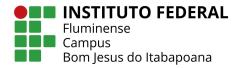


Material complementar Pilha, Lista e Fila

```
// Estrutura do nó
struct No {
  int data; // Dado armazenado no nó
  No* prox; // Ponteiro para o próximo nó
};
// Função para inserir no início
void insereInicio(No*& inicio, int valor) {
  No* novoNo = new No; // Cria um novo nó
  novoNo->data = valor; // Atribui o valor ao nó
  novoNo->prox = inicio; // O novo nó aponta para o antigo primeiro nó
  inicio = novoNo; // Atualiza o início da lista
}
// Função para inserir no final
void insereFinal(No*& inicio, int valor) {
  No* novoNo = new No; // Cria um novo nó
  novoNo->data = valor; // Atribui o valor ao nó
  novoNo->prox = nullptr; // O novo nó será o último, então aponta para nullptr
  if (inicio == nullptr) { // Se a lista estiver vazia
     inicio = novoNo; // O novo nó será o primeiro
     return;
  }
  No* atual = inicio; // Ponteiro auxiliar para percorrer a lista
  while (atual->prox != nullptr) { // Percorre até o último nó
     atual = atual->prox;
  }
  atual->prox = novoNo; // O último nó agora aponta para o novo nó
}
```



```
// Função para inserir de forma ordenada
void insereOrdenado(No*& inicio, int valor) {
  No* novoNo = new No; // Cria um novo nó
  novoNo->data = valor; // Atribui o valor ao nó
  novoNo->prox = nullptr;
  if (inicio == nullptr || inicio->data >= valor) { // Se a lista estiver vazia ou o valor for menor
que o primeiro elemento
     novoNo->prox = inicio; // O novo nó aponta para o antigo primeiro nó
     inicio = novoNo; // Atualiza o início da lista
     return;
  }
  No* atual = inicio; // Ponteiro auxiliar para percorrer a lista
  while (atual->prox != nullptr && atual->prox->data < valor) { // Encontra a posição correta
     atual = atual->prox;
  }
  novoNo->prox = atual->prox; // O novo nó aponta para o próximo nó
  atual->prox = novoNo; // O nó anterior aponta para o novo nó
}
// Função para remover do início
void removerInicio(No*& inicio) {
  if (inicio == nullptr) return; // Se a lista estiver vazia, não faz nada
  No* temp = inicio; // Salva o ponteiro do primeiro nó
  inicio = inicio->prox; // Atualiza o início para o próximo nó
  delete temp; // Libera a memória do nó removido
}
// Função para remover do final
void removerFinal(No*& inicio) {
  if (inicio == nullptr) return; // Se a lista estiver vazia, não faz nada
  if (inicio->prox == nullptr) { // Se houver apenas um nó na lista
     delete inicio; // Libera a memória
     inicio = nullptr; // Define o início como vazio
     return;
  }
  No* atual = inicio; // Ponteiro auxiliar para percorrer a lista
  No* anterior = nullptr; // Ponteiro para armazenar o nó anterior
  while (atual->prox != nullptr) { // Percorre até o último nó
     anterior = atual;
     atual = atual->prox;
  }
  anterior->prox = nullptr; // O novo último nó aponta para nullptr
```



```
delete atual; // Libera a memória do nó removido
}
// Função para remover um valor específico
void excluir(No*& inicio, int valor) {
  No* atual = inicio; // Ponteiro auxiliar para percorrer a lista
  No* anterior = nullptr; // Ponteiro para armazenar o nó anterior
  while (atual != nullptr && atual->data != valor) { // Percorre até encontrar o valor
     anterior = atual;
     atual = atual->prox;
  }
  if (atual == nullptr) return; // Se não encontrar o valor, sai da função
  if (anterior == nullptr) { // Se o nó a ser removido for o primeiro
     inicio = inicio->prox; // Atualiza o início da lista
  } else {
     anterior->prox = atual->prox; // O nó anterior aponta para o próximo nó
  }
  delete atual; // Libera a memória do nó removido
}
// Função para imprimir a lista
void imprimir(No* inicio) {
  No* atual = inicio; // Ponteiro auxiliar para percorrer a lista
  while (atual != nullptr) { // Percorre toda a lista
     cout << atual->data << " -> "; // Imprime o valor do nó
     atual = atual->prox; // Avança para o próximo nó
  cout << "nullptr" << endl; // Indica o fim da lista
```