Pilha, Fila e Lista
Algoritmos e Estrutura de Dados
Pilha, Fila e Lista
(Alocação Estática)

prof. Frederico Santos de Oliveira

Universidade Federal de Mato Grosso
Instituto de Engenharia

Agenda

Exercício 1

Implemente uma pilha utilizando alocação estática. As operações básicas são:

1. CriaPilhaVazia(S)
2. PilhaVazia(S)
3. Empilha(S,x)
4. Desempilha(S)

Exercício 1

Para isso, o primeiro passo é criar a estrutura de dados chamada Pilha, que possui uma variável chamada Topo, que aponta para o índice do elemento no topo da pilha, e um vetor de itens.

Algoritmo 1: Pilha

```
1 início
2     registro {
3         Inteiro: itens[MAX];
4         Inteiro Topo;
5     } Pilha;
```

Exercício 1

Na linguagem C, a implementação fica conforme o código a seguir:

```c
#define MAX 1000

typedef struct {
  int itens[MAX];
  int topo;
} pilha;
```

Assim, definindo uma constante chamada MAX com valor igual a 1000, será possível incluir no máximo 1000 itens na nossa pilha.

Pilhas

CriaPilhaVazia

A seguir, tem-se o pseudo-código referente à função CriaPilhaVazia:

Algoritmo 2: CriaPilhaVazia

Entrada: Pilha S

```
1 início
2 |  S.topo ← 0
```

Pilhas

CriaPilhaVazia

Sua implementação em C, fica da seguinte forma:

```
1 void CriaPilhaVazia(pilha *S) {
2   S -> topo = 0;
3 }
```

Observe que a função recebe como parâmetro um ponteiro do tipo `pilha`. Ao chamar essa função, caso tenha declarado uma variável estática do tipo pilha, será necessário utilizar o operador de endereçamento (&):

```
1 pilha S;
2 CriaPilhaVazia(&S);
```

Caso utilize um ponteiro, não é necessário utilizar o operador de endereçamento:

```
1 pilha *S  = malloc(sizeof(pilha));
2 CriaPilhaVazia(S);
```

Pilhas

PilhaVazia

Agora, vamos implementar a função que verifica se uma pilha está vazia.

Algoritmo 3: PilhaVazia

Entrada: Pilha S
Saída: Booleano informando se S está vazia

```
1 início
2     se (S.topo = 0) então
3         retorna Verdadeiro
4     senão
5         retorna Falso
```

A implementação na linguagem C fica da seguinte forma:

```c
int PilhaVazia(pilha S) {
  return (S.topo == 0);
}
```

Pilhas

Empilhar

A próxima função é `Empilha`:

Algoritmo 4: Empilhar

Entrada: Pilha S, item x a ser empilhado

```
1 início
2     se (S.topo = S.tamanhoMáximo) então
3         Erro overflow: pilha cheia.
4     senão
5         S.itens[S.topo] ← x
6         S.topo ← S.topo + 1
```

Pilhas

Empilhar

A implementação na linguagem C fica da seguinte forma:

```
1 void Empilha(int x, pilha *S) {
2   if (S -> topo == MAX)
3     printf(" Erro   pilha esta  cheia\n");
4   else  {
5     S->itens[S->topo] = x;
6     S->topo++;
7   }
8 }
```

Observe que essa função recebe um ponteiro como parâmetro de entrada.

Pilhas

Empilhar

Caso tenha declarado a pilha S como uma variável estática, ao invocar a função `Empilha`, a solução é utilizar o *operador de endereço* (&):

```
1 pilha S;
2 Empilha(10, &S)
```

Dessa forma, é passado como parâmetro o endereço da variável *S*, que será armazenado em um ponteiro.

Pilhas

Desempilhar

A próxima função é `Desempilha`:

Algoritmo 5: Desempilhar

Entrada: Pilha S
Saída: Elemento desempilhado

```
1 início
2     se (PilhaVazia(S)) então
3         retorna underflow: pilha vazia.
4     senão
5         x ← S.itens[ S.topo - 1]
6         S.topo ← S.topo - 1
7         retorna x
```

Pilhas

Desempilhar

A implementação na linguagem C fica da seguinte forma:

```
1 void Desempilha(pilha *S, int *item) {
2   if (PilhaVazia(*S))
3     printf(" Erro   pilha esta  vazia\n");
4   else
5   {
6     *item = S->itens[S->topo - 1];
7     S->topo--;
8   }
9 }
```

Observe que, assim como a função `Empilha`, essa função também recebe um ponteiro como parâmetro de entrada, e, além disso, também recebe um ponteiro no qual ficará armazenado o elemento desempilhado.

Pilhas

Desempilhar

Assim, para utilizar a função desempilha, pode-se utilizar o operador de endereçamento, conforme o código a seguir:

```
1  int main() {
2      // Declaracao de variaveis
3    pilha S;
4    int item = 0;
5    // Cria uma pilha vazia
6    CriaPilhaVazia(&S);
7      // Empilha elemento 10 na pilha
8    Empilha(10, &S);
9    // Desempilha elemento
10   Desempilha(&S, &item);
11   printf("Desempilhou %d.", item) ;
12
13   return(0);
14 }
```

Pilhas

Desempilhar

Também é possível utilizar um ponteiro do tipo `pilha`, e assim, não é necessário utilizar o operador de endereçamento:

```
int main() {
    // Declaracao de variaveis
  pilha *S = malloc(sizeof(pilha));
  int item = 0;
  // Cria uma pilha vazia
  CriaPilhaVazia(S);
    // Empilha elemento 10 na pilha
  Empilha(10, S);
  // Desempilha elemento
  Desempilha(S, &item);
  printf("Desempilhou %d.", item) ;
  // Desaloca memoria
  free(S);

  return(0);
}
```

Exercício 2

Implemente uma fila utilizando alocação estática. As operações básicas são:

1. CriaFilaVazia(Q)
2. FilaVazia(Q)
3. Enfileira(Q,x)
4. Desenfileira(Q)

Exercício 2

Assim como no exercício anterior, o primeiro passo é criar a estrutura de dados chamada Fila, que possui uma variável chamada `início` e outra chamada `fim`, e um vetor de `itens`.

Algoritmo 6: Fila

```
1 início
2     registro {
3         Inteiro: itens[MAX];
4         Inteiro início, fim;
5     } Fila;
```

Filas

A implementação na linguagem C fica da seguinte forma:

```c
#define MAX  1000

typedef struct {
  int itens[MAX];
  int inicio, fim;
} fila;
```

Definindo uma constante chamada MAX com valor igual a 1000, será possível incluir no máximo 1000 itens na nossa fila.

Fila

CriaFilaVazia

A próxima função é `CriaFilaVazia`:

Algoritmo 7: CriaFilaVazia

Entrada: Fila Q

1 **início**
2 $\quad Q$.início $\leftarrow 0$
3 $\quad Q$.fim $\leftarrow Q$.início

A implementação na linguagem C fica da seguinte forma:

```
1 void CriaFila(fila *Q) {
2   Q->inicio = 0;
3   Q->fim = Q->inicio;
4 }
```

Fila

FilaVazia

A próxima função é `FilaVazia`:

Algoritmo 8: FilaVazia

Entrada: Fila Q.

```
1 início
2     se (Q.início = Q.fim) então
3         retorna Verdadeiro
4     senão
5         retorna Falso
```

A implementação na linguagem C fica da seguinte forma:

```c
int FilaVazia(fila Q)
{
  return (Q.inicio == Q.fim);
}
```

Fila

Enfileira

A próxima função é `Enfileira`:

Algoritmo 9: Enfileira

Entrada: Fila Q, item x

```
1 início
2     se ( (Q.fim) MOD (Q.tamanhoMáximo) + 1 = Q.início) então
3         Erro overflow: fila cheia.
4     senão
5         Q.itens[Q.fim] ← x
6         Q.fim ← (Q.fim) MOD (Q.tamanhoMáximo) + 1
```

Fila

Enfileira

A implementação na linguagem C fica da seguinte forma:

```
1 void Enfileira(int x, fila *Q) {
2   if ((Q->fim+1 )% MAX  == Q->inicio)
3     printf(" Erro   fila esta  cheia\n");
4   else {
5     Q->itens[Q->fim - 1] = x;
6     Q->fim = Q->fim % MAX + 1;
7   }
8 }
```

Fila

Desenfileira

A próxima função é `Desenfileira`:

Algoritmo 10: Desenfileira

Entrada: Fila Q
Saída: Elemento desenfileirado

```
1 início
2     se FilaVazia(Q) então
3         retorna Erro underflow: fila vazia.
4     senão
5         x ← Q.itens[Q.início]
6         Q.início ← (Q.início) MOD (Q.tamanhoMáximo) + 1
7         retorna x
```

Fila

Desenfileira

A implementação na linguagem C fica da seguinte forma:

```
1 void Desenfileira(fila *Q, int *item) {
2   if (FilaVazia(*Q))
3     printf("Erro fila esta vazia\n");
4   else {
5     *item = Q->itens[Q->inicio - 1];
6     Q->inicio = Q->inicio % MAX + 1;
7   }
8 }
```

Pilhas

Desempilhar

Observe que ao utilizar uma variável estática é necessário usar o operador de endereçamento, conforme o código a seguir:

```
int main() {
    // Declaracao de variaveis
  fila Q;
  int item = 0;
  // Cria uma fila vazia
  CriaFilaVazia(&Q);
    // Enfileira elemento 10 na fila
  Enfileira(10, &Q);
  // Desenfileira elemento
  Desenfileira(&Q, &item);
  printf("Desenfileirou %d.", item) ;

  return(0);
}
```

Pilhas

Desempilhar

Também é possível utilizar um ponteiro do tipo `fila`, e assim, não é necessário utilizar o operador de endereçamento:

```
int main() {
    // Declaracao de variaveis
  fila *Q = malloc(sizeof(fila));
  int item = 0;
  // Cria uma fila vazia
  CriaFilaVazia(Q);
    // Enfileira elemento 10 na fila
  Enfileira(10, Q);
  // Desenfileira elemento
  Desenfileira(Q, &item);
  printf("Desenfileirou %d.", item) ;
  // Desaloca memoria
  free(Q);
  return(0);
}
```

Exercício 3

Implemente uma lista utilizando alocação estática. As operações básicas são:

1. CriaListaVazia(L)
2. ListaVazia(L)
3. Inserir(L, x, p)
4. Remover(L, p)

Exercício 3

Para isso, o primeiro passo é criar a estrutura de dados chamada Lista, que possui uma variável chamada primeiro, outra chamada último, e um vetor de itens.

Algoritmo 11: Lista

```
1 início
2     registro {
3         Inteiro: itens[MAX];
4         Inteiro primeiro, último
5     } Lista;
```

Exercício 1

Na linguagem C, a implementação fica conforme o código a seguir:

```c
#define MAX 1000

typedef struct {
  int itens[MAX];
  int primeiro, ultimo;
} lista;
```

Assim, definindo uma constante chamada MAX com valor igual a 1000, será possível incluir no máximo 1000 itens na nossa lista.

Lista

A primeira função é `CriaListaVazia`:

Algoritmo 12: CriaListaVazia

Entrada: Lista L

```
1 início
2     L.primeiro ← 0
3     L.ultimo ← L.primeiro
```

Lista

CriaListaVazia

A implementação na linguagem C fica da seguinte forma:

```
1 void CriaListaVazia(lista *l) {
2   l->primeiro = 0;
3   l->ultimo = l->primeiro;
4 }
```

A próxima função é `ListaVazia`:

Algoritmo 13: ListaVazia

Entrada: Lista L.

```
1 início
2     se (L.primeiro = L.ultimo) então
3         retorna Verdadeiro
4     senão
5         retorna Falso
```

Lista

ListaVazia

A implementação na linguagem C fica da seguinte forma:

```
1 int ListaVazia(lista l) {
2   return (l.primeiro == l.ultimo);
3 }
```

Lista

Insere

A próxima função é `Insere`:

Algoritmo 14: Inserir

Entrada: Lista L, elemento x, posição p

```
1  início
2      se ( (L.ultimo + 1) MOD L.tamanhoMáximo = L.primeiro ) então
3          Erro overflow: lista cheia.
4      senão
5          para ( i ← L.ultimo até p) faça
6              L.itens[ i+1 ] ← L.itens[ i ]
7          L.itens[ p ] ← x
8          L.ultimo = L.ultimo + 1
```

Lista

Insere

A implementação na linguagem C fica da seguinte forma:

```
1  void Insere(lista *l, int x, int p) {
2    if ((l->ultimo+1 )% MAX  == l->primeiro)
3      printf("Lista esta cheia\n");
4    else  {
5      for (int i = l->ultimo; i < p; i++)
6        l->itens[i+1] = l->itens[i];
7      l->itens[p] = x;
8      l->ultimo++;
9    }
10 }
```

Lista

Remove

A próxima função é `Remove`:

Algoritmo 15: Remove

Entrada: Lista L, posição p
Saída: Elemento removido

```
1 início
2     se ListaVazia(L) então
3         retorna Erro underflow: lista vazia.
4     senão
5         x ← L.itens[ p ]
6         para ( i ← p até L.ultimo ) faça
7             L.itens[ i ] ← L.itens[ i+1 ]
8         L.ultimo = L.ultimo - 1
9         retorna x
```

Lista

Remove

A implementação na linguagem C fica da seguinte forma:

```
1 void Remove(int p, lista *l, int *item) {
2   if (ListaVazia(*l)) {
3     printf(" Erro   Lista vazia\n");
4   }
5   *item = l->itens[p];
6   for (int aux = p; aux < l->ultimo; aux++)
7     l->itens[aux] = l->itens[aux+1];
8   l->ultimo--;
9 }
```

Pilha, Fila e Lista
Algoritmos e Estrutura de Dados
Pilha, Fila e Lista
(Alocação Estática)

prof. Frederico Santos de Oliveira

Universidade Federal de Mato Grosso
Instituto de Engenharia