Departamento de Informática e Matemática Aplicada – DIMAp – UFRN
Algoritmos e Estrutura de Dados I – DIM0110.0
Prática de Algoritmos e Estrutura de Dados I – DIM0110.1
Selan R. dos Santos – 16/10/2009

◁ Projeto de Programação ▷

Simulando inteiros longos e suas operações

Apresentação

O objetivo desta lista de exercícios é fornecer duas aplicações que devem fazer uso de *lista encadeada* para sua solução. Um dos pré-requisitos deste exercício é que estas estruturas de dados já estejam implementadas na forma de classes genéricas em C++. Espera-se que com estas duas aplicações seja possível perceber a importância das listas encadeadas como estrutura de dados na solução de problemas práticos.

Além da aplicação de listas encadeadas, este trabalho oferece uma oportunidade de ampliar sua experiência com importantes elementos da linguagem C++, como classes, objetos, construtor cópia, sobrecarga de operadores, bem como testagem e documentação de código.

1 Introdução

Os tipos básicos da linguagem C++ que representam os números inteiros posuem uma faixa de valores que são capazes de representar. A Tabela 1 apresenta os limites para os tipos básicos que representam números inteiros em C++.

Tipo Básico	*Memória*	*Intervalo Representado*
`unsigned short (int)`	2 bytes	0 a 65.535
`short`	2 bytes	-32.768 a 32.767
`unsigned int`	4 bytes	0 a 4294967295
`int`	4 bytes	-2147483648 a 2147483647
`unsigned long (int)`	4 bytes	0 a 4294967295
`long (int)`	4 bytes	-2147483648 a 2147483647
`unsigned long long (int)`	8 bytes	0 a 18446744073709551615
`long long (int)`	8 bytes	-9223372036854775808 a 9223372036854775807

Tabela 1: Faixa de valores dos tipos básicos do C++. O uso de `(int)` indica que a presenta do `int` na declaração do tipo é facultativa.

De acordo com os dados da Tabela 1, a maior representação permitida para um número inteiro terá, no máximo, 20 dígitos. Na prática existem várias aplicações, como em cálculo de chaves em Criptografia, para as quais são necessários manipular inteiros com dezenas, centenas e até milhares de dígitos.

O propósito deste exercício é criar uma classe especial de inteiro para lidar com números inteiros que apresentam uma quantidade arbitrariamente grande de dígitos. Uma forma de implementar esta classe é armazenar os dígitos do número inteiro em uma lista encadeada (um dígito por nó).

Você deve assumir que a classe `BigInt` deve tratar números inteiros positivos e negativos, bem como prover operações aritméticas básicas e operadores relacionais. Esta tarefa pode ser dividida em várias partes que podem ser implementadas na ordem sugerida ou na ordem que mais lhe convier.

- ⋆ **Parte 1:** Criar a base da classe `BigInt`, com membros tradicionais como construtor, construtor cópia, destrutor e operador de atribuição. Sua classe deve prover, pelo menos, três versões de construtor:
  (a) sem parâmetros, que deve criar um *inteiro grande* com valor zero;
  (b) que recebe um inteiro e cria um *inteiro grande* correspondente, e;
  (c) que recebe uma cadeia de caracteres e cria um *inteiro grande* correspondente.
- ⋆ **Parte 2:** Adicione a sua classe a habilidade de aplicar operadores relacionais (`<`, `<=`, `>`, `>=`, `==`, `!=`) sobre um par de `BigInt`. Estas operações devem ser disponibilizadas na forma de *sobrecarga de operador*.
- ⋆ **Parte 3:** Adicione a sua classe a habilidade de adicionar e subtrair um par de `BigInt`. Estas operações devem ser disponibilizadas na forma de *sobrecarga de operador*.
- ⋆ **Parte 4:** Adicione a sua classe a habilidade de multiplicar, dividir e calcular o resto da divisão inteira entre um par de `BigInt`. Estas operações devem ser disponibilizadas na forma de *sobrecarga de operador*.
- ⋆ **Parte 5:** Finalmente acrescente a sua classe a capacidade de exibir o inteiro através da sobrecarga de `<<` e também de receber um novo `BigInt` através da sobrecarga do operador `>>`.

Certifique-se de criar um código cliente para testar cada uma das funcionalidades solicitadas e também documentar seu código. Também realize testes para identificar qualquer tipo de vazamento de memória. Confira o Código 1 que apresenta um exemplo simples de código cliente que utiliza a classe `BigInt`.

Código 1 Exemplo de código-cliente que utiliza a classe `BigInt`.

```
1  int main( )
2  {
3     BigInt x( "123456789" ); // Versão (c)
4     BigInt y( 23456789 );    // Versão (b)
5     BigInt z( 456789 );      // Versão (b)
6     BigInt resultado;        // Versão (a)
7     resultado = x * y + z;   // Operadores aritiméticos e atribuição
8     // Sobrecarga de <<.
9     cout << x << " * " << y << " + " << z << " = " << resultado << endl;
10
11    BigInt esperado( "2895899850647310" );
12    assert( esperado == resultado ); // Operador relacional
13    cout << "---> Ok!\n";
14
15    return EXIT_SUCCESS;
16 }
```

2 Utilizando a Classe `BigInt`

Como forma de testar sua recém implementada classe `BigInt`, você deve tentar resolver dois problemas retirados da famosa competição internacional de programação: A Maratona de Programação da ACM.

2.1 Problema 324: Freqüência de Fatorial

Em uma tentativa desesperada de impulsionar seu decadente negócio de leitura de mão, a Madame Fênix decidiu oferecer, de lambuja, uma análise numerológica para os clientes que solicitarem seus serviços divinatórios. Ela conseguiu convencer seus crédulos clientes de que a freqüência de ocorrência dos dígitos da representação decimal de fatoriais são, na verdade, códigos secretos usados desde a antigüidade para a decifrar o futuro. Ao contrário do que acontece com a leitura da mão, Madame Fênix não pode simplesmente "invocar" os poderes secretos das freqüências sem as conhecê-las. Sabendo que você está estudando para se tornar um cientista da computação, Madame Fênixa solicitou-lhe que calcule estes valores para ela.

Lembre-se que a definição de $n!$ (fatorial de n) é apenas $1 \times 2 \times 3 \times \cdots \times n$. Como ela espera usar o dia da semana, o dia do mês ou o dia do ano como valor de n, seu programa deve ser capaz de determinar o número de ocorrênias de cada dígito decimal em números tão grandes quanto 366!, o qual possui 781 dígitos.

Entrada e Saída

Os dados de entrada para o programa são simplesmente uma lista de inteiros para o qual a contagem de dígitos deve ser feita. Todos os valores de entrada serão menores do que ou iguais a 366 e maiores do que zero, exceto o último inteiro, que será zero. Não se preocupe em processar o valor zero; apenas encerre o programa ao recebê-lo. O formato de saída não é muito crítico, mas você deve fazer com que seu programa gera resultados que sejam similares aos exemplos apresentados a seguir.

Madame Fênix será eternamente grata por seus seviços; ela pode até realizar uma leitura para você, caso seu programa produza resultados corretos!

Exemplo de Entrada

```
3
8
100
0
```

Exemplo de Saída

```
3! --
   (0)    0    (1)    0    (2)    0    (3)    0    (4)    0
   (5)    0    (6)    1    (7)    0    (8)    0    (9)    0
8! --
   (0)    2    (1)    0    (2)    1    (3)    1    (4)    1
   (5)    0    (6)    0    (7)    0    (8)    0    (9)    0
100! --
   (0)   30    (1)   15    (2)   19    (3)   10    (4)   10
   (5)   14    (6)   19    (7)    7    (8)   14    (9)   20
```

2.2 Problema 495: *Fibonacci Freeze*

Os números de Fibonacci $(0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots)$ são definidos pela recorrência:

$$\begin{cases} F_0 &= 0 \\ F_1 &= 1 \\ F_i &= F_{i-1} + F_{i-2}, \quad \text{para todo } i \geq 2 \end{cases}$$

Escreva um programa para calcular os números de Fibonacci.

Entrada e Saída

A entrada para o seu programa será uma seqüência de números menores ou iguais a 50000, cada um em uma linha separada, especificando qual número de Fibonacci deve ser calculado.

Seu programa deve gerar o número de Fibonacci solicitado para cada valor da entrada, um por linha.

Exemplo de Entrada

```
5
7
11
```

Exemplo de Saída

```
O numero Fibonacci para 5 eh 5
O numero Fibonacci para 7 eh 13
O numero Fibonacci para 11 eh 89
```

3 Avaliação do Programa

O programa completo deverá ser entregue sem erros de compilação, testado e totalmente documentado. O projeto de programação será avaliado sob os seguintes critérios:

- ⋆ Classe corretamente implementada (50 %)
- ⋆ Problemas da maratona corretamente resolvidos com a classe inteiro longo (50 %)
- ◦ Presença de erros de compilação e/ou execução (até −20%)
- ◦ Falta de documentação do programa com Doxygen (até −10%)
- ◦ Vazamento de memória identificado com o valgrind (até −10%)
- ◦ Falta ou incompletude do arquivo README.TXT (até −10%)

A pontuação acima não é definitiva e imutável. Ela serve apenas como um guia de como o trabalho será avaliado em linhas gerais. Desta forma, os professores têm total liberdade para realizar ajustes nas pontuações indicadas visando adequar a pontuação ao nível de dificuldade dos itens solicitados.

4 Tema de Pesquisa

Além do desenvolvimento do projeto de programação especificado, a equipe de trabalho deve realizar uma pesquisa teórica. Como retultado, deve ser elaborado um relatório descritivo para o trabalho e realizada a apresentação oral correspondente.

Vale a pena ressaltar que o relatório gerado deve apresentar as seguintes seções:

- ⋆ **Introdução**: descrição do problema tratado, fornecendo o contexto e deixando claro qual o tema de pesquisa/investigação;
- ⋆ **Revisão ou *Background***: nesta seção você deve citar fontes científicas relacionadas ao seu trabalho (referências bibliográficas) ou fornecer conceitos necessários ao entendimento da teoria que será apresentada mais adiante;
- ⋆ **Metodologia**: esta é a seção principal do relatório, onde deve ser descrito o que foi pesquisado e qual a metodologia utilizada.
- ⋆ **Resultados**: nesta seção deve-se apresentar os resultado obtidos; no caso específico deste projeto pode ser apresentado aqui uma comparação em termos de complexidade e capacidade de segurança de, pelo menos, dois algoritmos de criptografia atualmente em uso.
- ⋆ **Conclusão**: nesta seção você deve sumarizar o propósito da pesquisa, destacando os principais resultados alcançados; por favor, evite comentários óbvios do tipo "(...) foi muito bom estudar esta matéria, pois consegui aprender muito!"

Para este trabalho sua pesquisa deve investigar pelo menos dois algoritmos de criptografia que utilizem inteiros longos para sua implementação. O relatório deve descrever estes algoritmos comparativamente, enfatizando seus princípios, nível de segurança e complexidade temporal.

5 Entrega

O trabalho deve ser desenvolvido em triplas, tentando, dentro do possível, dividir as tarefas igualmente entre os componentes. Porém os componentes devem ser capazes de explicar qualquer trecho de código do programa, mesmo que o código tenha sido desenvolvido pelo outro membro da equipe.

Para a solução deste projeto **é obrigatório** a utilização das classes pilha, fila e lista sequencial que foram desenvolvidas em trabalhos anteriores. Não serão aceitas soluções que utilizem as estruturas de dados da biblioteca STL (e.g. `list`, `stack`, `vector`, etc.).

Você deve submeter dois componentes: o relatório de pesquisa e todo o código fonte correspondente ao projeto em uma pasta. Na pasta do programa deve existir um arquivo denominado de `README.TXT`, no qual a equipe deve indicar: (1) Nome e matrícula da dupla de desenvolvedores; (2) Instruções de como compilar o programa (ou então um `Makefile`); (3) Instruções de como executar o programa; e (4) Indicações de eventuais limitações ou incompletudes do programa.

Eventualmente, algumas duplas poderão ser convocadas para uma entrevista. O objetivo de tal entrevista é comprovar a verdadeira autoria do código entregue. Assim, qualquer um dos componentes da dupla deve ser capaz de explicar qualquer trecho de código do projeto. Trabalhos plagiados receberão nota **zero** automaticamente.

A entrega deve ser feita através da opção Tarefas da turma Virtual do Sigaa, em data divulgada no sistema. Entregas atrasadas terão sua nota final diminuída de um valor proporcional aos dias de atraso. **Somente serão aceitas entregas feita através do Sigaa.**

$\sim$ FIM $\sim$