Aula 03 - Conteúdo

1) Pilhas utilizando o enfoque seqüencial (vetor)
2) Filas utilizando o enfoque seqüencial (vetor)
3) Exercícios

Pilhas e Filas (Vetor)

Fundamentos

Uma das formas mais comuns em programação é a "lista linear", que pode ser representada por L = (a1, a2, ..., an), onde ai i = 1..n, são elementos de um conjunto específico.

Exemplos:

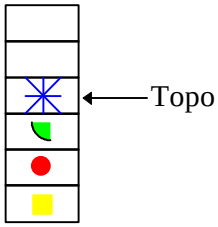
Dias da semana (SEG, TER, QUAR, QUIN, SEX, SAB, DOM)

Cartas do Baralho (Ás, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K)

Pilhas e Filas são dois dos mais comuns objetos de dados encontrados nos algoritmos de computação e são casos especiais das listas lineares.

Pilhas

Pilha (Stack): lista linear, em que todas as inserções e remoções de elementos só podem ser feitas numa extremidade chamada Topo. As pilhas também são chamadas de estruturas LIFO (Last In First Out), ou seja, o último elemento inserido é o primeiro a ser removido (UEPS - último que entra é o 1º sai).



Pilha

A maneira mais simples de se representar uma pilha num computador é através de um vetor de **m** elementos. Digamos que este vetor chama-se **P** e seus elementos tem índices 0, 1, 2, ..., m-1. O número máximo de elemento da pilha será **m**, o elemento da base será **P[0]**, o elemento seguinte **P[1]** e assim por diante. Devido a este fato, esta representação é chamada seqüencial.

Associada a pilha tem-se uma variável chamada **Topo**, de tal forma que o elemento no topo da pilha será **P[Topo]**. Pode-se convencionar que **Topo = -1** indica uma pilha vazia.

Operações associadas com uma pilha:

- Inicializar uma pilha P
- Inserir na pilha P
- Deletar da Pilha P
- Verificar se a pilha P está vazia
- Verificar se a pilha P está cheia

- Devolver o elemento do topo da pilha P

Exemplo de uso de Pilhas

- processamento das chamadas de sub-rotinas e seus retornos
- simulação de sistemas
- Análise de expressões Pós fixas (Calculadores HP)

Algoritmos para Pilha utilizando vetores

```
Convenções:
Topo    = -1 indica pilha vazia
Topo    = TAM_VET - 1 indica pilha cheia
TAM_VET = 10

INICIALIZAÇÃO
Topo = -1

INSERÇÃO
Se Topo = TAM_VET -1 então Escreva("PILHA CHEIA – INS. IMPOSSÍVEL)
senão
  Topo = Topo + 1
  P[Topo] = Elemento
fimse

DELEÇÃO
Se Topo = -1 então Escreva("PILHA VAZIA - REMOÇÃO IMPOSSÍVEL")
senão
  Elemento = P[Topo]
  Topo = Topo -1
fimse
```

Exemplo 3.1 – Implementação de pilha utilizando vetor em Pascal - pilha de strings (arquivo ProjEx301.dpr e Ex301.pas)

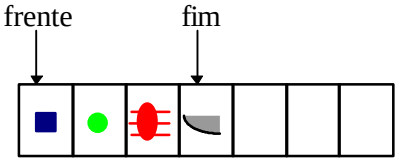
```
Type
  Pilha = class
  protected
    Topo: integer;
    P: array of string;
  public
    Procedure InicPilha(n: integer);
    Function PilhaVazia: boolean;
    Function PilhaCheia: boolean;
    Procedure InsPilha(x: string);
    Procedure DelPilha(var x:string);
    Procedure RetornaValTopo(var x:string);
    Function GetTopo: integer;
    Procedure ImpPilha;
  end;

var
  P1: Pilha;

implementation
{$R *.DFM}

Procedure Pilha.InicPilha(n: integer);
begin
  SetLength(P,n);
  Topo := -1;
```

```
end;

Function Pilha.PilhaVazia: boolean;
begin
  if (Topo = -1 ) then PilhaVazia := TRUE
  else PilhaVazia := FALSE;
end;

Function Pilha.PilhaCheia: boolean;
begin
  if (Topo = High(P) ) then PilhaCheia := TRUE
  else PilhaCheia := FALSE;
end;

Procedure Pilha.InsPilha(x: string);
begin
  if ( Topo < High(P) ) then
  begin
    Topo := Topo + 1;
    P[Topo] := x;
  end
  else
    ShowMessage('Pilha Cheia - Inserção Impossível');
end;

Procedure Pilha.DelPilha(var x:string);
begin
  if ( Topo <> -1 ) then
  begin
    x := P[Topo];
    Topo := Topo - 1;
  end
  else
    ShowMessage('Pilha Vazia - Remoção Impossível');
end;

Procedure Pilha.RetornaValTopo(var x:string);
begin
  if ( Topo <> -1 ) then
    x := P[Topo]
  else
    ShowMessage('Pilha Vazia - Impossível recuperar Topo');
end;

Procedure Pilha.ImpPilha;
var k:integer;
begin
  if ( Topo <> -1 ) then
  begin
    k := 0;
    while ( K <= Topo ) do
    begin
      ShowMessage('Elemento '+IntToStr(k)+': '+ P[k]);
      k := k+1;
    end
  end
  else
    ShowMessage('Pilha Vazia - Impressão Impossível');

end;
```

```
Function Pilha.GetTopo: integer;
begin
  GetTopo := Topo;
end;
```

Filas

Fila (Queue): lista linear em que todas as remoções são feitas numa extremidade chamada frente e todas as inserções são feitas na outra extremidade chamada fim (ré). As filas são chamadas de estruturas FIFO (First In First Out), pois o primeiro elemento inserido é o primeiro a ser removido (PEPS - 1º que entra é o 1º que sai).



Fila

Filas também podem ser representadas através de um vetor de **m** elementos (como as pilhas). Chamaremos este arranjo de **F** sendo que seus elementos tem índices de **0** até **m-1**. Além disso representaremos as duas extremidades da fila por duas variáveis chamadas, por exemplo, **frente** e **fim**.

Operações associadas com uma Fila

- Inicializar uma fila F com começo "frente" e final "fim"
- Inserir na fila F
- Deletar da fila F
- Verificar se F está vazia
- Verificar se F está cheia.
- Devolver o elemento do início da fila F

Exemplos do uso de Filas

- planejamento de serviços (processamento com lote)
- lista de prioridades (processos de 5.0.)
- Simulação

Algoritmos para Fila utilizando vetores

```
Convenções:
frente = fim = -1 indica fila vazia no início
frente = fim != -1 indica fila vazia nas demais situações.
fim = TAM_VET -1 indica fila cheia
TAM_VET = 10

INICIALIZAÇÃO
frente = -1
fim    = -1

INSERÇÃO
Se fim = TAM_VET -1 então “FILA CHEIA - INSERÇÃO IMPOSSÍVEL”
senão
  fim = fim + 1
```

```
  F[fim] = Elemento
fimse

DELEÇÃO
Se frente = fim então "FILA VAZIA - REMOÇÃO IMPOSSÍVEL"
senão
  frente = frente + 1
  Elemento = F[frente]
fimse
```

Exemplo 3.2 – Implementação de fila utilizando vetor em Pascal - fila de strings (arquivo ProjEx302.dpr e Ex302.pas)

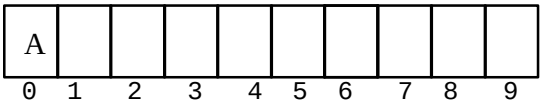
```
Convenções:
frente = fim = -1 indica fila vazia no início
frente = fim != -1 indica fila vazia nas demais situações.
fim = TAM_VET - 1 indica fila cheia

Type
  Fila = class
  protected
    Frente, Fim: integer;
    F: array of string;
  public
    Procedure InicFila(n: integer);
    Function FilaVazia: boolean;
    Function FilaCheia: boolean;
    Procedure InsFila(x: string);
    Procedure DelFila(var x:string);
    Procedure RetornaValFrente(var x:string);
    Function GetFrente: integer;
    Function GetFim: integer;
    Procedure ImpFila;
  end;

var
  F1: Fila;

implementation
{$R *.DFM}

Procedure Fila.InicFila(n: integer);
begin
  SetLength(F,n);
  Frente := -1;
  Fim := -1;
end;

Function Fila.FilaVazia: boolean;
begin
  if (frente = fim ) then FilaVazia := TRUE
  else FilaVazia := FALSE;
end;

Function Fila.FilaCheia: boolean;
begin
  if (Fim = High(F) ) then FilaCheia := TRUE
  else FilaCheia := FALSE;
end;

Procedure Fila.InsFila(x: string);
```

```
begin
  if ( Fim < High(F) ) then
  begin
    Fim := Fim + 1;
    F[Fim] := x;
  end
  else
    ShowMessage('Fila Cheia - Inserção Impossível');
end;

Procedure Fila.DelFila(var x:string);
begin
  if ( Frente <> Fim ) then
  begin
    Frente := Frente + 1;
    x := F[Frente];
  end
  else
    ShowMessage('Fila Vazia - Remoção Impossível');
end;

Procedure Fila.RetornaValFrente(var x:string);
begin
  if ( Frente <> Fim ) then
    x := F[Frente+1]
  else
    ShowMessage('Fila Vazia - impossível recuperar frente');
end;

Procedure Fila.ImpFila;
var k:integer;
begin
  if ( Frente <> Fim ) then
  begin
    k := Frente+1;
    while ( k <= Fim) do
    begin
      ShowMessage('Elemento '+IntToStr(k)+': '+ F[k]);
      k := k+1;
    end
  end
  else
    ShowMessage('Fila Vazia - Impressão Impossível');

end;

Function Fila.GetFrente: integer;
begin
  GetFrente := Frente;
end;

Function Fila.GetFim: integer;
begin
  GetFim := Fim;
end;
```

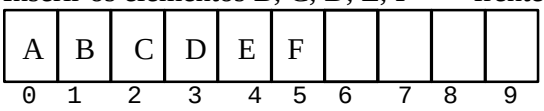
Nesta implementação pode acontecer de estarmos na condição de fila cheia mas com posições livres no início do vetor. Exemplo:

Fila Vazia frente = -1 fim = -1

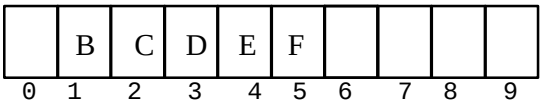
Inserir elemento A frente = -1 fim = 0



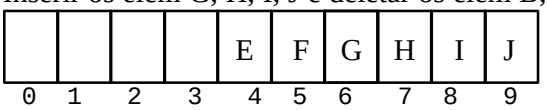
Inserir os elementos B, C, D, E, F frente = -1 fim = 5



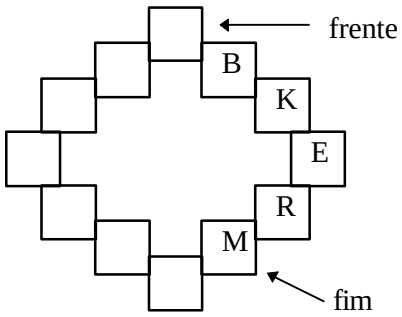
Deletar o elemento A frente = 0 fim = 5



Inserir os elem G, H, I, J e deletar os elem B, C, D frente = 3 fim = 9



Estamos na situação de fila cheia com a variável fim = 9 (fim = TAM_VET-1), porém temos posições livres que poderiam ser utilizadas para novas inserções. Para resolver este problema podemos enfocar o vetor como se fosse circular, ou seja, depois da posição TAM_VET-1 voltaríamos para a posição 0 do vetor.



Algoritmos para Fila Circular utilizando vetores

```
Convenções:
frente = fim = -1 indica fila vazia no início
frente = fim != -1 indica fila vazia nas demais situações.
fim + 1 = frente indica fila cheia
operador % indica resto da divisão inteira. Ex: 17 % 5 = 2
TAM_VET = 10
```

```
INICIALIZAÇÃO
frente = -1
fim    = -1

INSERÇÃO
Se((fim+1)%TAM_VET)=frente)ou(frente=-1 e fim=TAM_VET-1) então
  "FILA CHEIA - INSERÇÃO IMPOSSÍVEL"
senão
  fim = (fim + 1) % TAM_VET
  F[fim] = Elemento
fimse

DELEÇÃO
Se frente = fim então "FILA VAZIA - REMOÇÃO IMPOSSÍVEL"
senão
  frente = (frente + 1) % TAM_VET
  Elemento = F[frente]
fimse
```

Exemplo 4.3 – Implementação de fila circular utilizando vetor em Pascal (arquivo ProjEx303.dpr e Ex303.pas)

```
Convenções:
frente = fim = -1 indica fila vazia no início
frente = fim != -1 indica fila vazia nas demais situações.
fim + 1 = frente indica fila cheia

Type
  Fila = class
  protected
    Frente, Fim: integer;
    F: array of string;
  public
    Procedure InicFila(n: integer);
    Function FilaVazia: boolean;
    Function FilaCheia: boolean;
    Procedure InsFila(x: string);
    Procedure DelFila(var x:string);
    Procedure RetornaValFrente(var x:string);
    Function GetFrente: integer;
    Function GetFim: integer;
    Procedure ImpFila;
  end;

var
  F1: Fila;

implementation
{$R *.DFM}

Procedure Fila.InicFila(n: integer);
begin
  SetLength(F,n);
  Frente := -1;
  Fim := -1;
end;

Function Fila.FilaVazia: boolean;
begin
  if (frente = fim ) then FilaVazia := TRUE
```

```
  else FilaVazia := FALSE;
end;

Function Fila.FilaCheia: boolean;
begin
  if (Fim+1 = Frente ) then FilaCheia := TRUE
  else FilaCheia := FALSE;
end;

Procedure Fila.InsFila(x: string);
var tamF: integer;
begin
  tamF := High(F)+1;
  if ( (Frente = ((Fim+1) mod tamF) ) or
      ((Frente = -1) and (Fim = tamF-1) )) then
    ShowMessage('Fila Cheia - Inserção Impossível')
  else
  begin
    Fim := (Fim + 1) mod tamF;
    F[Fim] := x;
  end
end;

Procedure Fila.DelFila(var x:string);
var tamF: integer;
begin
  tamF := High(F)+1;
  if ( Frente = Fim ) then
    ShowMessage('Fila Vazia - Remoção Impossível')
  else
  begin
    Frente := (Frente + 1) mod tamF;
    x := F[Frente];
  end
end;

Procedure Fila.RetornaValFrente(var x:string);
var tamF: integer;
begin
  tamF := High(F)+1;
  if ( Frente = Fim ) then
    ShowMessage('Fila Vazia - Impossível recuperar frente')
  else
    x := F[(Frente + 1) mod tamF];
end;

Procedure Fila.ImpFila;
var tamF,k:integer;
begin
  tamF := High(F)+1;
  if ( Frente <> Fim ) then
  begin
    k := Frente+1;
    while ( k <> Fim) do
    begin
      ShowMessage('Elemento '+IntToStr(k)+': '+ F[k]);
      k := (k+1) mod tamF;
    end;
    ShowMessage('Índice '+IntToStr(k)+': '+ F[k]);
  end
  else
```

```
    ShowMessage('Fila Vazia - Impressão Impossível');

end;

Function Fila.GetFrente: integer;
begin
  GetFrente := Frente;
end;

Function Fila.GetFim: integer;
begin
  GetFim := Fim;
end;
```

Exercícios

3.01) Escreva um algoritmo para implementar duas pilhas num único vetor (arquivo ProjEx304.dpr e Ex304.pas)

3.02) Desenvolva os algoritmos que implementam duas filas num único vetor (arquivo ProjEx305.dpr e Ex305.pas)

3.03) Desenvolva um algoritmo para fila circular no qual não ocorra a perda de uma posição do vetor (arquivo ProjEx306.dpr e Ex306.pas)

Respostas

3.01) Duas pilhas num único vetor. Devemos elaborarar dois algoritmos de inicialização, inserção e deleção (arquivo ProjEx304.dpr e Ex304.pas).

```
Convecões:
TopoA é o topo da pilha A
TopoB é o topo da pilha B
TopoA = -1 indica pilha A vazia
TopoB = TAM_VET indica pilha B vazia
TopaA + 1 = TopoB indica pilha A e B cheias
TAM_VET = 10 (tamanho do vetor de lementos)

INICIALIZAÇÃO PILHA A
TopoA = -1

INICIALIZAÇÃO PILHA B
TopoB = TAM_VET

INSERÇÃO PILHA A
Se TopoA + 1 = TopoB então “PILHA CHEIA - INSERÇÃO IMPOSSÍVEL”
senão
  TopoA = TopoA + 1
  P[TopoA] = Elemento
fimse

INSERÇÃO PILHA B
Se TopoA + 1 = TopoB então “PILHA CHEIA - INSERÇÃO IMPOSSÍVEL”
senão
  TopoB = TopoB - 1
  P[TopoB] = Elemento
fimse

DELEÇÃO PILHA A
Se TopoA = -1 então “PILHA VAZIA - REMOÇÃO IMPOSSÍVEL”
```

```
senão
  Elemento = P[TopoA]
  TopoA = TopoA -1
fimse

DELEÇÃO PILHA B
Se TopoB = TAM_VET então “PILHA VAZIA - REMOÇÃO IMPOSSÍVEL”
senão
  Elemento = P[TopoB]
  TopoB = TopoB + 1
Fimse



Em Pascal
Type
  Pilha = class
  protected
    TopoA,TopoB: integer;
    P: array of string;
  public
    Procedure InicPilha(n: integer);
    Function InsPilhaA(x: string):boolean;
    Function InsPilhaB(x: string):boolean;
    Function DelPilhaA(var x:string):boolean;
    Function DelPilhaB(var x:string):boolean;
    Procedure RetornaValTopoA(var x:string);
    Procedure RetornaValTopoB(var x:string);
    Function GetTopoA: integer;
    Function GetTopoB: integer;
    Procedure ImpPilha;
  end;

var
  P1: Pilha;

implementation
{$R *.DFM}

Procedure Pilha.InicPilha(n: integer);
begin
  SetLength(P,n);
  TopoA := -1;
  TopoB := n;
end;


Function Pilha.InsPilhaA(x: string):boolean;
begin
  if ( TopoA = (TopoB-1) ) then
  begin
    ShowMessage('Pilhas A cheia - Inserção Impossível');
    InsPIlhaA := FALSE;
  end
  else
  begin
    TopoA := TopoA + 1;
    P[TopoA] := x;
    InsPilhaA := TRUE;
  end
end;
```

```
Function Pilha.InsPilhaB(x: string):boolean;
begin
  if ( TopoA = (TopoB-1) ) then
  begin
    ShowMessage('Pilhas B cheia - Inserção Impossível');
    InsPIlhaB := FALSE;
  end
  else
  begin
    TopoB := TopoB - 1;
    P[TopoB] := x;
    InsPIlhaB := TRUE;
  end
end;

Function Pilha.DelPilhaA(var x:string):boolean;
begin
  if ( TopoA <> -1 ) then
  begin
    x := P[TopoA];
    TopoA := TopoA - 1;
    DelPIlhaA := TRUE;
  end
  else
  begin
    ShowMessage('Pilha A Vazia - Remoção Impossível');
    DelPilhaA := FALSE;
  end;
end;

Function Pilha.DelPilhaB(var x:string):boolean;
begin
  if ( TopoB <> (High(P)+1) ) then
  begin
    x := P[TopoB];
    TopoB := TopoB + 1;
    DelPilhaB := TRUE;
  end
  else
  begin
    ShowMessage('Pilha B Vazia - Remoção Impossível');
    DelPilhaB := FALSE;
  end;
end;

Procedure Pilha.RetornaValTopoA(var x:string);
begin
  if ( TopoA <> -1 ) then
    x := P[TopoA]
  else
    ShowMessage('Pilha A Vazia - Impossível recuperar Topo');
end;

Procedure Pilha.RetornaValTopoB(var x:string);
begin
  if ( TopoB <> High(P)+1 ) then
    x := P[TopoB]
  else
    ShowMessage('Pilha B Vazia - Impossível recuperar Topo');
end;
```

```
Procedure Pilha.ImpPilha;
var k:integer;
begin
  if ( TopoA <> -1 ) then
  begin
    k := 0;
    while ( K <= TopoA ) do
    begin
      ShowMessage('Pilha A - elemento '+IntToStr(k)+': '+ P[k]);
      k := k+1;
    end
  end
  else
    ShowMessage('Pilha A Vazia - Impressão Impossível');

  if ( TopoA <> High(P)+1 ) then
  begin
    k := High(P);
    while ( K >= TopoB ) do
    begin
      ShowMessage('Pilha B - elemento '+IntToStr(k)+': '+ P[k]);
      k := k-1;
    end
  end
  else
    ShowMessage('Pilha B Vazia - Impressão Impossível');
end;

Function Pilha.GetTopoA: integer;
begin
  GetTopoA := TopoA;
end;

Function Pilha.GetTopoB: integer;
begin
  GetTopoB := TopoB;
end;
```

3.02) Duas filas num único vetor. Devemos elaborar dois algoritmos de inicialização, inserção e deleção (arquivo ProjEx305.dpr e Ex305.pas)

```
Convecões:
frenteA e fimA indicam início e fim da fila A
frenteB e fimB indicam início e fim da fila B
frenteA = fimA indica fila A vazia
frenteB = fimB indica fila B vazia
fimA + 1 = fimB indica filas A e B cheias
TAM_VET = 10 (tamanho do vetor de lementos)

INICIALIZAÇÃO FILA A
frenteA = -1
fimA    = -1

INICIALIZAÇÃO FILA B
frenteB = TAM_VET
fimB    = TAM_VET

INSERÇÃO FILA A
Se fimA + 1 = fimB então “FILA CHEIA - INSERÇÃO IMPOSSÍVEL”
senão
  fimA = fimA + 1
```

```
  F[fimA] = Elemento
fimse

INSERÇÃO FILA B
Se fimA + 1 = fimB então “FILA CHEIA - INSERÇÃO IMPOSSÍVEL”
senão
  fimB = fimB - 1
  F[fimB] = Elemento
fimse

DELEÇÃO FILA A
Se frenteA = fimA então “FILA VAZIA - REMOÇÃO IMPOSSÍVEL”
senão
  frenteA = frenteA + 1
  Elemento = F[frenteA]
fimse

DELEÇÃO FILA B
Se frenteB = fimB então “FILA VAZIA - REMOÇÃO IMPOSSÍVEL”
senão
  frenteB = frenteB - 1
  Elemento = F[frenteB]
fimse
```

3.03) Algoritmos para Fila Circular utilizando vetores - versão evitando a perda de um elemento do vetor (arquivo ProjEx306.dpr e Ex306.pas)

```
Convenções:
frente = fim = -1 indica fila vazia no início
frente = fim != -1 indica um único elemento na fila
fim + 1 = frente indica fila cheia
operador % indica resto da divisão inteira. Ex: 17 % 5 = 2
TAM_VET = 10

INICIALIZAÇÃO
frente = -1
fim    = -1

INSERÇÃO
Se ((fim + 1) % TAM_VET) = frente então
  “FILA CHEIA - INSERÇÃO IMPOSSÍVEL”
senão
  Se ( fim = -1 )           //primeira inserção
    frente = frente + 1
  fimse
  fim = (fim + 1) % TAM_VET
  F[fim] = Elemento
fimse

DELEÇÃO
Se fim = -1 então “FILA VAZIA - REMOÇÃO IMPOSSÍVEL”
senão
  Elemento = F[frente]
  Se ( frente = fim )
    fim = -1
    frente = -1
  senão
    frente = (frente + 1) % TAM_VET
  fimse
fimse
```