Projeto

Projeto que compõe a nota do curso de programação estruturada da Universidade Federal do ABC.

Recursos

- Linguagem C;
- Bibiotecas da linguagem C:
    - stdio.h;
    - string.h;
    - stdlib.h;

Relatório

- Nome: Gustavo Jun Miyamoto Hassegawa
- RA: 11202321477

Organização

Arquivos

O projeto foi dividido em 3 partes, um arquivo bignumber.h responsável pela interface, nesse arquivo foi definido os nomes das operações e os structs Nó e BigNumber.

Um arquivo bignumber.c responsável pela implementação, o arquivo possui um conjunto de algoritmos que realizam as operações como criar um BigNumber, adicionar dígitos, soma, subtração, multiplicação, divisão etc.

E por fim o client.c que é o arquivo cliente, que utiliza as operações definidas anteriormente. No andar do projeto optei por não fazer uma operação de receber a entrada do usuário e guardar a resposta na memória, o mecanismo de entrada foi implementado na função main do arquivo client.c.

Entrada

Para a implementação da leitura em que o programa recebe todos os dados de entrada e somente ao final dos dados imprime todas as respostas foi implementado um sistema de array dinâmico, quando o vetor não tem mais espaço disponível para o armazenamento de informações é realocado o dobro de memória para ele. O array armazena dentro de si as respostas das operações e assim entradas podem ser apagadas, liberando espaço na memória para mais operações.

Ao início do programa, o usuário insere os dois números e o sinal da operação desejada, o programa então identifica a operação pelo sinal e armazena a resposta dentro de um vetor, que no término da entrada irá imprimir as respostas em ordem e liberar a memória dos elementos do vetor e vetor.

Tipos Implementados

A estrutura do tipo bignumber foi implementada utilizando uma lista duplamente ligada, onde um nó aponta tanto para o nó anterior quanto para o posterior, e dentro do nó era armazenado um dígito de 0 a 9, ou seja cada nó seria uma casa decimal de um número.

Estrutura do node

A esturtura do tipo node possui a variável, do tipo short int, **digit** que armazena um único dígito do bignumber e dois ponteiros para node **prev**, que se refere ao nó anterior e **next** que se refere para o nó seguinte. Para o bom funcionamento do código a variável digit só poderia assumir valores de 0 a 9 e caso não existisse um nó posterior ou anterior os ponteiros apontariam para NULL.

```
typedef struct node {
    short int digit;
    struct node *prev;
    struct node *next;
} *Node;
```

Estrutura do bignumber

Na implementação de diferentes funções se tornava mais eficaz percorrer os dígitos do bignumber da maior casa para a menor e em outros casos da menor para a maior, então a estrutura bignumber foi implementada com dois ponteiros para node **begin** e **end** que apontam para o dígito de maior e de menor casa decimal respectivamente.

Além disso, dentro da estrutura do tipo bignumber consta a variável, tipo bool, **negative** que caso o número seja negativo negative será verdadeiro. E a variável, tipo int, **size** que se refere ao número de nós que o bignumber possui.

```
typedef struct bignumber {
    Node begin;
    Node end;
    int size;
    bool negative;
} *BigNumber;
```

Funcionalidades Implementadas corretamente

Criação de BigNumber

- **BigNumber create_bignumber()**

  A função create_bignumber() não recebe nenhum parâmetro e retorna um BigNumber vazio, ou seja um bignumber onde begin e end apontam para NULL, size é igual a zero e negative por padrão será falso.

- **BigNumber create_bignumber_zero()**

  A função create_bignumber_zero() diferente da create_bignumber() devolve um bignumber composto apenas pelo dígito "0", inicialmente esta cria um nó em que prev e next apontam para NULL e a variável digit é igual a zero, então assimila as varíaveis begin e end do bignumber para que apontem para o nó criado, size será igual à um e ao final retorna o bignumber.

Adição ou remoção de dígitos

- **void add_digit_end(BigNumber number, int digit)**

  A função add_digit_end adiciona um dígito no final do BigNumber, para isso, recebe a variável number que é um BigNumber e uma variável digit que é um int, então, aloca memória para um novo nó que será adicionado ao final da sequência do bignumber. Então o endereço prev do novo nó aponta para o nó do final de number, caso number seja um BigNumber vazio, a variável begin e end de number será o endereço do novo nó, uma vez que, não há nenhum nó na lista, caso não seja um BigNumber vazio o nó final do bignumber apontará agora para o novo nó e a variável end apontará para o nó adicionado.

- **void add_digit_head(BigNumber number, int digit)**

  A função add_digit_head é semelhante à função add_digit_end, mas adiciona um dígito no inicio do BigNumber.

- **void erase_digit_head(BigNumber number)**

  A função erase_digit_head elimina o dígito do início da lista ligada, ou seja, apaga o dígito de maior casa decimal do número, para isso, recebe a variável number que é um BigNumber, armazena o valor de endereço do segundo digit em uma variável temporária que substitui o valor begin de number, prev passa para apontar para NULL e a variável size é reduzida em uma unidade. O dígito que foi apagado deve ter sua memória liberada, uma vez que para criar os nós foi necessário alocar memória para o tamanho de um node.

- **void erase_digit_end(BigNumber number)**

  A função erase_digit_end faz a mesma coisa que erase_digit_head, mas para o último dígito.

Leitura, impressão e liberação de memória do BigNumber

- **int return_digit(char character)**

  A função return_digit recebe uma variável chamada character do tipo char e retorna um int, caso character seja um valor de 0 a 9, a função retornará o valor correspondente, caso contrário retorna -1.

- **void read_bignumber(BigNumber number)**

  A função read_bignumber recebe uma variável number do tipo BigNumber e lê a entrada do usuário armazenando-a em number. A função recebe a entrada do usuário enquanto a tecla ENTER não for pressionada ou detectar o final de entrada (End of File). Caso o usuário digite o caractere '-' o programa interpretará que se trata da entrada de um número negativo, assim alterando a variável negative de number para true. Além disso, só irá adicionar digitos que estão entre 0 e 9.

- **BigNumber char_bignumber(char *string)**

  O método char_bignumber recebe um parâmetro do tipo char *, string, que possui apenas dígitos e opcionalmente um dígito de sinal negativo no início.

- **void print_bignumber(BigNumber number)**

  A função print_bignumber recebe uma variável BigNumber chamada number, e imprime os dígitos armazenados na lista ligada. Inicialmente ocorre a

verificação se number é negativo, caso seja negativo o programa imprime o caractere '-', caso contrário não imprime nada. Após a verificação, imprime todos os digitos de number, começando pelo primeiro nó da lista ligada, até que um nó aponte para NULL, o que significa que o número acabou.

- **void erase_bignumber(BigNumber number)**

  A função erase_bignumber recebe um BigNumber chamado number que se deseja apagar, e libera a sua memória, como o tipo BigNumber, basicamente, se trata de uma lista ligada é necessário liberar a memória alocada em cada nó e após isso liberar a memória da variável number.

Normalização do BigNumber

- **void delete_left_zeros(BigNumber number)**

  A função delete_left_zeros apaga todos os zeros que estão a esquerda do número, ou seja remove dígitos que não representam importância no número, assim, liberando mais espaço na memória. O comando recebe uma variável BigNumber chamada number, que apagará todos os dígitos iguais à zero até encontrar um número diferente de zero.

  Além de padronizar como os números serão armazenados, isso faz com que menos memória seja utilizada.

- **void node_modularizer(BigNumber number)**

  O módulo node_modularizer recebe uma variável number do tipo BigNumber, a função modulariza o number para o padrão desejado:

  - Dígitos entre 0 e 9;
  - Sem Dígitos negativos;
  - Sem zeros à esquerda;

  Caso o primeiro dígito seja negativo, o atributo negative de number será verdadeiro. Quando um número maior que 9 é detectado o seu valor é modularizado no valor dez (currentNode->digit %= 10), e no dígito seguinte é adicionado o valor da divisão interia do digito antigo por dez.

Comparações entre dois BigNumber

- **int is_equal(BigNumber a, BigNumber b)**

  O método is_equal recebe dois elementos, a e b, do tipo BigNumber e verifica se são iguais, a primeira verificação ocorre verificando se ambos são positivos ou negativos, após isso se verifica se ambos tem o mesmo tamanho pelo atributo size, e por fim verifica se todos os dígitos são iguais, caso não passe em algum teste a função retorna 0, padronizado como resposta negativa, caso contrário retorna 1.

- **int is_bigger(BigNumber a, BigNumber b)**

  O procedimento is_bigger recebe dois parâmetros, tipo BigNumber, a e b, então, a função verifica se a é maior que b, caso seja verdadeiro retorna um int 1, caso contrário retorna 0, ou seja, b é maior ou igual que a. Inicialmente o programa verifica se os números são positivos ou negativos e se possuem tamanhos diferentes, caso passem nestes testes é verificado qual é maior ou menor percorrendo os dígitos da maior casa decimal até a menor, assim, quando um dígito for maior que o outro digito o programa para e retorna a reposta.

Operações matemáticas

- **BigNumber sum_bignumber(BigNumber number1, BigNumber number2)**

  A função sum_bignumber recebe duas variáveis, tipo BigNumber, number1 e number2, e retorna a soma desses dois números. Inicialmente a função identifica os sinais dos números a fim de facilitar a operação.

  - Caso 1 (base): number1 e number2 >= 0 : number1 + number2;
  - Caso 2: number1 >= 0 e number2 < 0 : number1 - number2;
  - Caso 3: number1 < 0 e number2 >= 0 : number2 - number1;
  - Caso 4: number1 < 0 e number2 < 0 : - (number 1 + number2).

  Ao separar as possíveis combinações nesses quatro casos podemos ter uma organização melhor do código para diferentes cenários de entrada e em menos linhas conseguir fazer um código que funcione para diferentes casos.

  O caso 2, é facilmente identificável verificando se number1 é positivo e number 2 é negativo, e sua operação na verdade se trata de uma subtração de number1 por number2, ação realizada pelo módulo sub_bignumber que será explicado posteriormente.

  O caso 3, ocorre quando number1 é negativo e number2 é positivo, e semelhante ao caso passado, se trata também de uma operação de subtração, mas, agora de number2 por number1

  Observando o caso 1 e caso 4, basicamente, seria somar os módulos de number1 e number2 e retornar o valor positivo se ambos forem positivos ou o valor negativo caso ambos sejam negativos.

  A soma foi implementada da mesma maneira que o método convêncional lecionado em escolas de ensino fundamental, onde cada dígito é somado com o dígito da sua casa correspondente, para isso, percorremos ambos os números do final até o começo de ambos ou números. Após somarmos todos os digitos agora basta chamarmos a função node_modularizer para a reposta correta, uma vez que a função soma salva alguns digitos com números maiores que 9.

- **void sum_bignumber_void(BigNumber number, BigNumber out)**

  O método sum_bignumber_void recebe dois BigNumber, number e out, e realiza a soma de number em out, armazenando a resposta em out. Se tornou útil em casos em que era necessário realizar sucessivas somas em uma mesma variável do tipo BigNumber.

  Funciona somente para casos em que number e out são positivos.

- **BigNumber sub_bignumber(BigNumber minuend, BigNumber subtrahend)**

  O procedimento sub_bignumber recebe os dois parâmetros minuend e subtrahend, do tipo BigNumber, que são, respectivamente, os operadores minuendo e subtraendo e uma operação de subtração.

A subtração foi implementada com o mesmo método que é ensinado em escolas no ensino fundamental, onde se subtrai cada digito em sua respectiva casa decimal e caso essa subtração de digitos resulte em um número negativo e exista algum digito a frente, o digito posterior é decrescido em 1 e o digito anterior é acrescido em 10.

Com o mesmo objetivo do caso das somas, facilitar a operação de acordo com o sinal das entradas é possível obter os seguintes casos:

- Caso 1 (base): minuendo e subtraendo >= 0 : minuendo - subtraendo;
- Caso 2: minuendo >= 0 e subtraendo < 0 : minuendo + subtraendo;
- Caso 3: minuendo < 0 e subtraendo >= 0 : - (minuendo + subtraendo);
- Caso 4: minuendo < 0 e subtraendo < 0 : subtraendo - minuendo.

- **void sub_bignumber_void(BigNumber number, BigNumber out)**

  A função sub_bignumber_void recebe dois parâmetros do tipo BigNumber, number e out, e realiza a operação de subtração de out por number e armazena o resultado em out. Foi implementado somente para casos positivos.

- **BigNumber multi_bignumber(BigNumber multiplicand, BigNumber multiplier)**

  O módulo multi_bignumber recebe dois parâmetros do tipo BigNumber, multiplicand e multiplier, respectivamente o multiplicando e multiplicador de uma multiplicação, e devolve um BigNumber contendo o valor da operação de multiplicação do multiplicando pelo multiplicador.

  A multiplicação foi implementada com o método usual, no qual se multiplica um dígito do multiplicador pelo multiplicando e caso o multiplicador tenha dígitos além da casa das unidades se multiplica 10 n vezes sendo que n é o index dos dígitos, por exemplo, 0 número 78945, 5 é o index 0, 4 é o index 1, 9 é o index 2, 8 é o index 3 e 7 é o index 4.

  Em alguns casos, podemos acelerar a resposta do programa por já sabermos as respostas esperadas, por exemplo no caso em que se multiplica algo por 0, independente do número, sempre a resposta será 0.

- **BigNumber div_bignumber(BigNumber dividend, BigNumber divisor)**

  A função div_bignumber recebe dois argumentos do tipo BigNumber, dividend e divisor que são, respectivamente, os elementos dividendo e divisor de uma divisão, a função realiza uma divisão inteira e retorna o valor no tipo BigNumber.

  A divisão foi implementada utilizando o método usual de resolver uma divisão inteira, por se tratar de uma divisão inteira, divisor sempre deve ser menor ou igual ao dividendo, caso contrário a resposta sempre será zero.

Outras:

- **BigNumber identify(BigNumber numberA, BigNumber numberB, char operation)**

  O método identify recebe três parâmetros, dois do tipo BigNumber e um char que identifica a operação matemática desejada. A função devolve a resposta da operação de acordo com o char escolhido.