Atividade prática - BigInt
Professor Wladimir A. Tavares

BigInt

A estrutura `bigint` foi projetada para representar números inteiros de grande tamanho, onde cada posição do vetor digit armazena um dígito individual do número no formato de caracteres e o campo size armazena a quantidade de dígitos usados.

```
typedef struct bigInt {
    char digit[101];
    int size;
} bigint;
```

Exemplo de Uso

Neste exemplo, `num` é uma variável do tipo `bigint`, que representa o número 123. No entanto, os dígitos do número são armazenados de maneira invertida para facilitar os cálculos, ou seja, o número 123 é armazenado como '3', '2' e '1' nas posições 0, 1 e 2 do vetor `num.digit`, respectivamente. O campo size é atribuído o valor 3, indicando que há três dígitos armazenados no vetor. O caractere especial $'\backslash 0'$é colocado na posição 3 para marcar o final da string, embora o número em si ocupe apenas as três primeiras posições.

```
bigint num;
num.size = 3;
num.digit[0] = '3';
num.digit[1] = '2';
num.digit[2] = '1';
num.digit[3] = '\0';
```

Criando um bigint

A função `init_bigint` tem o objetivo de inicializar uma variável do tipo bigint a partir de uma string de caracteres (char s[]) que representa um número. Durante o processo de inicialização, os dígitos da string são invertidos e armazenados na estrutura bigint para facilitar operações aritméticas.

```
void reverse(char s[]){
    int i = 0;
    int j = strlen(s)-1;
    while(i<j){
        char t = s[i];
        s[i] = s[j];
        s[j] = t;
        i++;
        j--;
    }
}
bigint init_bigint(char s[]){
    bigint b;
```

```
    b.size = strlen(s);
    strcpy(b.digit, s);
    reverse(b.digit);
    return b;
}

void imprime_bigint(bigint b){
    printf("bigint: ");
    for(int i = b.size-1; i >= 0; i--)
        printf("%c", b.digit[i]);
    printf("\n");
}


int main(){
    bigint a = init_bigint("123");
    imprime_bigint(a);
}
```

incremento

A função incremento foi projetada para incrementar em 1 um número inteiro muito grande, representado por um tipo especial chamado bigint. Esse tipo permite trabalhar com números que excedem o tamanho dos tipos numéricos padrão em C, como `int` ou `long long`.

```
bigint incremento(bigint b){
    bigint c;
    c.size = b.size;
    strcpy(c.digit, b.digit);

    for(int i = 0; i < c.size; i++){
        if( c.digit[i] < '9'){
            c.digit[i]++;
            return c;
        }else{
            c.digit[i] = '0';
        }
    }
    c.digit[c.size] = '1';
    c.size++;
    return c;
}


int main(){
    bigint a = init_bigint("9999");
    imprime_bigint(a); //bigint: 9999
    a = incremento(a);
    imprime_bigint(a);//bigint: 10000
}
```

Exercícios

1. Implemente a função maior(a,b), que compara dois números inteiros grandes representados por objetos do tipo bigint e retorna um dos seguintes valores:

- -1: se o número representado por a é maior que o número representado por b;
- 0: se os números representados por a e b são iguais;
- 1: se o número representado por b é maior que o número representado por a.

```
int maior(bigint a, bigint b){
    return 0;
}
```

2. Implemente a função soma, que recebe dois números grandes representados como objetos do tipo bigint e retorna um novo número grande, também representado como um objeto bigint, contendo o resultado da soma.

```
bigint soma(bigint a, bigint b){
    bigint c;
    return c;
}
```

3. Implemente a função shift, que desloca os dígitos de um número grande para a esquerda ou para a direita, conforme o valor de um inteiro n:

   - Se $n > 0$, o número deve ser deslocado para a esquerda, adicionando n zeros ao final do número.
   - Se $n < 0$, o número deve ser deslocado para a direita, removendo até $|n|$ dígitos do final do número. Caso o número tenha menos dígitos do que $|n|$, o resultado deve ser 0

```
bigint shift(bigint a, int n){

}
```

Exemplo de uso:

```
bigint a = init_bigint("123");
a = shift(a, 3); //123000

bigint a = init_bigint("12378");
a = shift(a, -3); //12
```

4. Seu Boquinha é o dono de um bar famoso por suas habilidades únicas de cálculo. Ele tem um jeito peculiar de realizar somas: quando a operação resulta em um ”vai-um”tradicional (carregar um valor adicional para o próximo dígito), ele pode inverter o valor do ”vai-um”para maximizar o valor da soma. Por exemplo:

   Em vez de calcular 6+9=15, o Seu Boquinha inverte o ”vai-um”, resultando em 51

   Assim, a soma entre dois números pode ter resultados muito diferentes dos cálculos matemáticos tradicionais.

   Implemente a função `soma_boquinha`, que calcula a soma de dois números grandes de acordo com as regras do Seu Boquinha. A soma deve ser realizada dígito a dígito, e o ”vai-um”deve ser sempre o maior valor entre os dígitos da soma.

```
bigint soma_boquinha(bigint a, bigint b){
    bigint c = init_bigint("");
    return c;
}
```