

Programação Procedimental

Filas e Pilhas

Aula 11

Prof. Felipe A. Louza



Roteiro

- 1 Filas
- 2 TAD Filas
- 3 Outras alternativas
- 4 Exemplos de aplicações
- 5 Aplicação de Filas
- 6 Pilhas
- 7 TAD Pilhas
- 8 Outra alternativa
- 9 Exemplos de aplicações
- 10 Aplicação de Pilhas
- 11 Referências

Filas

- Uma impressora é **compartilhada** em um laboratório
- Alunos enviam documentos **quase** ao mesmo tempo



Como gerenciar a lista de tarefas de impressão?


Fila

Fila:

- Remove primeiro objetos **inseridos há mais tempo**
- **FIFO** (first-in first-out): primeiro a entrar é primeiro a sair

Operações:

- **Enfileira** (queue): adiciona item no “fim”
- **Desenfileira** (dequeue): remove item do “início”

Exemplo: **Enfileira**()



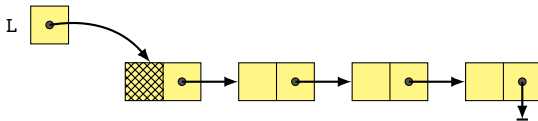
Roteiro

- 1 Filas
- 2 TAD Filas**
- 3 Outras alternativas
- 4 Exemplos de aplicações
- 5 Aplicação de Filas
- 6 Pilhas
- 7 TAD Pilhas
- 8 Outra alternativa
- 9 Exemplos de aplicações
- 10 Aplicação de Pilhas
- 11 Referências

TAD Filas

TAD Filas:

- Implementação: Lista Ligada com nó cabeça.



- Operações:
 - 1 `fila_criar()`
 - 2 `fila_inserir()`
 - 3 `fila_topo()`
 - 4 `fila_remove()`
 - 5 `fila_tamanho()`
 - 6 `fila_liberar()`

TAD Filas

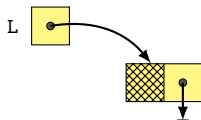
Boa prática:

Vamos **reutilizar** o TAD criado para listas ligadas.

minhaFila.h

```
1  #include "minhaLista.h"
2  /* TAD: Minha Fila */
3
4  /* Tipo Exportado */
5  typedef Lista Fila;
6
7  /* Funções Exportadas */
8  void fila_criar(Fila** p);
9  void fila_inserir(Fila* p, int valor);
10
11  int  fila_topo(Fila* p);
12  void fila_remover(Fila* p);
13
14  int  fila_tamanho(Fila* p);
15  void fila_liberar(Fila** p);
```

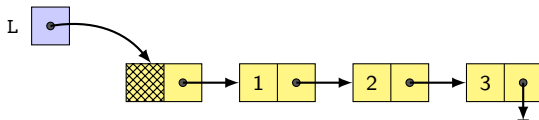
TAD Filas



minhaFila.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "minhaFila.h"
4
5 void fila_criar(Fila** p){
6     cria_lista(p); //(*p)->v = 0;
7 }
8 ...
```

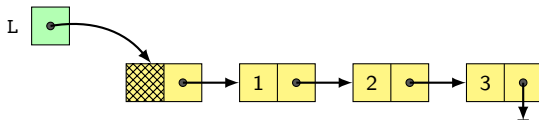

TAD Filas



minhaFila.c

```
1 ...  
2 void fila_inserir(Fila* p, int valor){  
3     insere_final(p, valor);  
4     p->v++;  
5 }  
6 ...
```

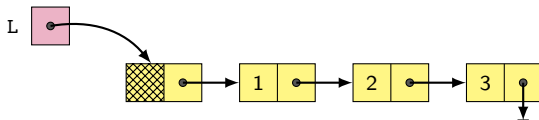
TAD Filas



minhaFila.c

```
1 ...  
2 int fila_topo(Fila* p){  
3     Fila *aux = p->prox;  
4     if(aux!=NULL) return aux->v;  
5     else return -1;  
6 }  
7  
8 void fila_remover(Fila* p){  
9     remove_comeco(p);  
10    p->v--;  
11 }
```

TAD Filas



minhaFila.c

```
1 ...  
2 int fila_tamanho(Fila* p){  
3     return p->v;  
4 }  
5  
6 void fila_liberar(Fila** p){  
7     libera_lista(p);  
8 }
```

TAD Filas

meuProgramaFila.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "minhaFila.h"
4
5 int main(){
6
7     Fila *F;
8     fila_criar(&F);
9
10    int i;
11    for(i=1; i<10; i++) fila_inserir(F, i); // <- 1, 2, 3, ... 9
12
13    while(fila_tamanho(F) > 0){
14        printf("%d\n", fila_topo(&F));
15        fila_remover(F);
16    }
17
18    fila_liberar(&F);
19
20    return 0;
21 }
```

Como compilar?

Temos dois TADs diferentes:

- `minhaFila`
- `minhaLista`

Vamos compilar por partes:

```
1 gcc -Wall -Werror -c meuProgramaFila.c
```

- vai gerar o arquivo compilado `meuProgramaFila.o`

```
1 gcc -Wall -Werror -c minhaLista.c  
2 gcc -Wall -Werror -c minhaFila.c
```

- vai gerar os arquivos `minhaFila.o` e `minhaLista.o`

```
1 gcc meuProgramaFila.o minhaLista.o minhaFila.o -o meuProgramaFila
```

- faz a linkagem, gerando o executável `meuProgramaFila`

Makefile

Criando um **Makefile**:

```
1 all: meuProgramaFila
2
3 meuProgramaFila: meuProgramaFila.o minhaLista.o minhaFila.o
4     gcc meuProgramaFila.o minhaLista.o minhaFila.o -lm -o meuProgramaFila
5
6 meuProgramaFila.o: meuProgramaFila.c minhaLista.h minhaFila.h
7     gcc -Wall -Werror -c meuProgramaFila.c
8
9 %.o: %.c %.h
10    gcc -Wall -Werror -c $<
```

Basta executar **make** na pasta com os arquivos:

- **meuProgramaFila.c**, **minhaFila.c**, **minhaFila.h**, **minhaLista.c**, **minhaLista.h**, e **Makefile**

Apenas recompila o que for necessário!

Makefile

Alguns detalhes:

- Caso `meuProgramaFila.c` utilize dois TADs, precisamos incluir os comandos `#ifndef NOME`, `#define NOME` e `#endif`:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "minhaFila.h"
4  #include "minhaLista.h"
5
6  int main(){
7
8      ...
9      ...
10     ...
11
12     return 0;
13 }
```

```
1  #ifndef LISTA
2  #define LISTA
3
4  /* TAD: Minha Lista */
5  /* Tipo Exportado */
6  ...
7
8  /* Funções Exportadas */
9  ...
10 ...
11 ...
12
13 #endif
```

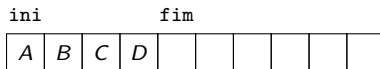
Roteiro

- 1 Filas
- 2 TAD Filas
- 3 Outras alternativas**
- 4 Exemplos de aplicações
- 5 Aplicação de Filas
- 6 Pilhas
- 7 TAD Pilhas
- 8 Outra alternativa
- 9 Exemplos de aplicações
- 10 Aplicação de Pilhas
- 11 Referências

Fila: implementação com vetor

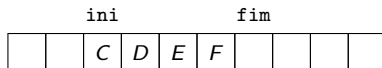
Primeira ideia:

- Inserimos no final do vetor
- Removemos do começo, e movemos para a esquerda ← ruim!



Segunda ideia:

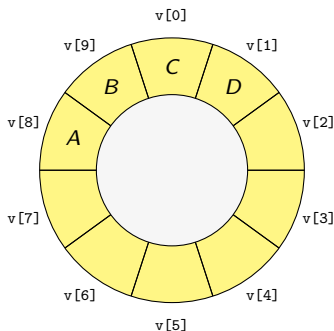
- Variável **ini** indica o começa da fila
- Variável **fim** indica o fim da fila



E se, ao inserir, tivermos espaço apenas à esquerda de **ini**?

Fila: implementação com vetor (fila circular)

Solução: considerar o vetor de tamanho **N** de maneira **circular**



As manipulações de índices são realizadas módulo **N**

Roteiro

- 1 Filas
- 2 TAD Filas
- 3 Outras alternativas
- 4 Exemplos de aplicações**
- 5 Aplicação de Filas
- 6 Pilhas
- 7 TAD Pilhas
- 8 Outra alternativa
- 9 Exemplos de aplicações
- 10 Aplicação de Pilhas
- 11 Referências

Exemplos de aplicações

Algumas aplicações de filas:

- Gerenciamento de fila de impressão
- Buffer do teclado
- Escalonamento de processos
- Comunicação entre aplicativos/computadores
- Percurso de estruturas de dados complexas (grafos etc.)

Roteiro

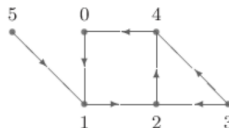
- 1 Filas
- 2 TAD Filas
- 3 Outras alternativas
- 4 Exemplos de aplicações
- 5 Aplicação de Filas**
- 6 Pilhas
- 7 TAD Pilhas
- 8 Outra alternativa
- 9 Exemplos de aplicações
- 10 Aplicação de Pilhas
- 11 Referências

Aplicação de Filas

Aplicação de Filas:

- Imagine N cidades numeradas de 0 a N-1 e **interligadas por estradas** de mão única.
- As **ligações entre as cidades** são representadas por uma matriz $A[N][N]$ da seguinte maneira:
 - $A[i][j]$ **vale 1** se existe estrada de i para j , ou **vale 0** caso contrário.

	0	1	2	3	4	5
0	0	1	0	0	0	0
1	0	0	1	0	0	0
2	0	0	0	0	1	0
3	0	0	1	0	1	0
4	1	0	0	0	0	0
5	0	1	0	0	0	0



¹<https://www.ime.usp.br/~pf/algoritmos/aulas/fila.html>

Aplicação de Filas

Aplicação de Filas:

- A distância de uma cidade i a uma cidade j é o menor número de estradas que precisamos percorrer para ir de i a j.

Nosso problema:

- dada uma cidade i, determinar a distância de i a cada uma das demas cidades.

As distâncias podem ser armazenadas em um vetor **dist**

- dist[j]** é a distância de i \rightsquigarrow j.
- Se for impossível ir de i a j, então **dist[j]=N**.

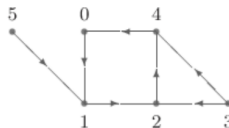
i	0	1	2	3	4	5
dist[i]	2	3	1	0	1	6

Aplicação de Filas

Algoritmo: fila de cidades ativas.

- **Passo 1:** insira **i** na Fila.
- **Passo 2:** Remove o topo da Fila em j, e insere as cidade vizinhas à **j** que ainda não foram visitadas na Fila (ex. j_1, j_2 e j_3). Além disso, $dist[j_i] = dist[j] + 1$
- **Passo 3:** Repita o passo 2 enquanto a Fila não estiver vazia.

	0	1	2	3	4	5
0	0	1	0	0	0	0
1	0	0	1	0	0	0
2	0	0	0	0	1	0
3	0	0	1	0	1	0
4	1	0	0	0	0	0
5	0	1	0	0	0	0



Roteiro

- 1 Filas
- 2 TAD Filas
- 3 Outras alternativas
- 4 Exemplos de aplicações
- 5 Aplicação de Filas
- 6 Pilhas**
- 7 TAD Pilhas
- 8 Outra alternativa
- 9 Exemplos de aplicações
- 10 Aplicação de Pilhas
- 11 Referências

Pilha

Vamos pensar em um **pilha de pratos**:

- **Empilha** os pratos limpos sobre os que já estão na pilha
- **Desempilha** o prato de cima para usar



Pilha:

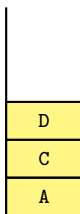
- Remove primeiro objetos inseridos há menos tempo
- **LIFO** (last-in first-out): último a entrar é primeiro a sair

Pilha

Operações:

- **Empilha** (push): adiciona no topo da pilha
- **Desempilha** (pop): remove do topo da pilha

Exemplo: **Empilha(A)** **Empilha(C)** **Empilha(D)**



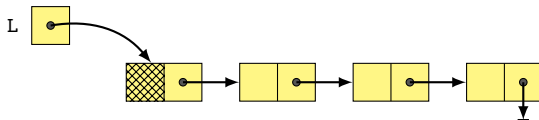
Roteiro

- 1 Filas
- 2 TAD Filas
- 3 Outras alternativas
- 4 Exemplos de aplicações
- 5 Aplicação de Filas
- 6 Pilhas
- 7 TAD Pilhas**
- 8 Outra alternativa
- 9 Exemplos de aplicações
- 10 Aplicação de Pilhas
- 11 Referências

TAD Pilhas

TAD Pilhas:

- Implementação: Lista Ligada.



- Operações:
 - 1 pilha_criar()
 - 2 pilha_inserir()
 - 3 pilha_remove()
 - 4 pilha_topo()
 - 5 pilha_tamanho()
 - 6 pilha_liberar()

TAD Pilhas

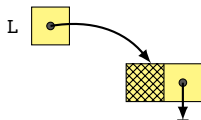
Boa prática:

Vamos **reutilizar** o TAD criado para listas ligadas.

minhaPilha.h

```
1  #include "minhaLista.h"
2  /* TAD: Minha Pilha */
3
4  /* Tipo Exportado */
5  typedef Lista Pilha;
6
7  /* Funções Exportadas */
8  void pilha_criar(Pilha** p);
9  void pilha_inserir(Pilha* p, int valor);
10
11 int pilha_topo(Pilha* p);
12 void pilha_remover(Pilha* p);
13
14 int pilha_tamanho(Pilha* p);
15 void pilha_liberar(Pilha** p);
```

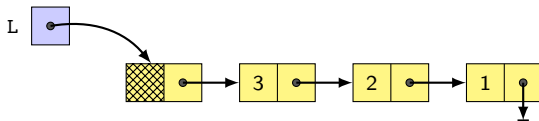
TAD Pilhas



minhaPilha.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "minhaPilha.h"
4
5 void pilha_criar(Pilha** p){
6     cria_lista(p); //(*p)->v = 0;
7 }
8 ...
```

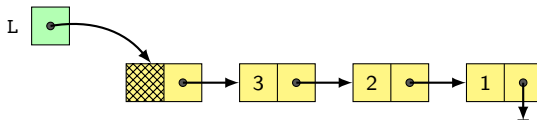
TAD Pilhas



minhaPilha.c

```
1 ...  
2 void pilha_inserir(Pilha* p, int valor){// push 1, 2, 3  
3     insere_comeco(p, valor);  
4     p->v++;  
5 }  
6 ...
```

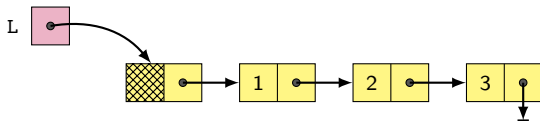

TAD Pilhas



minhaPilha.c

```
1 ...  
2 int pilha_topo(Pilha* p){  
3     Pilha *aux = p->prox;  
4     if(aux!=NULL) return aux->v;  
5     else return -1;  
6 }  
7  
8 void pilha_remove(Pilha* p){  
9     remove_comeco(p);  
10    p->v--;  
11 }
```

TAD Pilhas



minhaPilha.c

```
1 ...  
2 int pilha_tamanho(Pilha* p){  
3     return p->v;  
4 }  
5  
6 void pilha_liberar(Pilha** p){  
7     libera_lista(p);  
8 }
```

TAD Pilhas

meuProgramaPilha.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "minhaPilha.h"
4
5 int main(){
6
7     Pilha *P;
8     pilha_criar(&P);
9
10    int i;
11    for(i=1; i<10; i++) pilha_inserir(P, i); // <- 1, 2, 3, ... 9
12
13    while(pilha_tamanho(P) > 0){
14        printf("%d\n", pilha_topo(P));
15        pilha_remover(P);
16    }
17
18    pilha_liberar(&P);
19
20    return 0;
21 }
```

Como compilar?

Temos dois TADs diferentes:

- `minhaPilha`
- `minhaLista`

Vamos compilar por partes:

```
1 gcc -Wall -Werror -c meuProgramaPilha.c
```

- vai gerar o arquivo compilado `meuProgramaPilha.o`

```
1 gcc -Wall -Werror -c minhaLista.c  
2 gcc -Wall -Werror -c minhaPilha.c
```

- vai gerar os arquivos `minhaPilha.o` e `minhaLista.o`

```
1 gcc meuProgramaPilha.o minhaLista.o minhaPilha.o -o meuProgramaPilha
```

- faz a linkagem, gerando o executável `meuProgramaPilha`

Makefile

Criando um **Makefile**:

```
1 all: meuProgramaPilha
2
3 meuProgramaPilha: meuProgramaPilha.o minhaLista.o minhaPilha.o
4     gcc meuProgramaPilha.o minhaLista.o minhaPilha.o -lm -o meuProgramaPilha
5
6 meuProgramaPilha.o: meuProgramaPilha.c minhaLista.h minhaPilha.h
7     gcc -Wall -Werror -c meuProgramaPilha.c
8
9 %.o: %.c %.h
10    gcc -Wall -Werror -c $<
```

Basta executar **make** na pasta com os arquivos:

- **meuProgramaPilha.c**, **minhaPilha.c**, **minhaPilha.h**, **minhaLista.c**, **minhaLista.h**, e **Makefile**

Apenas recompila o que for necessário!

Alguns detalhes:

- Caso `meuProgramaPilha.c` utilize dois TADs, precisamos incluir os comandos `#ifndef NOME`, `#define NOME` e `#endif`:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "minhaPilha.h"
4 #include "minhaLista.h"
5
6 int main(){
7
8     ...
9     ...
10    ...
11
12    return 0;
13 }
```

```
1 #ifndef LISTA
2 #define LISTA
3
4 /* TAD: Minha Lista */
5 /* Tipo Exportado */
6 ...
7
8 /* Funções Exportadas */
9 ...
10 ...
11 ...
12
13 #endif
```

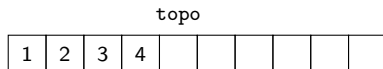
Roteiro

- 1 Filas
- 2 TAD Filas
- 3 Outras alternativas
- 4 Exemplos de aplicações
- 5 Aplicação de Filas
- 6 Pilhas
- 7 TAD Pilhas
- 8 Outra alternativa**
- 9 Exemplos de aplicações
- 10 Aplicação de Pilhas
- 11 Referências

Pilha: implementação com vetor

Implementação com vetor:

- Inserimos no topo do vetor
- Removemos do topo



```
1 typedef struct {
2     int *v;
3     int topo;
4 } Pilha;
5
6 ...
7
8 void pilha_inserir(Pilha *p, int valor) {
9     p->v[p->topo] = valor;
10    p->topo++;
11 }
12
13 int pilha_topo(Pilha *p) {
14     p->topo--;
15     return p->v[p->topo];
16 }
```


Roteiro

- 1 Filas
- 2 TAD Filas
- 3 Outras alternativas
- 4 Exemplos de aplicações
- 5 Aplicação de Filas
- 6 Pilhas
- 7 TAD Pilhas
- 8 Outra alternativa
- 9 Exemplos de aplicações**
- 10 Aplicação de Pilhas
- 11 Referências

Exemplos de aplicações

Algumas aplicações de pilhas:

- Balanceamento de parênteses
 - expressões matemáticas
 - linguagens de programação
 - HTML...
- Cálculo e conversão de notações
 - pré-fixa
 - pós-fixa
 - infixa (com parênteses)
- Percurso de estruturas de dados complexas (grafos etc.)

Roteiro

- 1 Filas
- 2 TAD Filas
- 3 Outras alternativas
- 4 Exemplos de aplicações
- 5 Aplicação de Filas
- 6 Pilhas
- 7 TAD Pilhas
- 8 Outra alternativa
- 9 Exemplos de aplicações
- 10 Aplicação de Pilhas**
- 11 Referências

Exercício

Temos uma **expressão aritmética** e queremos saber se os **parênteses** estão **balanceados**

- Exemplos corretos:
 - $(a + b)$
 - $(a \cdot b) + (c / (d - e))$
- Exemplos incorretos:
 - $(a + b$
 - $(a \cdot b) + (c / d - e))$
 - $)a + b($

Nosso problema:

- Escreva uma função que, dada uma **sequência de parênteses**, diz se ela **é balanceada ou não**
 - Vamos ignorar operandos e operadores
 - $()(())$

Balanceamento de delimitadores

Uma sequência de delimitadores é **balanceada** se for

- vazia
- ou [sequência balanceada]
- ou (sequência balanceada)
- ou a concatenação de duas sequências balanceadas

Exemplos:

Balanceada

[[]]
([() ([])])

Não Balanceada

([]
[[))
([)]

Algoritmo: para testar, leia cada símbolo e se:

- 1 leu (ou [: empilha o símbolo lido
- 2 leu]: desempilha [
- 3 leu): desempilha (

(() [()])

```
1 int eh_balanceada(char *str) {
2     Pilha *P;
3     int i, ok = 1;
4     char par;
5     pilha_criar(&P);
6     for (i = 0; ok && str[i] != '\0'; i++)
7         if (str[i] == '[' && str[i] == '(')
8             pilha_inserir(P, str[i]);
9         else if (pilha_tamanho(P)==0)
10            ok = 0;
11        else {
12            par = pilha_topo(P);
13            pilha_remover(P);
14            if (str[i] == ']' && par != '[')
15                ok = 0;
16            if (str[i] == ')' && par != '(')
17                ok = 0;
18        }
19    if (pilha_tamanho(P)>0) ok = 0;
20    pilha_liberar(&P);
21    return ok;
22 }
```

parenteses.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "minhaPilha.h"
4
5  int eh_balanceada(char *str);
6
7  int main(int argc, char **argv){
8
9      printf("%s\n", argv[1]);
10     if(eh_balanceada(argv[1])) printf("SIM\n");
11     else printf("NAO\n");
12
13     return 0;
14 }
```


Como compilar?

Temos dois TADs diferentes:

- `minhaPilha`
- `minhaLista`

Vamos compilar por partes:

```
1 gcc -Wall -Werror -c parenteses.c
```

- vai gerar o arquivo compilado `parenteses.o`

```
1 gcc -Wall -Werror -c minhaLista.c  
2 gcc -Wall -Werror -c minhaPilha.c
```

- vai gerar os arquivos `minhaPilha.o` e `minhaLista.o`

```
1 gcc parenteses.o minhaLista.o minhaPilha.o -o parenteses
```

- faz a linkagem, gerando o executável `parenteses`

Como compilar?

Vamos executar:

```
1 ./parenteses "() () ()"  
2 () () ()  
3 SIM
```

```
1 ./parenteses "(())"  
2 (() )  
3 SIM
```

```
1 ./parenteses "() () []"  
2 () () []  
3 NAO
```

Dúvidas?

Roteiro

- 1 Filas
- 2 TAD Filas
- 3 Outras alternativas
- 4 Exemplos de aplicações
- 5 Aplicação de Filas
- 6 Pilhas
- 7 TAD Pilhas
- 8 Outra alternativa
- 9 Exemplos de aplicações
- 10 Aplicação de Pilhas
- 11 Referências**

- ① Feofiloff, Paulo. Algoritmos em linguagem C. Elsevier Brasil, 2009.
- ② Materiais adaptados dos slides dos Profs. Rafael Schouery e Lehilton L. C. Pedrosa, da UNICAMP.