Ciência da Computação
Estrutura de Dados I
Prof. André Kishimoto

Observações:

- *As instruções sobre entrega, prazo e avaliação estão descritas após o enunciado.*
- *A parte 2 de 2 da Avaliação Parcial 2 (P2) será realizada no **dia 27/05/2021, em horário de aula** (será uma prova individual e com consulta, inclusive à sua implementação da parte 1 de 2 descrita abaixo, e valerá 4,0 pontos).*

Avaliação Parcial 2 (P2), parte 1 de 2 – INDIVIDUAL

Usando sua própria implementação C++ de lista ligada/encadeada (*linked list*), adicione as seguintes operações à implementação da sua lista:

OPERAÇÃO	COMPORTAMENTO
InsertBefore(list, beforeId, id, name)	Cria um nó contendo *id* (número inteiro) e *name* (string) e insere o novo nó antes do nó que contém o id *beforeId*, exceto: - Se a fila estiver vazia, não insere o novo nó na lista e retorna false. - Se não existe um nó na lista com o id *beforeId*, não insere o novo nó na lista e retorna false. Pré-condição: A lista ligada/encadeada *list* é válida. Pós-condição: Se foi possível inserir o novo nó na lista, a lista deve conter o novo nó, na posição correta, e a função retorna true ("nó inserido na lista"). Caso contrário, a função retorna false ("não foi possível inserir nó na lista"), sem inserir o novo nó na lista.
InsertAfter(list, afterId, id, name)	Cria um nó contendo *id* (número inteiro) e *name* (string) e insere o novo nó depois do nó que contém o id *afterId*, exceto: - Se a fila estiver vazia, não insere o novo nó na lista e retorna false. - Se não existe um nó na lista com o id *afterId*, não insere o novo nó na lista e retorna false. Pré-condição: A lista ligada/encadeada *list* é válida. Pós-condição: Se foi possível inserir o novo nó na lista, a lista deve conter o novo nó, na posição correta, e a função retorna true ("nó inserido na lista"). Caso contrário, a função retorna false ("não foi possível inserir nó na lista"), sem inserir o novo nó na lista.

Além de implementar as operações da tabela acima, inclua, como comentário dentro de cada função implementada, qual é a complexidade do algoritmo (tempo de execução). Use a notação Big-O e apresente sua justificativa para cada complexidade indicada.

Para essa parte prática da P2, sua lista pode ser do tipo simplesmente ligada, duplamente ligada ou circular (simplesmente ou duplamente ligada).

O seu arquivo main.cpp deve conter exatamente o código abaixo, com exceção das linhas com comentários TODO, que devem ser substituídos pelas instruções corretas da sua implementação da lista. Caso faça a implementação usando classes, o código do arquivo main.cpp deve ser adaptado de acordo.

```
// main.cpp
// TODO: Inserir sua identificação aqui.
#include <iostream>
#include <clocale>
#include "LinkedList.h"

using namespace std;

void Print(const LinkedList* list)
{
  // TODO: Percorrer todos os nós da lista e imprimir as informações de cada nó,
  // conforme a saída exibida no enunciado.
}

void PrintListInfo(const LinkedList* list)
{
  if (IsEmpty(list))
  {
    cout << "Lista vazia! (" << Count(list) << ")\n\n";
  }
  else
  {
    cout << "Lista (" << Count(list) << "): \n";
    Print(list);
  }
}

void PrintListInfoAfterInsertion(const LinkedList* list, bool didInsert)
{
  if (didInsert)
  {
    PrintListInfo(list);
  }
  else
  {
    cout << "Não foi possível inserir novo nó na lista!\n\n";
  }
}

void PrintListInfoAfterRemoval(const LinkedList* list, Node* node)
{
  if (node != nullptr)
  {
    cout << "Nó removido -> id: " << node->id << ", name: " << node->name << "\n\n";
    PrintListInfo(list);
  }
  else
  {
    cout << "Não foi possível remover o nó da lista!\n\n";
  }
}

int main()
{
  setlocale(LC_ALL, "pt_BR");
```

```
    cout << "*** ESTRUTURA DE DADOS I - Avaliação Parcial 2 (P2) ***\n\n";

    LinkedList* list = Create();
    PrintListInfo(list);

    Append(list, 1, "Carol");
    Append(list, 2, "Eric");
    Append(list, 3, "John");
    PrintListInfo(list);

    bool didInsert = InsertAfter(list, 5, 4, "Leo");
    PrintListInfoAfterInsertion(list, didInsert);

    didInsert = InsertAfter(list, 2, 4, "Leo");
    PrintListInfoAfterInsertion(list, didInsert);

    didInsert = InsertAfter(list, 2, 5, "Julia");
    PrintListInfoAfterInsertion(list, didInsert);

    didInsert = InsertBefore(list, 4, 6, "Lisa");
    PrintListInfoAfterInsertion(list, didInsert);

    Node* temp = RemoveNode(list, 99);
    PrintListInfoAfterRemoval(list, temp);
    // TODO: Liberar memória alocada para o nó que foi removido.

    temp = RemoveNode(list, 2);
    PrintListInfoAfterRemoval(list, temp);
    // TODO: Liberar memória alocada para o nó que foi removido.

    temp = RemoveHead(list);
    PrintListInfoAfterRemoval(list, temp);
    // TODO: Liberar memória alocada para o nó que foi removido.

    temp = RemoveTail(list);
    PrintListInfoAfterRemoval(list, temp);
    // TODO: Liberar memória alocada para o nó que foi removido.

    Clear(list);
    PrintListInfo(list);

    didInsert = InsertAfter(list, 44, 33, "Olga");
    PrintListInfoAfterInsertion(list, didInsert);

    didInsert = InsertBefore(list, 11, 22, "Thomas");
    PrintListInfoAfterInsertion(list, didInsert);

    Append(list, 44, "Bia");
    PrintListInfo(list);

    didInsert = InsertAfter(list, 44, 55, "Angela");
    PrintListInfoAfterInsertion(list, didInsert);

    didInsert = InsertBefore(list, 44, 66, "Karen");
    PrintListInfoAfterInsertion(list, didInsert);

    // TODO: Liberar memória alocada para a lista.

    cout << "Fim.\n";
}
```

A execução do código acima reproduz a seguinte saída:

```
*** ESTRUTURA DE DADOS I - Avaliação Parcial 2 (P2) ***

Lista vazia! (0)

Lista (3):
[1] Carol
[2] Eric
[3] John

Não foi possível inserir novo nó na lista!

Lista (4):
[1] Carol
[2] Eric
[4] Leo
[3] John

Lista (5):
[1] Carol
[2] Eric
[5] Julia
[4] Leo
[3] John

Lista (6):
[1] Carol
[2] Eric
[5] Julia
[6] Lisa
[4] Leo
[3] John

Não foi possível remover o nó da lista!

Nó removido -> id: 2, name: Eric

Lista (5):
[1] Carol
[5] Julia
[6] Lisa
[4] Leo
[3] John

Nó removido -> id: 1, name: Carol

Lista (4):
[5] Julia
[6] Lisa
[4] Leo
[3] John

Nó removido -> id: 3, name: John

Lista (3):
[5] Julia
[6] Lisa
[4] Leo

Lista vazia! (0)

Não foi possível inserir novo nó na lista!

Não foi possível inserir novo nó na lista!
```

```
Lista (1):
[44] Bia

Lista (2):
[44] Bia
[55] Angela

Lista (3):
[66] Karen
[44] Bia
[55] Angela

Fim.
```

Desenvolvimento (6,0 pontos)

- Sua solução deve ser escrita apenas com a linguagem C++ e não deve usar a STL (projetos usando STL não serão aceitos).
- Tente sempre trabalhar com arquivos .cpp/.h, modularizando o seu código.
- A sua implementação da lista deve satisfazer as instruções da main.cpp listada nesse documento e a execução do código deve reproduzir a mesma saída indicada nas páginas anteriores.

Identificação e referências

- Coloque sua identificação - nome e TIA - no início de cada arquivo de código, como comentário (use // no começo de cada linha que queira comentar).
- Inclua como comentário quaisquer referências (livros, artigos, sites, entre outros) usadas para solucionar o problema.

Entrega

- **Código:** Compacte todos os arquivos .cpp/.h ou o projeto completo criado na IDE que você está usando (mas sem os intermediários como bin e obj) no formato zip OU comite todos os arquivos .cpp/.h ou o projeto completo criado na IDE que você está usando (mas sem os intermediários como bin e obj) em um repositório git.
- **Arquivo texto (.txt):**
  - Se o código está em um repositório git, envie um arquivo txt no Moodle contendo sua identificação e o link do repositório.
- **Prazo de entrega:** via link do Moodle até 27/05/2021 11:00.

Informações importantes sobre critérios de avaliação

Embora essa atividade seja uma avaliação da disciplina, sempre considero que as atividades também podem ser usadas para nos acostumarmos com o mercado de trabalho. Portanto, leve em consideração os seguintes critérios que vou aplicar na avaliação:

- Será descontado 1,0 (um) ponto caso a entrega não respeite o enunciado. Exemplos:
  - O enunciado pede para enviar um arquivo compactado no formato zip, mas o arquivo enviado está no formato rar.
  - O enunciado pede um arquivo texto no formato txt, mas foi enviado um documento do Word.

    - Não há identificação nem referências (caso aplicável) nos arquivos de código.

- Será descontado 1,0 (um) ponto caso o arquivo zip OU o repositório git contenha pastas e arquivos desnecessários.
  Exemplo:
    - Pastas intermediárias criadas no processo de compilação (Debug, obj, bin, ...).

- O projeto deve ser desenvolvido em linguagem C++ e não em linguagem C. Caso a solução apresentada use funcionalidades da linguagem C e que tenham equivalentes em C++, será descontado 2,0 (dois) pontos.
  Atente-se a esse detalhe quando estiver pesquisando e verificando exemplos na internet e outros materiais, principalmente de assuntos que não vimos até o momento (essa atividade pode ser resolvida só com o que foi visto em aula, com suas devidas adaptações).
  Exemplo:
    - Declarar arrays de tamanho variável (padronizado no C99, mas erro em C++ pois não há suporte para VLA), ex. int n = 10; char arr[n];.

- Projeto que possui erros de compilação ou que trava durante a execução automaticamente perde 50% da nota máxima.
  Sobre erros de compilação: considero apenas erros, não há problema se o projeto tiver warnings (apesar que warnings podem avisar possíveis travamentos em tempo de execução, como loop infinito, divisão por zero etc.).
  Quando há necessidade de entrada de dados por parte do usuário, assumo que o usuário vai inserir as informações corretas (ex. tipos de dados corretos), a menos que o enunciado explicite que você deve garantir que os dados de entrada estejam corretos.

- Entregas que são cópias de outros projetos serão automaticamente zeradas. Tenha em mente que códigos em que a pessoa só alterou nomes de variáveis, funções etc. são consideradas como cópia.

Em uma situação profissional, os itens indicados acima atrapalham (e muito) o trabalho da equipe. E o último item é gravíssimo (o ideal também é remover todos os warnings e sempre validar os dados).