

Forum:[Hack Forums](#)

Thread: [Subtracting arbitrary large integers in C](#)

kplishere - November 29, 2018, 4:00 pm

Hi,
long post ahead. I much appreciate any kind of help.

I'm doing my home assignment which includes performing arithmetics with arbitrary large numbers that are too long to fit in basic data types such as int or long long. Now, this means that if I want to do arithmetics with these numbers, I need to write my own functions to do it, which are based on the methods used to perform addition/subtraction on paper. Right now, I'm struggling with subtraction.

Please note: I'm a beginner to C. What I do is very likely not to be the most efficient solution, but this is what I could figure out so far.

I store a number in a structure like this:

```
Code typedef struct Arbitrary_Large_Integer
{
    char digits[];
} ALI;
```

I know that it could be easier to use char arrays instead of a structure with a single member which barely makes sense, but I'm forced to put structures in my code this time (that's a requirement for my assignment).

About the function: I want to know the difference of number n and a, both stored in char arrays in ALI structures. Basically, what I'm doing is initialising two integers (temp_n and temp_a) with the current digits of n and a, subtracting them and placing the result in a new ALI instance named k. If the j-th digits of a is greater than the i-th digit of n, then I add 10 to the digit of n, finish the subtraction, and in the next turn, I increase temp_a by one. The value of number a certainly falls between 1 and n - 1 (that's given). If a is shorter than n, as soon as I reach the last digits of a, I put the remaining digits of n to the result array k. And I do this all backwards, so the initialising value of i would be the size of n - 1.

So far, that's primary school maths. I wrote a code which is supposed to perform this, yet I get incorrect results every time.

Here is the function:

```
Code ALI *subtraction(ALI n, ALI a, int nLength, int aLength)
{
    ALI *result;
    result = (ALI*)malloc(nLength * sizeof(ALI));
    if (result == NULL)
        printf("ERROR");

    int temp_n, temp_a, difference;
    int i = nLength - 1; //iterator for number 'n'
    int j = aLength - 1; //iterator for number 'a'
    int k = 0; //iterator for number 'k', n - a = k
```

```
bool carry = false; //to decide whether a carry is needed or not the turn
```

```
for (i; i >= 0; i--)
```

```
{  
    //subtracting 48 from n.digits[i], so temp_n gets the actual number  
    //and not its ASCII code when the value is passed  
    temp_n = n.digits[i] - ASCII_CONVERT;  
    temp_a = a.digits[j] - ASCII_CONVERT;
```

```
    //Performing subtraction the same way as it's used on paper
```

```
    if (carry) //if there is carry, a needs to be increased by one
```

```
{  
    temp_a++;  
    carry = false;  
}
```

```
    if (temp_n >= temp_a)
```

```
    {  
        difference = temp_n - temp_a;  
    }
```

```
    //I wrote else if instead of else so I can clearly see the condition
```

```
    else if (temp_a > temp_n)
```

```
{  
    temp_n += 10;  
    difference = temp_n - temp_a;  
    carry = true;  
}
```

```
    //placing the difference in array k, but first converting it back to ASCII
```

```
    result->digits[k] = difference + ASCII_CONVERT;
```

```
    k++;
```

```
    //n is certainly longer than a, so after every subtraction is performed on a's digits,
```

```
    //I place the remaining digits of n in k
```

```
    if (j == 0)
```

```
{  
    for (int l = i - 1; l >= 0; l--)  
    {  
        result->digits[k] = n.digits[l];  
        k++;  
    }
```

```
    //don't forget to close the array
```

```
    result->digits[k] = '\0';
```

```
    break;
```

```
}
```

```
j--;
```

```
}
```

```
    //reverse the result array
```

```
    _strrev(result->digits);
```

```
    return result;
```

```
}
```

First "turn":

imgur.com/5a9ksmZ.png

can't post embeddd images until I reach 50 points...

It seems like when the array is passed to the function, its value changes for some reason. I can't figure out what's wrong with it.

This is a quite long thread, so I'd be more than grateful if one could help me. I don't speak native English, I hope I could draw up my problem clearly.

Thanks!

I managed to solve the problem

Europa - January 10, 2019, 6:59 pm

Code typedef struct Arbitrary_Large_Integer

```
{  
    char digits[];  
} ALI;
```

First off typedef is not a valid standard C structure. The Flexible Array Member (FAM) .digits must be accompanied by at least one more prior named member in addition to the flexible array member. It's recommend to put .nLength as the first member.

Since your code is using a non-standard C, watch out for nLength sizeof(ALI) may be the same as nLength 0.

You're also attempting to use .digits as a string with _strrev(), themalloc() is too small by 1, at least.

Also malloc(0)?