Listas, filas e pilhas em C

EDUARDO HABIB BECHELANE MAIA

HABIB@CEFETMG.BR

CEFET-MG DIVINÓPOLIS

ENGENHARIA DE COMPUTAÇÃO

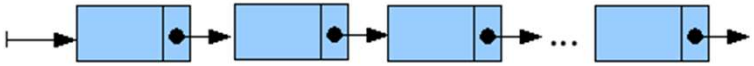
Listas simplesmente encadeada

Listas lineares

1. É uma das formas mais simples de interligar elementos
2. Estrutura em que as operações inserir, retirar e localizar são definidas.
3. Podem crescer ou diminuir de tamanho durante a execução de um programa, de acordo com a demanda.
4. Itens podem ser acessados, inseridos ou retirados de uma lista.
5. Duas listas podem ser concatenadas para formar uma lista única, ou uma pode ser partida em duas ou mais listas.
6. Adequadas quando não é possível prever a demanda por memória, permitindo a manipulação de quantidades imprevisíveis de dados, de formato também imprevisível.

Listas Lineares

1. Diferente do que ocorre em vetores, nas listas lineares os elementos não estão necessariamente armazenados sequencialmente na memória

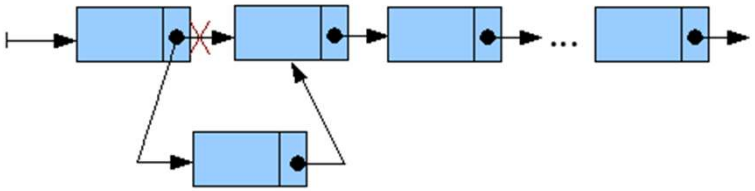


2. Exemplo de lista duplamente encadeada

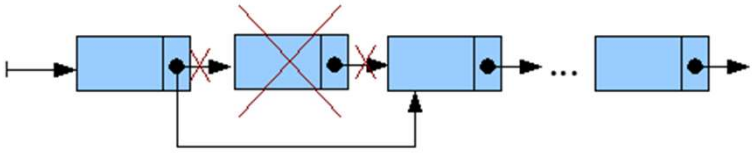
IMAGENS RETIRADAS DE HTTPS://WWW.COS.UFRJ.BR/~RFARIAS/COS121/AULA_11.HTML

Inserção e remoção

1. Inserção em lista:



2. Remoção



IMAGENS RETIRADAS DE HTTPS://WWW.COS.UFRJ.BR/~RFARIAS/COS121/AULA_11.HTML

Exemplo lista.h

```
#ifndef LISTA_H
#define LISTA_H

struct Nodo {
   int info;
   struct Nodo *prox;
};

struct ListaSimplesEnc {
   struct Nodo *prim;
};

void criarLista(struct ListaSimplesEnc *pList);
void mostrarLista(struct ListaSimplesEnc *pList);
void inserirIni(struct ListaSimplesEnc *pList, int v);
void removerIni(struct ListaSimplesEnc *pList);
int estaVazia(struct ListaSimplesEnc *pList);

#endif
```

Exemplo lista.c

```
#include <stdio.h>
#include <stdlib.h>
#include "lista.h"

void criarLista(struct ListaSimplesEnc *pList) {
    pList->prim = NULL;
}

void mostrarLista(struct ListaSimplesEnc *pList) {
    struct Nodo *p;

    for (p = pList->prim; p != NULL; p = p->prox) {
        printf("%d\t", p->info);
    }

    printf("\n");
}

void inserirIni(struct ListaSimplesEnc *pList, int v) {
    struct Nodo *novo;
    novo = (struct Nodo*)malloc(sizeof(struct Nodo));
    novo->info = v;
    novo->prox = pList->prim;
    pList->prim = novo;
}

void removerIni(struct ListaSimplesEnc *pList) {
    struct Nodo *pAux = pList->prim;
    pList->prim = pList->prim->prox;
    free(pAux);
}

int estaVazia(struct ListaSimplesEnc *pList) {
    return (pList->prim == NULL);
}
```

CEFET-MG DIVINÓPOLIS

ENGENHARIA DE COMPUTAÇÃO

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "lista.h"

int main() {
    struct ListaSimplesEnc minhaLista;
    int valor, op;

    criarLista(&minhaLista);

    while (1) {
        printf("1 - Inserir elemento no inicio\n");
        printf("2 - Inserir elemento em ordem (so se a lista estiver ordenada)\n");
        printf("3 - Remover elemento no inicio\n");
        printf("4 - Remover elemento\n");
        printf("5 - Mostrar lista\n");
        printf("6 - Sair\n");
        printf("Opcao? ");
        scanf("%d", &op);

        switch (op) {
            case 1: // inserir elemento no inicio
                printf("Valor? ");
                scanf("%d", &valor);
                inserirIni(&minhaLista, valor);
                break;
            case 2: // inserir elemento ordenado
                printf("Valor? ");
                scanf("%d", &valor);
                // TODO
                break;
            case 3: // remover o primeiro
                // TODO
                break;
            case 4: // remover determinado elemento
                // TODO
                break;
            case 5: // mostrar lista
                if (estaVazia(&minhaLista)) {
                    printf("Lista vazia\n");
                } else {
                    mostrarLista(&minhaLista);
                }
                break;
            case 6: // abandonar o programa
                exit(0);
        }
    }

    return 0;
}
```

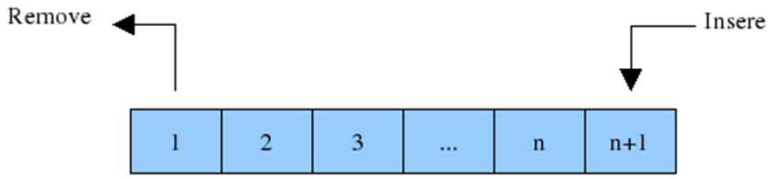
CEFET-MG DIVINÓPOLIS

ENGENHARIA DE COMPUTAÇÃO

Filas

Filas

1. São estruturas de dados do tipo FIFO (first-in first-out)
   a. Primeiro a entrar, primeiro a sair

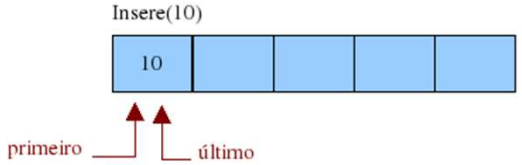


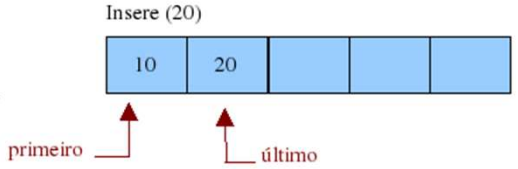
2. Exemplos:
   a. Controle de impressão
   b. Fila de pessoas
   c. etc

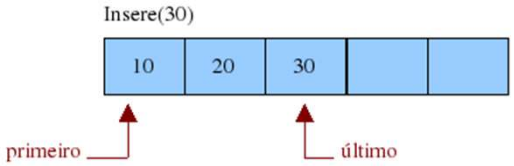
IMAGENS RETIRADAS DE HTTPS://WWW.COS.UFRJ.BR/~RFARIAS/COS121/FILAS.HTML

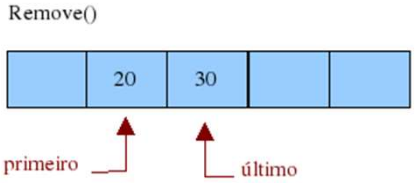
Operações

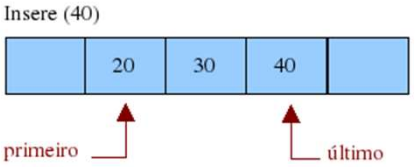
1. criação da fila;
2. enfileirar;
3. desenfileirar;
4. mostrar a fila;
5. verificar se a fila está vazia;

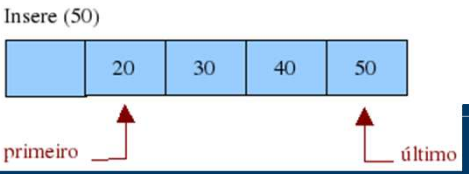

Insere(10)
10
primeiro
último


Insere (20)
10
20
primeiro
último


Insere(30)
10
20
30
primeiro
último


Remove()
20
30
primeiro
último


Insere (40)
20
30
40
primeiro
último


Insere (50)
20
30
40
50
primeiro
último

fila.h

```
#ifndef FILA_H
#define FILA_H

// Define o tipo de dados que será armazenado na fila
typedef int TipoDado;

// Define o nó da fila
typedef struct no {
   TipoDado dado; // O dado do nó
   struct no *prox; // O ponteiro para o próximo nó
} No;

// Define a fila
typedef struct fila {
   No *inicio; // O ponteiro para o início da fila
   No *fim; // O ponteiro para o fim da fila
} Fila;

// Cria uma fila vazia e retorna seu endereço
Fila* criar_fila();

// Insere um dado no fim da fila
void enfileirar(Fila *f, TipoDado dado);

// Remove um dado do início da fila e retorna seu valor
TipoDado desenfileirar(Fila *f);

// Imprime os dados da fila na ordem do início ao fim
void mostrar_fila(Fila *f);

// Verifica se a fila está vazia e retorna 1 se sim, 0 se não
int fila_vazia(Fila *f);

// Libera a memória de uma fila e seus nós
void liberar_fila(Fila *f);

#endif
```

CEFET-MG DIVINÓPOLIS

ENGENHARIA DE COMPUTAÇÃO

Fila.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "fila.h"

// Cria uma fila vazia e retorna seu endereço
Fila* criar_fila() {
    Fila *f = (Fila*) malloc(sizeof(Fila)); // Aloca memória para a fila
    if (f == NULL) { // Verifica se a alocação foi bem sucedida
        printf("Erro na alocação de memória.\n");
        exit(1);
    }
    f->inicio = NULL; // Inicializa o início da fila como NULL
    f->fim = NULL; // Inicializa o fim da fila como NULL
    return f; // Retorna o endereço da fila criada
}

// Insere um dado no fim da fila
void enfileirar(Fila *f, TipoDado dado) {
    if (f != NULL) { // Verifica se a fila existe
        No *novo = (No*) malloc(sizeof(No)); // Aloca memória para o novo nó
        if (novo == NULL) { // Verifica se a alocação foi bem sucedida
            printf("Erro na alocação de memória.\n");
            exit(1);
        }
        novo->dado = dado; // Atribui o dado ao novo nó
        novo->prox = NULL; // Atribui NULL ao ponteiro para o próximo nó do novo nó

        if (fila_vazia(f)) { // Se a fila estiver vazia, o novo nó será o início e o fim da fila
            f->inicio = novo;
            f->fim = novo;
        } else { // Se a fila não estiver vazia, o novo nó será o próximo do fim da fila e o novo fim da fila
            f->fim->prox = novo;
            f->fim = novo;
        }
    }
}

// Remove um dado do início da fila e retorna seu valor
TipoDado desenfileirar(Fila *f) {
    if (f != NULL && !fila_vazia(f)) { // Verifica se a fila existe e não está vazia
        No *aux = f->inicio; // Guarda o endereço do início da fila em aux
        TipoDado dado = aux->dado; // Guarda o dado do início da fila em dado
        f->inicio = f->inicio->prox; // Atualiza o início da fila para o próximo nó
        free(aux); // Libera a memória do nó apontado por aux
        if (fila_vazia(f)) { // Se a fila ficou vazia, o fim da fila também deve ser NULL
            f->fim = NULL;
        }
        return dado; // Retorna o dado removido
    } else { // Se a fila não existe ou está vazia, retorna um valor inválido
        printf("Fila inexistente ou vazia.\n");
        return -1;
    }
}

// Imprime os dados da fila na ordem do início ao fim
void mostrar_fila(Fila *f) {
    if (f != NULL && !fila_vazia(f)) { // Verifica se a fila existe e não está vazia
        No *aux = f->inicio; // Cria um ponteiro auxiliar para percorrer a fila
        printf("Fila: ");
        while (aux != NULL) { // Enquanto o auxiliar não for NULL
            printf("%d ", aux->dado); // Imprime o dado do nó apontado por aux
            aux = aux->prox; // Atualiza o auxiliar para o próximo nó
        }
        printf("\n");
    } else { // Se a fila não existe ou está vazia, imprime uma mensagem de erro
        printf("Fila inexistente ou vazia.\n");
    }
}

// Verifica se a fila está vazia e retorna 1 se sim, 0 se não
int fila_vazia(Fila *f) {
    if (f != NULL) { // Verifica se a fila existe
        return f->inicio == NULL; // Retorna 1 se o início da fila for NULL, 0 caso contrário
    } else { // Se a fila não existe, retorna -1
        return -1;
    }
}

// Libera a memória de uma fila e seus nós
void liberar_fila(Fila *f) {
    if (f != NULL) { // Verifica se a fila existe
        No *aux; // Cria um ponteiro auxiliar para percorrer a fila
        while (f->inicio != NULL) { // Enquanto o início da fila não for NULL
            aux = f->inicio; // Guarda o endereço do início da fila em aux
            f->inicio = f->inicio->prox; // Atualiza o início da fila para o próximo nó
            free(aux); // Libera a memória do nó apontado por aux
        }
        free(f); // Libera a memória da fila
    }
}
```

CEFET-MG DIVINÓPOLIS

ENGENHARIA DE COMPUTAÇÃO

Main.c

```c
#include <stdio.h>
#include "fila.h"

int main() {
    Fila *f = criar_fila(); // Cria uma fila vazia
    int opcao, dado; // Variáveis para armazenar a opção do menu e o dado a ser inserido
ou removido

    do {
        printf("Escolha uma opção:\n");
        printf("1 - Inserir na fila\n");
        printf("2 - Remover da fila\n");
        printf("3 - Imprimir a fila\n");
        printf("0 - Sair\n");
        scanf("%d", &opcao);

        switch (opcao) {
            case 1: // Inserir na fila
                printf("Digite o dado a ser inserido: ");
                scanf("%d", &dado);
                enfileirar(f, dado);
                break;
            case 2: // Remover da fila
                dado = desenfileirar(f);
                if (dado != -1) {
                    printf("Dado removido: %d\n", dado);
                }
                break;
            case 3: // Imprimir a fila
                mostrar_fila(f);
                break;
            case 0: // Sair
                break;
            default: // Opção inválida
                printf("Opção inválida.\n");
                break;
        }
    } while (opcao != 0);

    liberar_fila(f); // Libera a memória da fila
```

CEFET-MG DIVINÓPOLIS

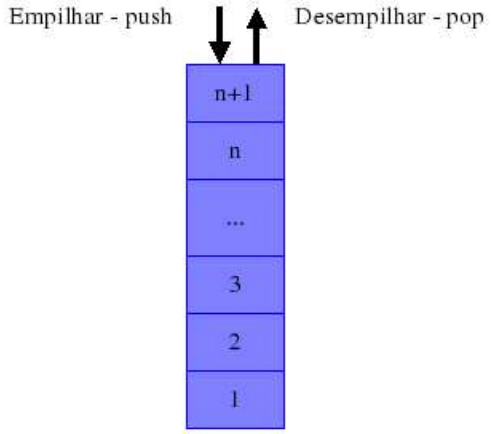
ENGENHARIA DE COMPUTAÇÃO

Pilhas

Pilhas

São estruturas de dados do tipo LIFO (last-in first-out),

- último elemento inserido é o primeiro a ser retirado.
- Uma pilha permite acesso a apenas um item de dados
  - O último.
  - Para retirar o penúltimo, deve-se remover o último.



Exemplos de pilha

- Funções recursivas em compiladores;
- Mecanismo de desfazer/refazer dos editores de texto;
- Navegação entre páginas Web;
- Pilha de livros
- Pilha de pratos
- etc

Implementação

Pode ser realizada através de vetor

Através de listas encadeadas.

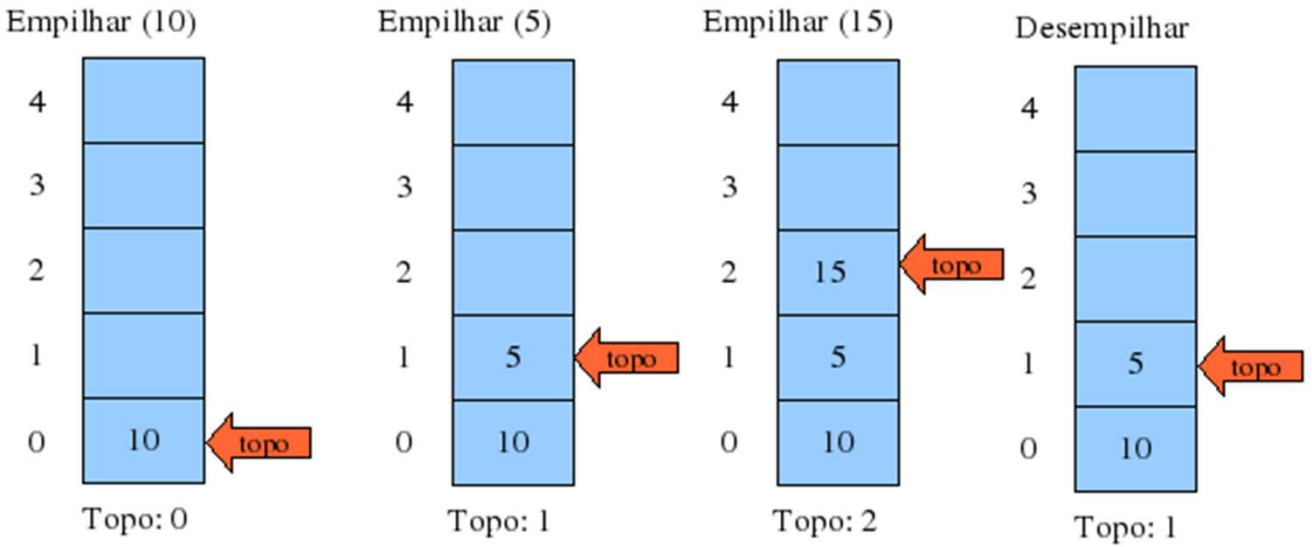
A manipulação dos elementos é realizada em apenas no topo

A outra extremidade é chamada de base.

Operações

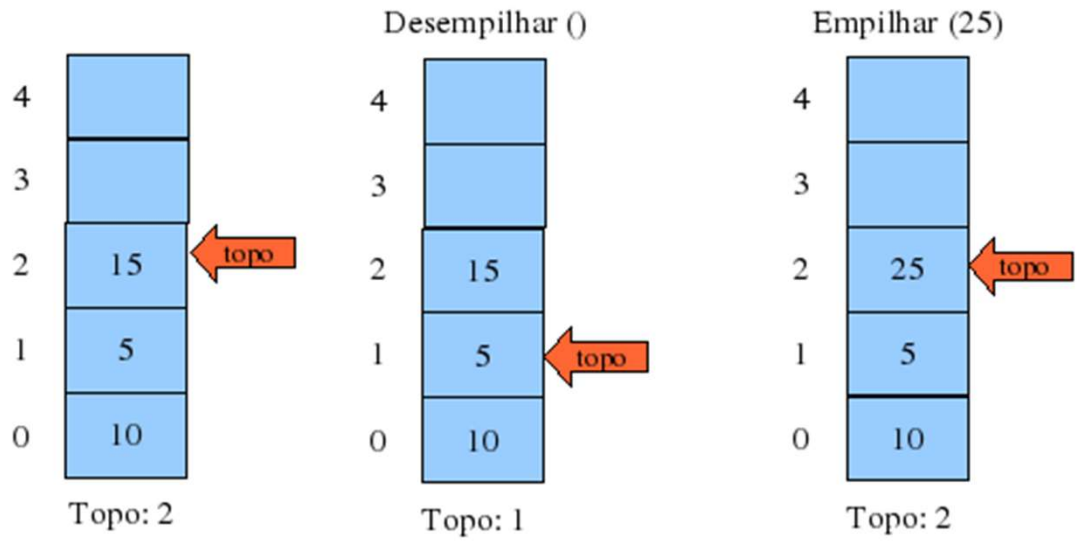
- criação da pilha;
- empilhar (push) - o elemento é o parâmetro nesta operação;
- desempilhar (pop);
- mostrar o topo;
- verificar se a pilha está vazia;
- verificar se a pilha está cheia

Exemplo


Empilhar (10)
4
3
2
1
0
10
topo
Topo: 0
Empilhar (5)
4
3
2
1
0
5
10
topo
Topo: 1
Empilhar (15)
4
3
2
1
0
15
5
10
topo
Topo: 2
Desempilhar
4
3
2
1
0
5
10
topo
Topo: 1

Exemplo

Em vetores, a remoção de um elemento da pilha é realizada apenas alterando-se a informação da posição do topo .



IMAGENS RETIRADAS DE: HTTPS://WWW.COS.UFRJ.BR/~RFARIAS/COS121/PILHAS.HTML

pilha.h

```
#ifndef PILHA_H
#define PILHA_H

struct Pilha {
        int topo; /* posição elemento topo */
        int capa;
        float *pElem;
};

void criarpilha(struct Pilha *p, int c);
int estavazia(struct Pilha *p);
int estacheia(struct Pilha *p);
void empilhar(struct Pilha *p, float v);
float desempilhar(struct Pilha *p);
float retornatopo(struct Pilha *p);

#endif  // PILHA_H
```

pilha.c

```
#include <stdlib.h>
#include “pilha.h"

void criarpilha(struct Pilha *p, int c) {
        p->topo = -1;
        p->capa = c;
        p->pElem = (float*) malloc(c * sizeof(float));
}

int estavazia(struct Pilha *p) {
        if (p->topo == -1)
                return 1;   // true
        else
                return 0;   // false
}

int estacheia(struct Pilha *p) {
        if (p->topo == p->capa - 1)
                return 1;
        else
                return 0;
}

void empilhar(struct Pilha *p, float v) {
        p->topo++;
        p->pElem[p->topo] = v;
}

float desempilhar(struct Pilha *p) {
        float aux = p->pElem[p->topo];
        p->topo--;
        return aux;
}

float retornatopo(struct Pilha *p) {
        return p->pElem[p->topo];
}
```

CEFET-MG DIVINÓPOLIS

ENGENHARIA DE COMPUTAÇÃO

```c
#include “pilha.h"
#include <stdio.h>
#include <stdlib.h>

int main() {
 struct Pilha minhapilha;
 int capacidade, op;
 float valor;

 printf("\nCapacidade da pilha? ");
 scanf("%d", &capacidade);

 criarpilha(&minhapilha, capacidade);

 while (1) { /* loop infinito */
  printf("\n1- empilhar (push)\n");
  printf("2- desempilhar (POP)\n");
  printf("3- Mostrar o topo \n");
  printf("4- sair\n");
  printf("\nopcao? ");
  scanf("%d", &op);

  switch (op) {
  case 1: // push

   if (estacheia(&minhapilha) == 1)

    printf("\nPILHA CHEIA! \n");

   else {
    printf("\nVALOR? ");
    scanf("%f", &valor);
    empilhar(&minhapilha, valor);
   }
    break;

  case 2: // pop
   if (estavazia(&minhapilha) == 1)

    printf("\nPILHA VAZIA! \n");

   else {
    valor = desempilhar(&minhapilha);
    printf("\n%.1f DESEMPILHADO!\n", valor);
   }
   break;

  case 3: // mostrar o topo
   if (estavazia(&minhapilha) == 1)

    printf("\nPILHA VAZIA!\n");

   else {
    valor = retornatopo(&minhapilha);
    printf("\nTOPO: %.1f\n", valor);
   }
   break;

  case 4:
   exit(0);

  default:
   printf("\nOPCAO INVALIDA! \n");
  }
 }

 return 0;
}
```

CEFET-MG DIVINÓPOLIS

ENGENHARIA DE COMPUTAÇÃO

Referências

https://www.cos.ufrj.br/~rfarias/cos121/aula_11.html

https://www.cos.ufrj.br/~rfarias/cos121/filas.html