s)

Arrays, Basic Mathematics

Skills to be learned

C, Low-Level implementation of arithmetic operations, BigIntegers

Overview

Objective

Add support for Big Integers (numbers greater than 18446744073709551615) in C along with basic mathematical operations and functions.

Project Context

Languages like C++/Java support classes like Bigintigers allowing developers to utilize numbers with up to 100 digits. **INTAL-BigIntiger** aims to provide support up to 1000 digits in the C language along with basic mathematical operations (comparison, addition, subtraction, multiplication) and some mathematical functions (Fibonacci series, factorial).

Project Context

Languages like C++/Java support classes like BigIntegers allowing developers to utilize numbers with up to 100 digits. INTAL aims to provide support up to 1000 digits in the C language along with basic mathematical operations (comparison, addition, subtraction, multiplication) and some mathematical functions (Fibonacci series, factorial).

Project Stages

The project consists of the following stages:

Start >>>>>> Write all Intal Functions >>>>>> Write main function for implimentation >>>>>> Stop.

High-Level Approach

• Write a function to compare INTALs; returns equal, lesser than or greater than comparisons of inputted numbers.
• Implement basic arithmetic operations on INTAL: Addition, Subtraction, Multiplication
• Implement applications of INTAL: Factorial

Applications

• Calculate factorial of very large integers
• Calculate the very large nth Fibonacci number
• Calculate the binomial coefficient of very large numbers
• Perform binary exponentiation on very large numbers or raise numbers to very large powers
• Greatest Common Divisor

Task 2

INTAL Comparator

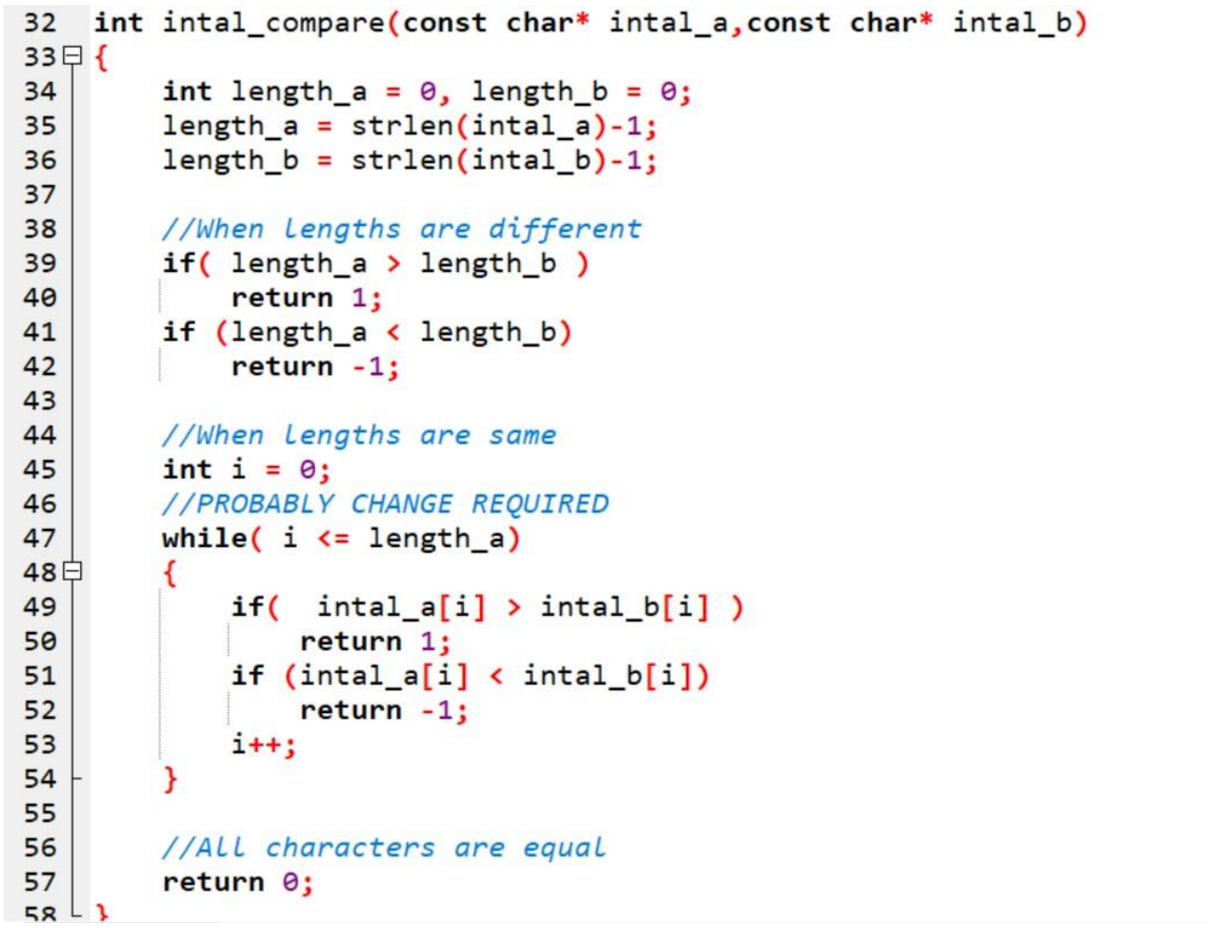
Write a function that compares two INTALs.

Requirements

• Write a function that accepts two INTALs as function parameters and returns `0` or `-1` or
`1` if both INTALs are equal or the first INTAL is lesser than the second INTAL or second INTAL is lesser than the first INTAL respectively.
• The function must run in O(n) time (complexity), where n is the number of digits in INTAL or the number of characters in the array.
• INTALs passed to the comparator may be of unequal length and should be kept in mind
while determining which INTAL is greater.

Skills

• the for loop
• Comparator in C



```c
int intal_compare(const char* intal_a,const char* intal_b)
{
    int length_a = 0, length_b = 0;
    length_a = strlen(intal_a)-1;
    length_b = strlen(intal_b)-1;

    //When Lengths are different
    if( length_a > length_b )
        return 1;
    if (length_a < length_b)
        return -1;

    //When Lengths are same
    int i = 0;
    //PROBABLY CHANGE REQUIRED
    while( i <= length_a)
    {
        if(  intal_a[i] > intal_b[i] )
            return 1;
        if (intal_a[i] < intal_b[i])
            return -1;
        i++;
    }

    //All characters are equal
    return 0;
}
```

Expected Outcome

Your function should be able to correctly identify the larger INTAL

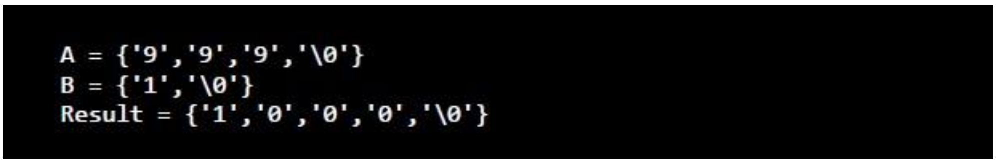
Task 3

INTAL Adder

Write a function that adds two INTALs and returns their sum.

Requirements

• Write a function that accepts two INTALs as function parameters and returns the sum of both INTALs.
• The function must run in O(n) time, where n is the number of digits in INTAL or the number of characters in the array.
• INTALs passed to the adder may be of unequal length and should be kept in mind while
determining the sum of INTALs.
• Use the same concept of elementary mathematical addition for adding the INTALs, i.e., basically in case the ith element exceeds 9, a carryover needs to be made to the next element.
• Example:



```
A = {'9','9','9','\0'}
B = {'1','\0'}
Result = {'1','0','0','0','\0'}
```

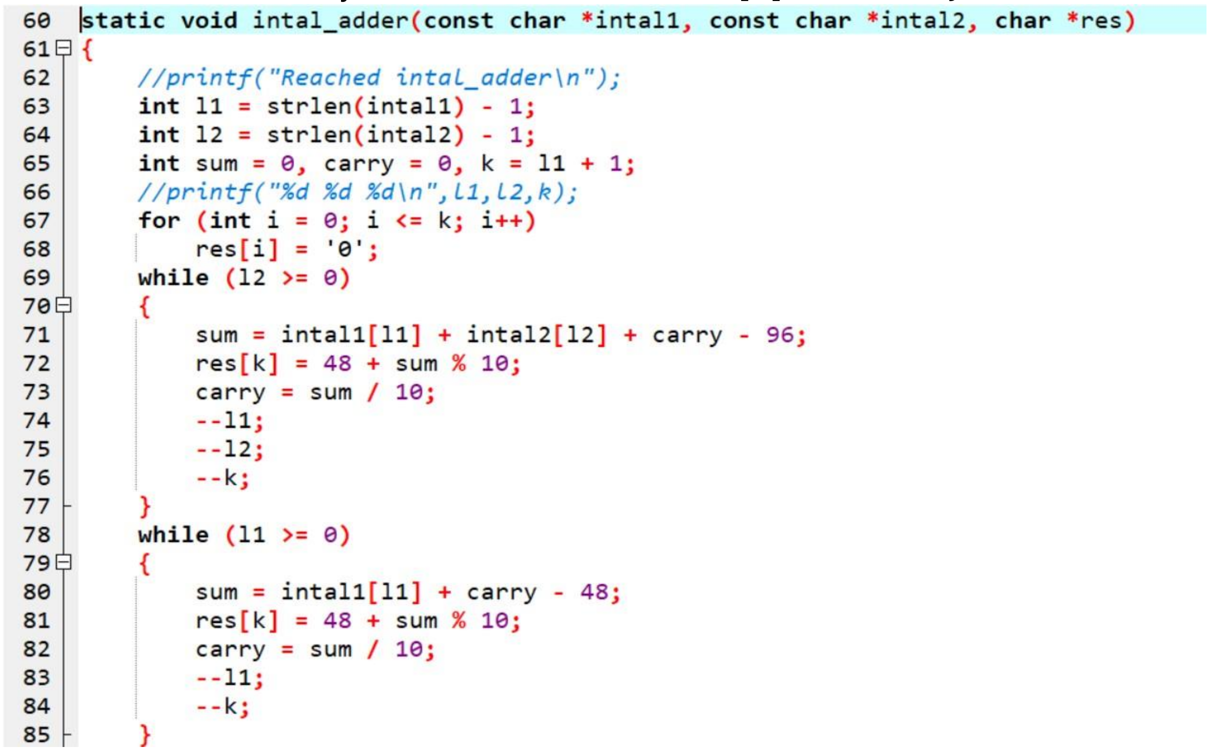
• Try visualizing the Additions link provided in references.

Skills

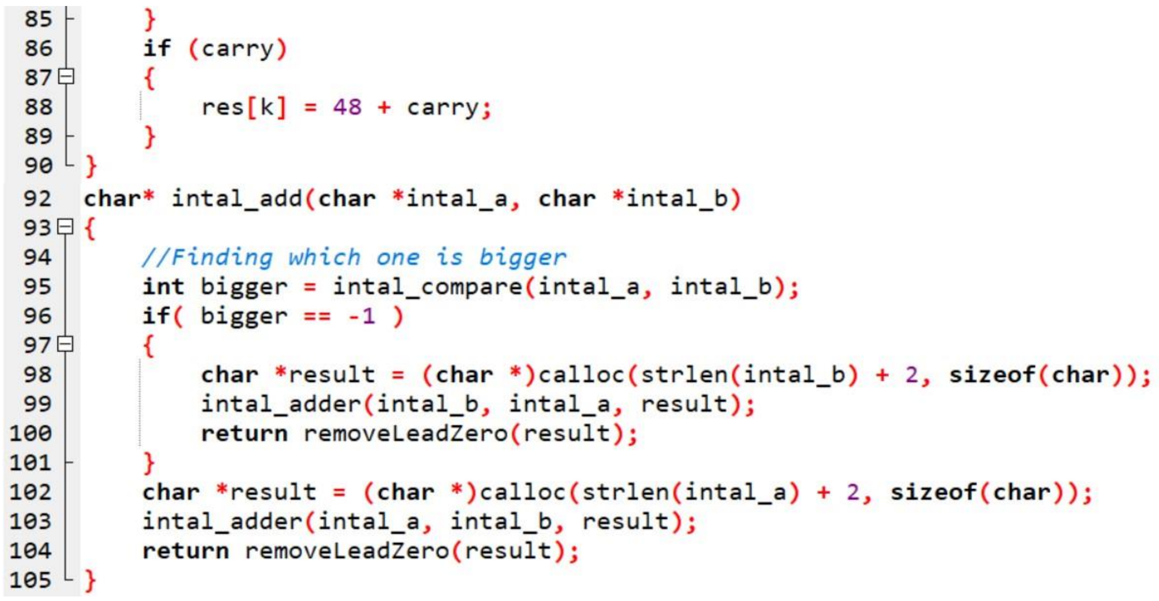
• Additions
• the for loop

Process

- Compare both the INTALs and find the larger INTAL
- Initialize result INTAL
- Sum of each digit will be `intal_a[i] + intal_b[j] + carry - 96`, where `intal_a[i]` is the ith digit and `intal_b[j]` is the jth and `carry` is a flag in case there has been a carryover from the previous digit addition. Finally, you subtract 96 to get the numerical value.
- `result[k]` will be 48 `+ numerical value obtained in previous step mod 10`
- `carry` will be `sum divided by 10`
- Repeat the process through the entire INTAL.
- In case there is a carry at the end, initialize `result[k]` to 48+carry



```c
static void intal_adder(const char *intal1, const char *intal2, char *res)
{
    //printf("Reached intal_adder\n");
    int l1 = strlen(intal1) - 1;
    int l2 = strlen(intal2) - 1;
    int sum = 0, carry = 0, k = l1 + 1;
    //printf("%d %d %d\n",l1,l2,k);
    for (int i = 0; i <= k; i++)
        res[i] = '0';
    while (l2 >= 0)
    {
        sum = intal1[l1] + intal2[l2] + carry - 96;
        res[k] = 48 + sum % 10;
        carry = sum / 10;
        --l1;
        --l2;
        --k;
    }
    while (l1 >= 0)
    {
        sum = intal1[l1] + carry - 48;
        res[k] = 48 + sum % 10;
        carry = sum / 10;
        --l1;
        --k;
    }
```



```c
    }
    if (carry)
    {
        res[k] = 48 + carry;
    }
}
char* intal_add(char *intal_a, char *intal_b)
{
    //Finding which one is bigger
    int bigger = intal_compare(intal_a, intal_b);
    if( bigger == -1 )
    {
        char *result = (char *)calloc(strlen(intal_b) + 2, sizeof(char));
        intal_adder(intal_b, intal_a, result);
        return removeLeadZero(result);
    }
    char *result = (char *)calloc(strlen(intal_a) + 2, sizeof(char));
    intal_adder(intal_a, intal_b, result);
    return removeLeadZero(result);
}
```

Expected Outcome

Your function should be able to do basic addition on the given INTALs and return the result.

Task 3

More Arithmetic Functions

There are more arithmetic functions using same logics. Checkout intal.c file for all operations performs in this project.

Implementation

To test your implementation of the above functions, refer to my Github repository **INTAL-BigIntiger**, specifically the sample_test_case.c file which contains a suite of test cases for the above-implemented functions. You may clone the repo of this project.
For complete implementation run the implementation.c file or you can copy it and run it on a c compiler. If you are a mac user you can run it in the vs code or feel free to use other applications but if you are in Windows or other than mac os run it on online gcc compiler