



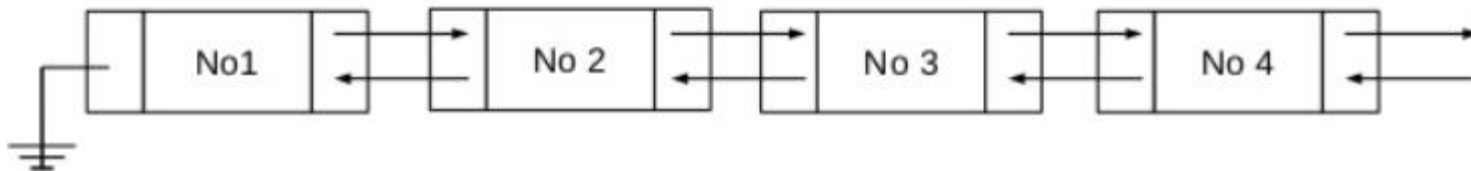
Lista Duplamente Encadeada

Prof. Valéria Farinazzo

E-mail: valfarinazzo@hotmail.com

Motivação

- Necessidade de:
 - Percorrer os elementos de trás para frente
 - A ideia é ter dois links em cada nó, apontando para direções opostas



Em uma lista duplamente encadeada é possível percorrer a lista em ambas as direções.

Também apresenta uma maior segurança do que uma lista simplesmente encadeada uma vez que, existe sempre dois ponteiros apontando para cada registro.



A célula modificada

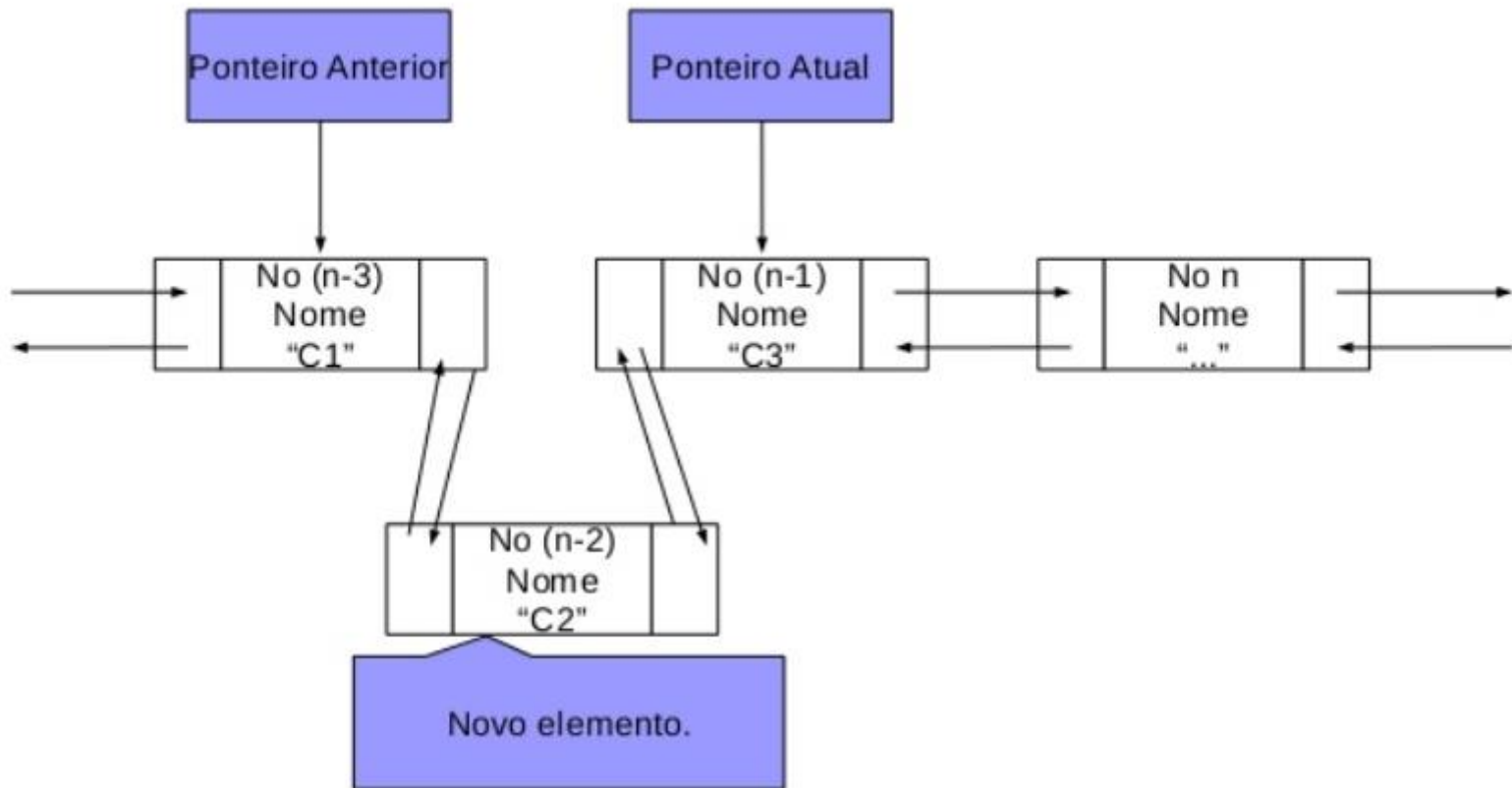
```
struct Node{  
    public:  
        int info;  
        Node *proximo,*anterior;  
        Node() {  
            proximo=anterior=NULL;  
        }  
        Node(int e1, Node *a=NULL,Node*p=NULL){  
            info=e1;proximo=p,anterior=a;  
        }  
};
```




Classe Lista Duplamente Encadeada

```
class ListaDuplamente{  
    private:  
        Node *primeiro, *ultimo;  
  
    public:  
        ListaDuplamente();  
        ~ListaDuplamente();  
        bool vazia();  
        bool insereinicio(int);  
        bool inserefinal(int);  
        bool insere(int);  
        bool removeprimeiro(int &);  
        bool removeultimo(int &);  
        bool remove(int);  
        int isInList(int);  
        void printList();  
        void inverseprintList();  
};
```

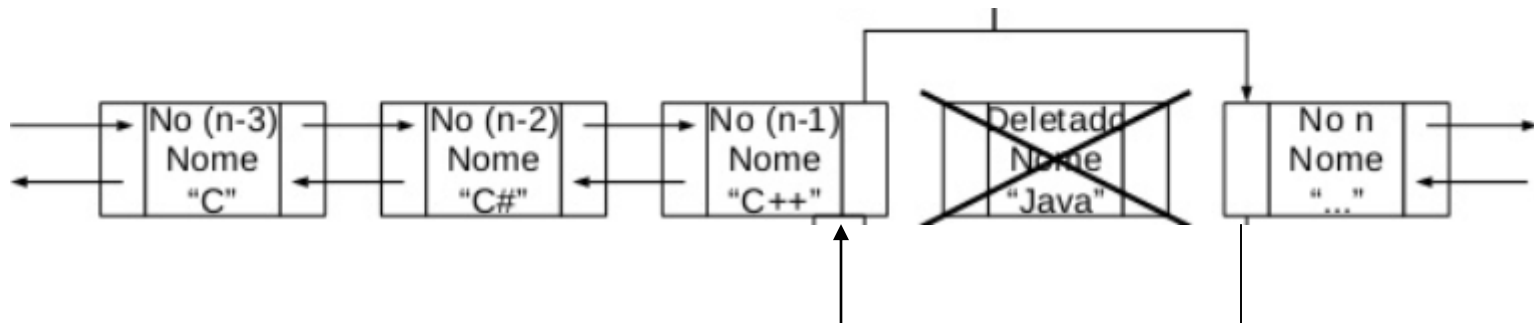
Inserção

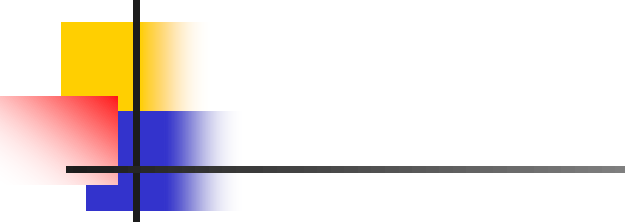




```
bool ListaDuplamente::insere(int el){
    Node *novo;
    novo = new Node(el,NULL,NULL);
    if (novo == NULL) return false;
    if(ultimo==NULL){//primeiro no
        ultimo = primeiro = novo;
    }
    else{
        Node *aux=primeiro;
        while(aux!=NULL && el>aux->info){
            aux = aux->proximo;
        }
        if(el<=primeiro->info){
            novo->proximo = primeiro;
            primeiro->anterior = novo;
            primeiro = novo;
        }
        else if(aux==NULL){//inserir depois do ultimo
            ultimo->proximo = novo;
            novo->anterior = ultimo;
            ultimo = novo;
        }
        else{//inserir no meio
            aux->anterior->proximo=novo;
            novo->anterior = aux->anterior;
            novo->proximo=aux;
            aux->anterior = novo;
        }
    }
    return true;
}
```

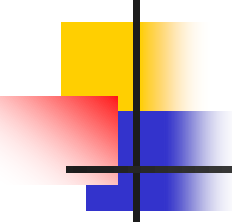
Remoção





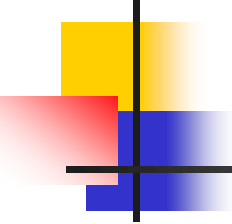
```
bool ListaDuplamente::remove(int el){
    if(vazia()) return false;
    if(primeiro==ultimo &&el==primeiro->info){
        delete primeiro;
        primeiro = ultimo = NULL;
    }
    else if(el==primeiro->info){
        Node *tmp=primeiro;
        primeiro = primeiro->proximo;
        primeiro->anterior = NULL;
        delete tmp;
    }
    else {
        Node *ant, *tmp;
        ant = primeiro;
        tmp = primeiro->proximo;
        while(tmp!=NULL && tmp->info !=el){
            ant = tmp;
            tmp = tmp->proximo;
        }
        if(tmp == NULL) return false;

        ant->proximo=tmp->proximo;
        if(tmp==ultimo){
            ultimo=ant;}
        else{
            tmp->proximo->anterior = ant;
        }
        delete tmp;
    }
    return true;
}
```

Percorrendo uma lista duplamente encadeada

```
Node *tmp;  
cout<<"\nLista: ";  
for(tmp=primeiro;tmp!=NULL;tmp=tmp->proximo)  
    cout<<tmp->info<<" ";
```



Percorrendo uma lista duplamente encadeada

```
Node *tmp = ultimo;  
    cout<<"\nLista inversa: ";  
    while(tmp!=NULL){  
        cout<<tmp->info<<" ";  
        tmp = tmp->anterior;  
    }
```



Exercícios

- Com base nos conceitos vistos em aula, faça todos os métodos declarados na classe ListaDuplamente.
- Documente todos os métodos