Universidade Federal do Ceará - Campus Quixadá
QXD0010 – Estruturas de Dados – Turma 03A
Prof. Atílio Gomes Luiz

Primeiro Projeto

As soluções das questões descritas neste documento devem ser entregues até a meia-noite do dia **25/09/2020** pelo Moodle.

Leia atentamente as instruções abaixo.

Instruções:

- Este trabalho pode ser feito em **dupla** ou **individualmente** e deve ser implementado usando a linguagem de programação C++ **(Não aceitarei mais do que dois alunos por projeto)**
- Coloque a solução de cada questão em uma pasta específica. O seu trabalho deve ser compactado (.zip, .rar, etc.) e enviado para o Moodle na atividade correspondente ao Projeto 01.
- Identifique o seu código-fonte colocando o **nome** e **matrícula** dos integrantes da dupla como comentário no início de seu código.
- Indente corretamente o seu código para facilitar o entendimento.
- As estruturas de dados devem ser implementadas como TAD.
- Os programas-fonte devem estar devidamente organizados e documentados.
- Observação: Lembre-se de desalocar os endereços de memória alocados quando os mesmos não forem mais ser usados.
- **Observação: Qualquer indício de plágio resultará em nota ZERO para todos os envolvidos.**

DICA: COMECE O TRABALHO O QUANTO ANTES.

Questão 1: [Listas Circulares Duplamente Encadeadas] A estrutura de lista simplesmente encadeada, vista durante a aula, caracteriza-se por formar um encadeamento simples entre os nós: cada nó armazena um ponteiro para o próximo elemento da lista. Dessa forma, não temos como percorrer eficientemente os elementos em ordem inversa. O encadeamento simples também dificulta a retirada de um elemento da lista. Mesmo se tivermos o ponteiro do elemento que desejamos retirar, temos de percorrer a lista, elemento por elemento, para encontrar o elemento anterior, pois, dado o ponteiro para um determinado elemento, não temos como acessar diretamente seu elemento anterior.

Para solucionar esses problemas, podemos formar o que chamamos de **listas duplamente encadeadas**. Nelas, cada elemento tem um ponteiro para o próximo elemento e um ponteiro para o elemento anterior. Assim, dado um elemento, podemos acessar os dois elementos adjacentes: o próximo e o anterior. A lista duplamente encadeada pode ou não ter um nó cabeça e pode ou não ser circular, conforme as conveniências do programador. Uma **lista circular duplamente encadeada** é uma lista duplamente encadeada na qual o último elemento da lista passa a ter como próximo o primeiro elemento, que, por sua vez, passa a ter o último como anterior. A Figura 1 ilustra uma lista duplamente encadeada com estrutura circular e a presença de um nó cabeça.

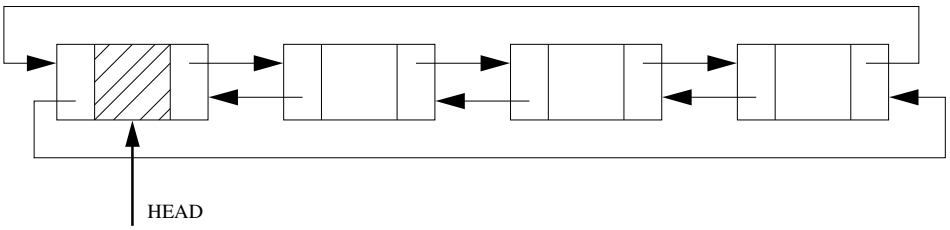


Figura 1: Lista circular duplamente encadeada com nó cabeça.

Problema: Implemente em C++ o Tipo Abstrato de Dados Lista Linear usando como base a estrutura de dados Lista Circular Duplamente Encadeada. A sua estrutura de dados deve ser encapsulada por uma classe chamada `List`, que deve suportar as seguintes operações:

- `List()`: Construtor da classe. Deve iniciar todos os atributos da classe com valores válidos.
- `~List()`: Destrutor da classe. Libera memória previamente alocada.
- `void push_back(int key)`: Insere um inteiro *key* ao final da lista.
- `int pop_back()`: Remove elemento do final da lista e retorna seu valor.
- `void insertAfter(int key, int k)`: Insere um novo nó com valor *key* após o k-ésimo nó da lista.
- `void removeNode(Node *p)`: Remove da lista o nó apontado pelo ponteiro p.
- `void remove(int key)`: Remove da lista a primeira ocorrência do inteiro *key*
- `void removeAll(int key)`: Remove da lista todas as ocorrências do inteiro *key*.

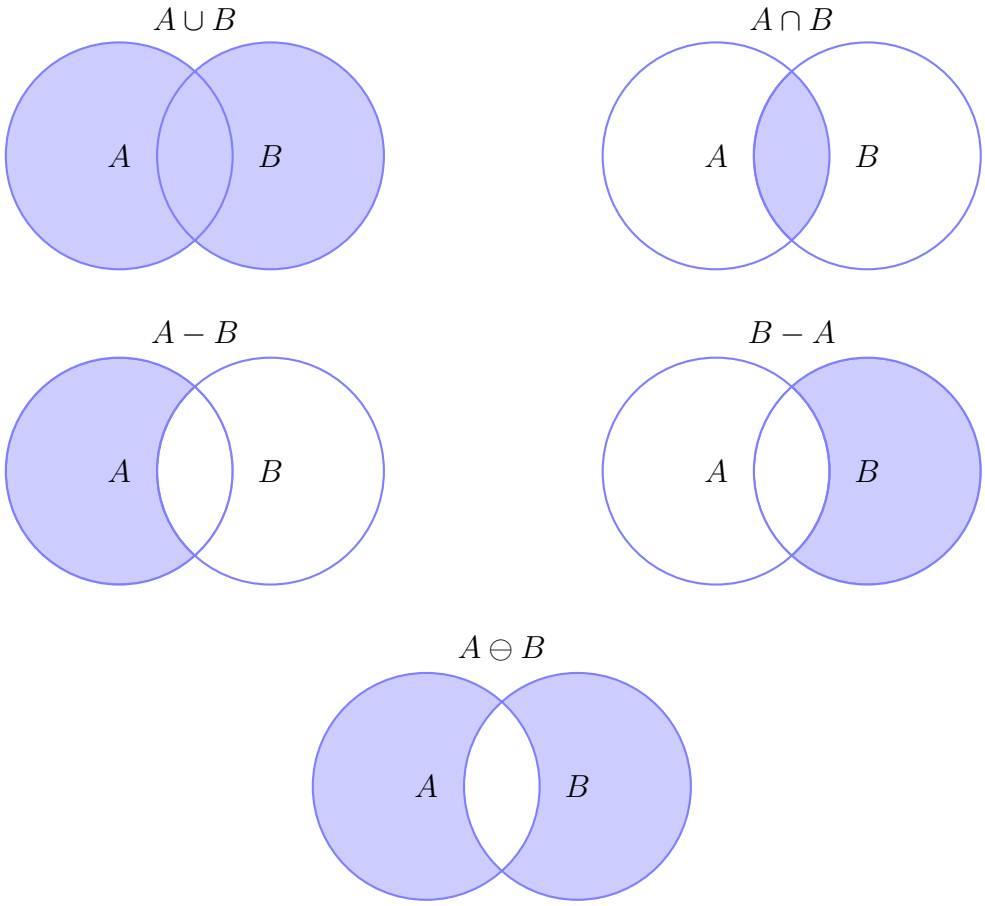
- `int removeNodeAt(int k)`: Remove o k-ésimo nó da lista encadeada e retorna o seu valor. Caso o k-ésimo nó não exista, o programa retorna o valor especial `INT_MIN` definido no cabeçalho `climits`.
- `void print()`: Imprime os elementos da lista.
- `void printReverse()`: Imprime os elementos da lista em ordem reversa.
- `bool empty()`: Retorna `true` se a lista estiver vazia e `false` caso contrário.
- `int size()`: Retorna o número de nós da lista.
- `void clear()`: Remove todos os elementos da lista e deixa apenas o nó cabeça.
- `void concat(List *lst)`: Concatena a lista atual com a lista `lst` passada por parâmetro. Após essa operação ser executada, `lst` será uma lista vazia, ou seja, o único nó de `lst` será o nó cabeça.
- `List *copy()`: Retorna um ponteiro para uma cópia desta lista.
- `void copyArray(int n, int arr[])`: Copia os elementos do array *arr* para a lista. O array *arr* tem n elementos. Todos os elementos anteriores da lista são mantidos e os elementos do array *arr* devem ser adicionados após os elementos originais.
- `bool equal(List *lst)`: Determina se a lista passada por parâmetro é igual à lista em questão. Duas listas são iguais se elas possuem o mesmo tamanho e o valor do k-ésimo elemento da primeira lista é igual ao k-ésimo elemento da segunda lista.
- `List* separate(int n)`: Recebe como parâmetro um valor inteiro n e divide a lista em duas, de forma à segunda lista começar no primeiro nó logo após a primeira ocorrência de n na lista original. A função deve retornar um ponteiro para a segunda subdivisão da lista original, enquanto a cabeça da lista original deve continuar apontando para o primeiro elemento da primeira lista, caso ele não tenha sido o primeiro a ter valor n.
- `void merge_lists(List *list2)`: Recebe uma `List` como parâmetro e constrói uma nova lista com a intercalação dos nós da lista original com os nós da lista passada por parâmetro. Ao final desta operação, `list2` deve ficar vazia.

Escreva um programa principal (`main.cpp`) com **um menu de opções** para que o usuário possa utilizar e testar TODAS as operações da estrutura `List` que você implementou.

Questão 2: [TAD Conjunto] Um **conjunto** é uma coleção de elementos. Todos os elementos de um conjunto são diferentes entre si e a ordem entre eles é irrelevante. Um exemplo de conjunto muito estudado é o conjunto dos números inteiros $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$. As operações básicas de um conjunto são:

- **União:** se A e B são conjuntos, então $A \cup B$ é o conjunto de elementos que são membros de A ou de B ou de ambos.
- **Intersecção:** se A e B são conjuntos, então $A \cap B$ é o conjunto de elementos que estão em A e em B.
- **Diferença:** se A e B são conjuntos, então $A - B$ é o conjunto de elementos de A que não estão em B.
- **Diferença Simétrica:** se A e B são conjuntos, então $A \ominus B$ é a união do conjunto de elementos de A que não estão em B com o conjunto dos elementos de B que não estão em A.

Estas operações são ilustradas nos diagramas de Venn da figura abaixo.



Problema: criar um tipo de dado Conjunto de Inteiros e disponibilizá-lo na forma de TAD (conjunto.c, conjunto.h e main.cpp).

As funções que devem ser implementadas são:

- `união`(A,B,C): recebe os conjuntos A e B como parâmetro e retorna o conjunto C = A $\cup$ B.
- `intersecção`(A,B,C): recebe os conjuntos A e B como parâmetro e retorna o conjunto C = A $\cap$ B.
- `diferença`(A,B,C): recebe os conjuntos A e B como parâmetro e retorna o conjunto C = A - B.
- `diferença simétrica`(A,B,C): recebe os conjuntos A e B como parâmetro e retorna o conjunto C = A $\ominus$ B.
- `membro`(y,A): recebe o conjunto A e um elemento y e retorna um 1 se $y \in$ A e 0 caso contrário.
- `criaConjVazio`(): cria um conjunto vazio e retorna o conjunto criado.
- `insere`(y,A): recebe o conjunto A e um elemento y e adiciona y ao conjunto A, isto é, A = A $\cup$ y.
- `remove`(y,A): recebe o conjunto A e um elemento y e remove y do conjunto A, isto é, A = A - y.
- `copia`(A,B): faz uma cópia do conjunto A em B.
- `min`(A): retorna o valor mínimo do conjunto A.
- `max`(A): retorna o valor máximo do conjunto A.
- `igual`(A,B): retorna `true` se os conjuntos A e B são iguais e `false` caso contrário.

Escreva um programa principal (`main.cpp`) com **um menu de opções** para que o usuário possa utilizar e testar TODAS as operações do TAD que você implementou.

Informações adicionais para este trabalho:

- Cada estrutura de dados deve ter um TAD que será utilizado pelo programa de aplicação;

- **Deverá ser submetido:**
  - Um **relatório do trabalho** realizado, contendo a especificação completa das estruturas de dados utilizadas;
  - Uma seção descrevendo como o trabalho foi dividido entre a dupla, se for o caso; além das dificuldades encontradas.
  - Os programas fonte devidamente organizados e documentados.

- Um dos parâmetros utilizados na avaliação da qualidade de uma implementação consiste na constatação da presença ou ausência de comentários. Comente o seu código. Mas também não comente por comentar, forneça bons comentários.

- Outro parâmetro de avaliação de código é a *portabilidade*. Dentre as diversas preocupações da portabilidade, existe a tentativa de codificar programas que sejam compiláveis em qualquer sistema operacional. Como testarei o seu código em uma máquina que roda Linux, não use bibliotecas que só existem para o sistema Windows como, por exemplo, a biblioteca `conio.h` e outras tantas.

- Este trabalho corresponde a **Avaliação Parcial 2** e vale 10 pontos.

- Não serão aceitos trabalhos submetidos após o prazo final.