

# Estruturas Compostas

## Parte II

### Filas e Pilhas

Leonardo Tórtoro Pereira

Slides fortemente baseados no material do professor Ricardo Farias:  
<http://www.cos.ufrj.br/~rfarias/cos121/>

Pilhas

# Pilhas

- Também conhecida como *stack*
- Estrutura de dados nas quais o último elemento inserido é o primeiro a ser retirado
- ◆ Só é permitido acesso a 1 único item por vez
  - O último que foi inserido

# Pilhas

- Por seu tipo de acesso, é uma estrutura LIFO
  - ◆ Last-In First-Out

# Pilhas

- Pilha é construída a partir da base até o topo
- ◆ Topo = nome dado ao último elemento inserido
- Qualquer operação com a pilha ocorre no topo

# Pilhas

- Duas operações essenciais ocorrem numa pilha
  - ◆ Inserção, também conhecido como *push*
  - ◆ Remoção, também conhecido como *pop*

# Pilhas

→ *Push*

- ◆ Insere um elemento no topo da pilha
- ◆ Aumenta o tamanho da pilha
- ◆ Atualiza qual elemento é o topo

# Pilhas

→ *Pop*

- ◆ Remove o elemento no topo da pilha
- ◆ Reduz o tamanho da pilha
- ◆ Atualiza qual elemento é o topo

# Pilhas

- Existem duas maneiras de implementar uma pilha
  - ◆ Vetores
  - ◆ Lista Ligada
- Vamos ver agora a implementação em Vetor

# Pilhas

- Vamos ver um exemplo!
- <https://www.cs.usfca.edu/~galles/visualization/StackArray.html>

# Implementação de Pilha com Vetores

# Struct Pilha

```
struct Pilha {  
  
    int topo; //Posição do elemento topo  
    int capacidade;  
    float *pElem; //Elemento da pilha. No caso, float  
  
};
```

# Criar Pilha

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

# Push - Empilhar

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

# Pop - Desempilhar

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

# Obter elemento do topo

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

# Checar se está vazia

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

# Checar se está cheia

```
int isFull( struct Pilha *p ){  
  
    if(p->topo == p->capacidade - 1)  
        return 1;  
  
    else  
        return 0;  
  
}
```

# Testando!

```
int main(){

    struct Pilha minhapilha;
    int capacidade, op;
    float valor;

    printf( "\nCapacidade da pilha? " );
    scanf( "%d", &capacidade );
    criarpilha (&minhapilha, capacidade);
    while( 1 ){ /* loop infinito */
```

# Testando!

```
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);
```

# Testando!

```
switch (op){
    case 1: //push
        if( isFull( &minhapilha ) == 1 )
            printf("\nPILHA CHEIA! \n");
        else {
            printf("\nVALOR? ");
            scanf("%f", &valor);
            push(&minhapilha, valor);
        }
        break;
```

# Testando!

```
case 2: //pop
    if ( isEmpty(&minhapilha) == 1 )
        printf( "\nPILHA VAZIA! \n" );
    else{
        valor = pop(&minhapilha);
        printf ( "\n%.1f DESEMPILHADO!\n", valor );
    }
    break;
```

# Testando!

```
case 3: // mostrar o topo
    if ( isEmpty(&minhapilha) == 1 )
        printf( "\nPILHA VAZIA!\n" );
    else {
        valor = retornatopo (&minhapilha);
        printf ( "\nTOPO: %.1f\n", valor );
    }
    break;
```

# Testando!

```
case 4:  
    exit(0);  
default: printf( "\nOPCAO INVALIDA! \n" );  
}  
}  
}
```

Filas

# Filas

- Também conhecida como *queue*
- Estrutura de dados nas quais o elemento inserido há mais tempo é o primeiro a ser retirado
- ◆ Só é permitido acesso a 1 único item por vez
  - O mais antigo

# Filas

- Por seu tipo de acesso, é uma estrutura FIFO
  - ◆ First-In First-Out

# Filas

- Adiciona-se elementos no fim da fila (*tail*)
- Remove-se elementos no início dela (*head*)

# Filas

- Duas operações essenciais ocorrem numa fila
  - ◆ Inserção, também conhecido como *enqueue*
  - ◆ Remoção, também conhecido como *dequeue*

# Filas

→ *Enqueue*

- ◆ Insere um elemento no fim da fila
- ◆ Aumenta o tamanho da pilha
- ◆ Atualiza qual elemento é o último

# Filas

→ *Dequeue*

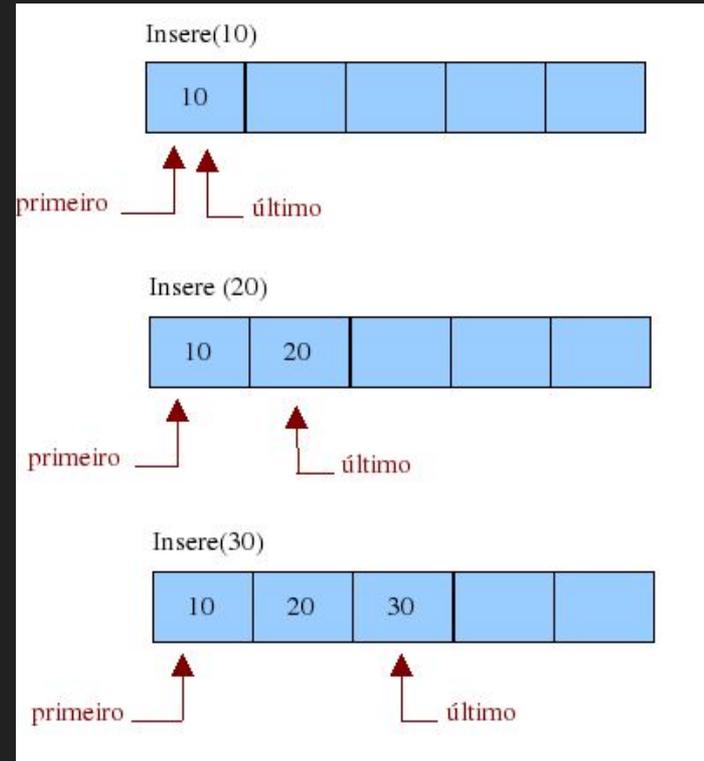
- ◆ Remove o elemento no início da fila
- ◆ Reduz o tamanho da fila
- ◆ Atualiza qual elemento é o início

# Filas

- Existem duas maneiras de implementar uma fila
  - ◆ Vetores
  - ◆ Lista Ligada
- Vamos ver agora a implementação em Vetor

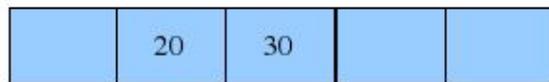
# Filas

- Vamos ver um exemplo!
- <https://www.cs.usfca.edu/~galles/visualization/QueueArray.html>



<http://www.cos.ufrj.br/~rfarias/cos121/filas.html>

Remove()



primeiro  último 

Inser (40)



primeiro  último 

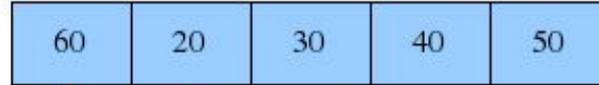
Inser (50)



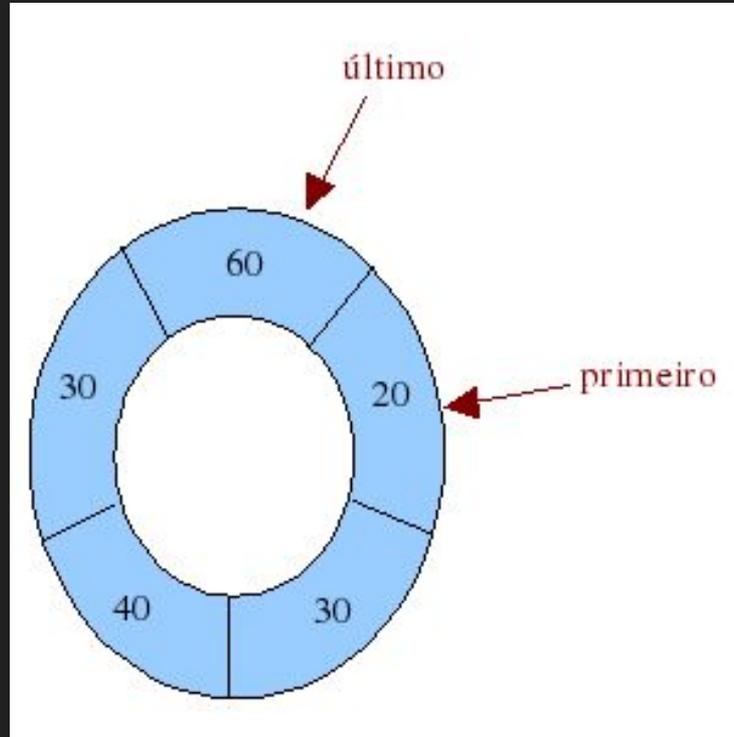
primeiro  último 

E se inserirmos mais um  
elemento?

Inser (60)



último ——— primeiro



<http://www.cos.ufrj.br/~rfarias/cos121/filas.html>

# Implementação de Pilha com Vetores

# Struct Fila

```
struct Fila {  
  
    int capacidade;  
    float *dados;  
    int primeiro;  
    int ultimo;  
    int nItens;  
  
};
```

# Criar Fila

```
void criarFila( struct Fila *f, int c ) {  
  
    f->capacidade = c;  
    f->dados = (float*) malloc (f->capacidade *  
sizeof(float));  
    f->primeiro = 0;  
    f->ultimo = -1;  
    f->nItens = 0;  
  
}
```

# Inserir - Enqueue

```
void enqueue(struct Fila *f, int v) {  
  
    if(f->ultimo == f->capacidade-1)  
        f->ultimo = -1;  
  
    f->ultimo++;  
    f->dados[f->ultimo] = v; // incrementa ultimo e insere  
    f->nItens++; // mais um item inserido  
  
}
```

# Remover - Dequeue

```
int dequeue( struct Fila *f ) { // pega o item do começo da
fila

    int temp = f->dados[f->primeiro++]; // pega o valor e
incrementa o primeiro
    if(f->primeiro == f->capacidade)
        f->primeiro = 0;

    f->nItens--; // um item retirado
    return temp;
}
```

# Está Vazia ou Está Cheia?

```
int isEmpty( struct Fila *f ) { // True se estiver vazia

    return (f->nItens==0);

}

int isFull( struct Fila *f ) { // True se estiver cheia

    return (f->nItens == f->capacidade);

}
```

# Mostrar a Fila

```
void mostrarFila(struct Fila *f){
    int cont, i;
    for ( cont=0, i= f->primeiro; cont < f->nItens; cont++){

        printf("%.2f\t",f->dados[i++]);
        if (i == f->capacidade)
            i=0;

    }
    printf("\n\n");
}
```

Testando!

# Testando

```
void main () {  
  
    int opcao, capa;  
    float valor;  
    struct Fila umaFila;  
  
    // cria a fila  
    printf("\nCapacidade da fila? ");  
    scanf("%d",&capa);  
    criarFila(&umaFila, capa);  
}
```

# Testando

```
// apresenta menu
while( 1 ){

    printf("\n1 - Inserir elemento\n2 - Remover
           elemento\n3 - Mostrar Fila\n0 - Sair\n\nOpcao? ");

    scanf("%d", &opcao);

    switch(opcao) {
        case 0: exit(0);
```

# Testando

```
case 1: // insere elemento
    if (estaCheia(&umaFila)) {
        printf("\nFila Cheia!!!\n\n");
    }
    else {
        printf("\nValor do elemento a ser inserido");
        scanf("%f", &valor);
        inserir(&umaFila, valor);
    }
    break;
```

# Testando

```
case 2: // remove elemento
    if (estaVazia(&umaFila)) {
        printf("\nFila vazia!!!\n\n");
    }
    else {
        valor = remover(&umaFila);
        printf("\n%1f removido com sucesso\n", valor);
    }
    break;
```

# Testando

```
case 3: // mostrar fila
    if (estaVazia(&umaFila)) {
        printf("\nFila vazia!!!\n\n");
    }
    else {
        printf("\nConteudo da fila => ");
        mostrarFila(&umaFila);
    }
    break;
```

# Testando

```
default:  
    printf("\nOpcao Invalida\n\n");  
}  
}  
}
```

# Referências

# Referências

1. <https://www.cs.usfca.edu/~galles/visualization/Algorithms.html>
2. <http://www.cos.ufrj.br/~rfarias/cos121/pilhas.html>
3. <https://www.vivaolinux.com.br/script/Pilha-usando-lista-encadeada>
4. <http://www.cos.ufrj.br/~rfarias/cos121/filas.html>