FACULDADE DE COMPUTAÇÃO E INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
ESTRUTURA DE DADOS I – PROVA SUBSTITUTIVA – 2º SEMESTRE/2018
PROF. LUCIANO SILVA – 05/12/2018

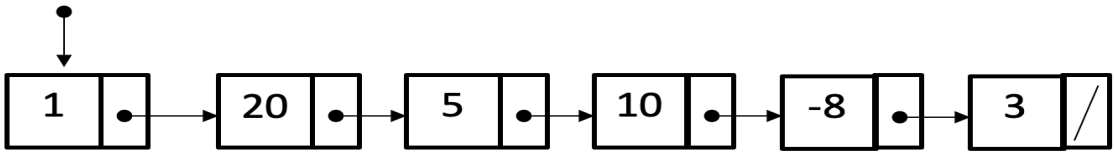
NOME: GABARITO

As implementações de referência para listas ligadas simples, duplamente encadeadas, pilhas e filhas, encontram-se no final desta prova.

1. (1.0 ponto) Considere o método abaixo, aplicado à estrutura de uma lista ligada simples:

```
template <typename E>
void ListaLigada<E>::enigma(){
   int cont=0;
   No<E> * p;
   for (p = cabeca; p != NULL && cont<3; p = p->prox,cont++);
   if (p!=NULL)
        std::cout << p-> elem << " " ;
}
```

(a) **(1.0 ponto)** Mostre qual será a saída (std::cout) do método enigma, quando aplicado à lista abaixo:



cont = 0 .... saída 1
cont = 1 .... saída 20
cont = 2 .... saída 5

(b) **(2.0 pontos)** Tendo como inspiração o método enigma, implemente o método **element(int n)**, que **exibe o n-ésimo elemento da lista, a partir da cabeça**. Por exemplo, 1 é o 0-ésimo elemento, 20 o 1-ésimo, 5 o 2-ésimo e assim por diante.

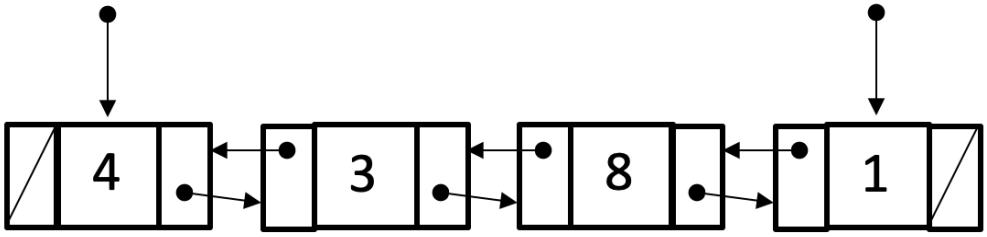
```
template <typename E>
void ListaLigada<E>::element(int n){

   int cont=0;
   No<E> * p;
   for (p = cabeca; p != NULL && cont<n+1; p = p->prox,cont++);
   if (p!=NULL)
        std::cout << p-> elem << " " ;
}
```

2. **(3.0 pontos)** Considere o método abaixo, aplicado à estrutura de uma lista duplamente ligada não-vazia:

```
template <typename E>
void ListaDuplamenteLigada<E>::enigma(){
   DNo<E> * p;
    DNo<E> * q;
   for (p = cabeca; p->prox!=NULL ; p=p->prox) ;
   for (q = fim; q->prev!=NULL ; q=q->prev) ;
   cout<< p->elem << " " << q->elem << std::endl;
}
```

(a) **(1.0 ponto)** Mostre qual será a saída (std::cout) do método enigma, quando aplicado à lista abaixo:



1 4

(c) **(2.0 pontos)** Tendo como inspiração o método enigma, implemente o método inverte, que troca os apontadores inicio e fim de uma lista duplamente ligada, ou seja, o início será o fim e, o fim, será o início.

```
template <typename E>
void ListaDuplamenteLigada<E>::inverte(){
        for ( ; cabeca->prox!=NULL ; cabeca=cabeca->prox) ;
        for ( ; fim->prev!=NULL ; fim=fim->prev) ;
}
```

3. **(2.0 pontos)** Em aula, conhecemos as notações prefixa, infixa e posfixa de uma expressão aritmética. Por exemplo, se considerarmos a expressão infixa 2+3*5, teremos as seguintes expressões prefixa e posfixa associadas:

   prefixa = + * 3 5 2

   posfixa = 2 5 3 * +

   Utilizando a pilha instanciada abaixo, construa um trecho de programa em C++ que **resolva a expressão prefixa** contida no vetor de caracteres expr (notação prefixa) para a notação posfixa.

   Sugestão: para converter caracteres para números inteiros, use a função atoi(char). Por exemplo, atoi('3')=3. Para fazer o procedimento ao contrário, use itoa. Por exemplo, itoa(3)='3';

```
char expr[5] = { '+','*', '3','5','2'};
Pilha<char> p;

// Passo 1 – Empilhar e desempilhar para transformar de prefixa para posfixa

for (int i=0;i<5;i++)
    p.empilha(expr[i]);
for (int i=0;i<5;i++){
    expr[i]=p.topo();
    p.desempilha();
}

// Aqui teremos expr[5] = { '2','5','3','*','+'}; Para resolver, vamos usar uma pilha de inteiros

Pilha<int> q;

q.empilha(atoi(expr[0])); // empilha 2 convertido para número
q.empilha(atoi(expr[1])); // empilha 5 convertido para número
q.empilha(atoi(expr[2])); // empilha 3 convertido para número
int a=q.topo(); // desempilha o 3
q.desempilha();
int b=q.topo(); // desempilha o 5
q.desempilha();
q.empilha(a*b); // empilha 5 3 * = 15
a=q.topo(); // desempilha 15
q.desempilha();
b=q.topo(); // desempilha 2
q.desempilha();
q.empilha(a+b); // empilha 2 15 +
std::cout<<q.topo() << std:: endl;   // mostra o resultado
```

4. **(2.0 pontos)** Queremos ordenar a sequencia de números mostrada abaixo em **ordem crescente**:

-1 -3 20 10 2

Utilizando uma **fila de prioridade**, implemente um trecho que código C++ para fazer esta ordenação em **ordem crescente**, utilizando a fila abaixo.

```
Fila<int> q;

q.enfilera (-1,2);   // segundo número na ordenação
q.enfilera (-3,1);   // primeiro número na ordenação
q.enfilera (20,5);   // quinto número na ordenação
q.enfilera (10,4);   // quarto número na ordenação
q.enfilera (2,3);   // primeiro número na ordenação

// Desenfilerar para mostrar a sequencia ordenada

for (int i=0;i<5;i++){
    std::cout << q.inicio() << std::endl;
    q.desenfilera();
}
```

CÓDIGOS DE REFERÊNCIA – LISTA LIGADA SIMPLES

```
template <typename E>
class No{
   private:
     E elem; // informação do no
     No<E>* prox; // próximo elemento
     friend class ListaLigada<E>;
};
```

```
template <typename E>
class ListaLigada{
   private:
     No<E> cabeca; // inicio da lista ligada (head)
   public:
     ListaLigada();
     ~ListaLigada();
     bool vazia() const;
     const E& inicio() const;
     void insereInicio(const E& e);
     void removeInicio();
};
```

CÓDIGOS DE REFERÊNCIA – LISTA DUPLAMENTE LIGADA

```
template <typename E>
class DNo{
   private:
     E elem; // informação do no
     DNo<E>* prox; // próximo elemento
     DNo<E>* prev; // elemento anterior
     friend class ListaDuplamenteLigada<E>;
};
```

```
template <typename E>
class ListaDuplamenteLigada{

   private:
     DNo<E> * cabeca; // inicio da lista ligada (head)
     DNo<E> * fim;    // final da lista ligada (trailer)

   public:
     ListaDuplamenteLigada();
     ~ListaDuplamenteLigada();
     bool vazia() const;
     const E& inicio() const;
     const E& final() const;
     void insereInicio(const E& e);
     void insereFinal(const E& e);
     void removeInicio();
     void removeFinal();
};
```

CÓDIGOS DE REFERÊNCIA – PILHA

POLÍTICA LIFO – O ÚLTIMO QUE ENTRA É O PRIMEIRO QUE SAI (LAST IN FIRST OUT)

```
#ifndef PILHA_H
#define PILHA_H

#include "lista_duplamente_ligada.cpp"

template <typename E>
class Pilha{

  private:
    ListaDuplamenteLigada<E>* p;

  public:
    Pilha();
    ~Pilha();
    bool vazia();
    void empilha(const E& e);
    const E& topo() const;
    void desempilha();
};

#endif
```

CÓDIGOS DE REFERÊNCIA – FILA

POLÍTICA FIFO – O PRIMEIRO QUE ENTRA É O PRIMEIRO QUE SAI (FIRST IN FIRST OUT)

```
#ifndef FILA_H
#define FILA _H

#include "lista_duplamente_ligada.cpp"

template <typename E>
class Fila{

  private:
    ListaDuplamenteLigada<E>* p;

  public:
    Fila();
    ~Fila();
    bool vazia();
    void enfilera(const E& e);
    void enfilera(const E& e,int prioridade);
    const E& inicio() const;
    void desenfilera();
};
#endif
```