



Estrutura de Dados

Pilhas

Prof. Thiago Caproni Tavares ¹ Paulo Muniz de Ávila ²

¹thiago.tavares@ifsuldeminas.edu.br

²paulo.avila@ifsuldeminas.edu.br

Última Atualização: 2 de fevereiro de 2016

1 Introdução

2 Operações

- Inicialização
- Inserindo na Lista
- Remoções

3 Análise da Complexidade

1 Introdução

2 Operações

- Inicialização
- Inserindo na Lista
- Remoções

3 Análise da Complexidade

- Utiliza uma política LIFO (*Last In First Out*): o primeiro elemento inserido será o último a ser removido;
- Cada elemento armazena um ou vários dados (estrutura homogênea ou heterogênea) e um ponteiro para o próximo, permitindo o encadeamento e mantendo a estrutura linear;
- **Operações:** inserir, consultar, remover e esvaziar;
- Qualquer estrutura desse tipo possui um ponteiro denominado TOPO, no qual todas as operações de inserção e remoção acontecem;
 - Assim, as operações ocorrem sempre na mesma extremidade.

1 Introdução

2 Operações

- Inicialização
- Inserindo na Lista
- Remoções

3 Análise da Complexidade

1 Introdução

2 Operações

- Inicialização

- Inserindo na Lista

- Remoções

3 Análise da Complexidade

Inicialização

1ª Operação:
Inicializa a Pilha

```
void inicializar (Pilha *pilha){  
    pilha->topo = NULL;  
    pilha->tam = 0;  
}
```

Lista	
tam	
lixo	
topo	
NULL	

Inicialização

1ª Operação:
Inicializa a Pilha

```
void inicializar (Pilha *pilha){  
    pilha->topo = NULL;  
    pilha->tam = 0;  
}
```

Lista

tam
0
topo
NULL

1 Introdução

2 Operações

- Inicialização
- **Inserindo na Lista**
- Remoções

3 Análise da Complexidade

Inserindo no Início da Lista

2ª Operação:
Inserção do #9

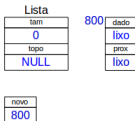
```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

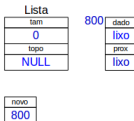
2ª Operação:
Inserção do #9

```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));
    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

2ª Operação:
Inserção do #9

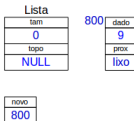
```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

2ª Operação:
Inserção do #9

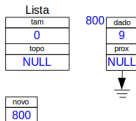
```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

2ª Operação:
Inserção do #9

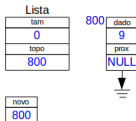
```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

2ª Operação:
Inserção do #9

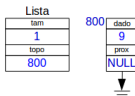
```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

2ª Operação:
Inserção do #9

```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

3ª Operação:
Inserção do #3

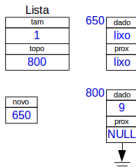
```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

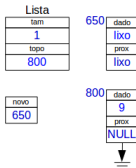
3ª Operação:
Inserção do #3

```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));
    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

3ª Operação:
Inserção do #3

```
int push(Pilha *pilha, int dado)
```

```
{
```

```
    cel *novo = malloc(sizeof(cel));
```

```
    if(novo == NULL)
```

```
        return 0;
```

```
    novo->dado = dado;
```

```
    novo->prox = pilha->topo;
```

```
    pilha->topo = novo;
```

```
    pilha->tam++;
```

```
    return 1;
```

```
}
```

Lista

tam
1
topo
800

novo
650

650
dado
3
prox
lixo

800
dado
9
prox
NULL



Inserindo no Início da Lista

3ª Operação:
Inserção do #3

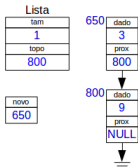
```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

3ª Operação:
Inserção do #3

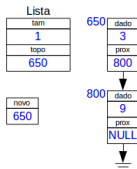
```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

3ª Operação:
Inserção do #3

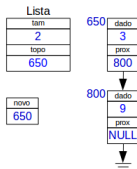
```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

3ª Operação:
Inserção do #3

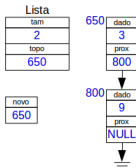
```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

4ª Operação:
Inserção do #5

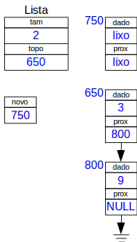
```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

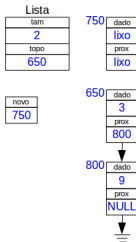
4ª Operação:
Inserção do #5

```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));
    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

4ª Operação:
Inserção do #5

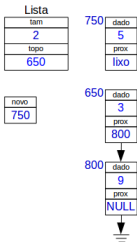
```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

4ª Operação:
Inserção do #5

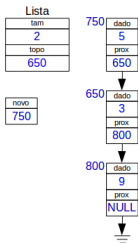
```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

4ª Operação:
Inserção do #5

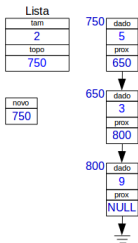
```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

4ª Operação:
Inserção do #5

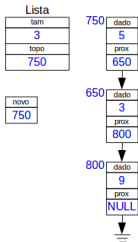
```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

4ª Operação:
Inserção do #5

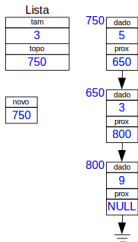
```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

5ª Operação:
Inserção do #1

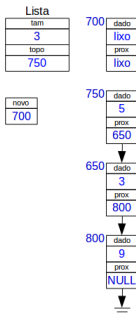
```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

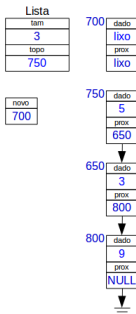
    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

5ª Operação:
Inserção do #1

```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));
    if(novo == NULL)
        return 0;
    novo->dado = dado;
    novo->prox = pilha->topo;
    pilha->topo = novo;
    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

5ª Operação:
Inserção do #1

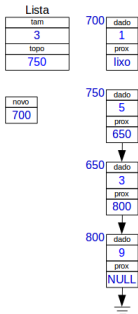
```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

5ª Operação:
Inserção do #1

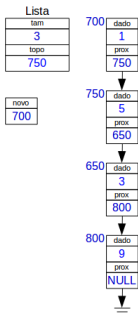
```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

5ª Operação:
Inserção do #1

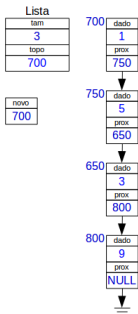
```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

5ª Operação:
Inserção do #1

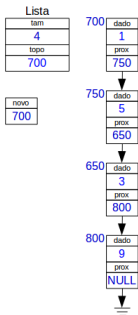
```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



Inserindo no Início da Lista

5ª Operação:
Inserção do #1

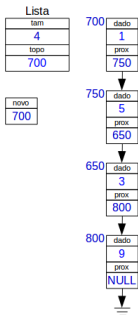
```
int push(Pilha *pilha, int dado)
{
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;

    novo->prox = pilha->topo;
    pilha->topo = novo;

    pilha->tam++;
    return 1;
}
```



1 Introdução

2 Operações

- Inicialização
- Inserindo na Lista
- Remoções

3 Análise da Complexidade

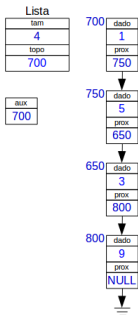
Removendo Elementos

6ª Operação:
Remoção da Pilha

```
int pop(Pilha *pilha)
{
    cel *aux = pilha->topo;
    int rem = aux->dado;

    pilha->topo = pilha->topo->prox;
    free(aux);

    pilha->tam--;
    return rem;
}
```



Removendo Elementos

6ª Operação:
Remoção da Pilha

```
int pop(Pilha *pilha)
{
    cel *aux = pilha->topo;
    int rem = aux->dado;

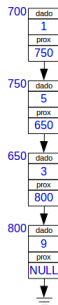
    pilha->topo = pilha->topo->prox;
    free(aux);

    pilha->tam--;
    return rem;
}
```

Lista

tam
4
topo
700

aux
700
rem
1



Removendo Elementos

6ª Operação:
Remoção da Pilha

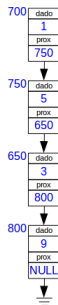
```
int pop(Pilha *pilha)
{
    cel *aux = pilha->topo;
    int rem = aux->dado;

    pilha->topo = pilha->topo->prox;
    free(aux);

    pilha->tam--;
    return rem;
}
```

Lista	
tam	4
topo	750

aux	700
rem	1



Removendo Elementos

6ª Operação:
Remoção da Pilha

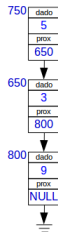
```
int pop(Pilha *pilha)
{
    cel *aux = pilha->topo;
    int rem = aux->dado;

    pilha->topo = pilha->topo->prox;
    free(aux);

    pilha->tam--;
    return rem;
}
```

Lista	
tam	4
topo	750

aux	700
rem	1

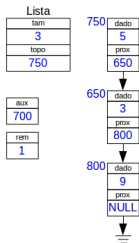


Removendo Elementos

6ª Operação:
Remoção da Pilha

```
int pop(Pilha *pilha)
{
    cel *aux = pilha->topo;
    int rem = aux->dado;

    pilha->topo = pilha->topo->prox;
    free(aux);
    pilha->tam--;
    return rem;
}
```



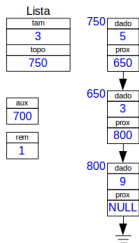
Removendo Elementos

6ª Operação:
Remoção da Pilha

```
int pop(Pilha *pilha)
{
    cel *aux = pilha->topo;
    int rem = aux->dado;

    pilha->topo = pilha->topo->prox;
    free(aux);

    pilha->tam--;
    return rem;
}
```



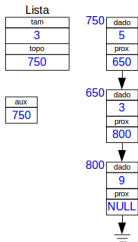
Removendo Elementos

7ª Operação:
Remoção da Pilha

```
int pop(Pilha *pilha)
{
    cel *aux = pilha->topo;
    int rem = aux->dado;

    pilha->topo = pilha->topo->prox;
    free(aux);

    pilha->tam--;
    return rem;
}
```



Removendo Elementos

7ª Operação:
Remoção da Pilha

```
int pop(Pilha *pilha)
{
    cel *aux = pilha->topo;
    int rem = aux->dado;

    pilha->topo = pilha->topo->prox;
    free(aux);

    pilha->tam--;
    return rem;
}
```

Lista	
tam	
3	
topo	
750	

aux	
750	
rem	
5	



Removendo Elementos

7ª Operação:
Remoção da Pilha

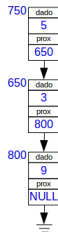
```
int pop(Pilha *pilha)
{
    cel *aux = pilha->topo;
    int rem = aux->dado;

    pilha->topo = pilha->topo->prox;
    free(aux);

    pilha->tam--;
    return rem;
}
```

Lista	
tam	
3	
topo	
650	

aux	
750	
rem	
5	



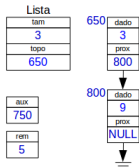
Removendo Elementos

7ª Operação:
Remoção da Pilha

```
int pop(Pilha *pilha)
{
    cel *aux = pilha->topo;
    int rem = aux->dado;

    pilha->topo = pilha->topo->prox;
    free(aux);

    pilha->tam--;
    return rem;
}
```

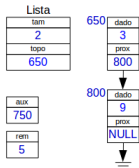


Removendo Elementos

7ª Operação:
Remoção da Pilha

```
int pop(Pilha *pilha)
{
    cel *aux = pilha->topo;
    int rem = aux->dado;

    pilha->topo = pilha->topo->prox;
    free(aux);
    pilha->tam--;
    return rem;
}
```



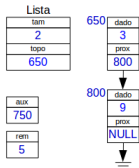
Removendo Elementos

7ª Operação:
Remoção da Pilha

```
int pop(Pilha *pilha)
{
    cel *aux = pilha->topo;
    int rem = aux->dado;

    pilha->topo = pilha->topo->prox;
    free(aux);

    pilha->tam--;
    return rem;
}
```



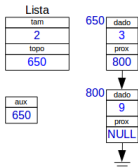
Removendo Elementos

8ª Operação:
Remoção da Pilha

```
int pop(Pilha *pilha)
{
    cel *aux = pilha->topo;
    int rem = aux->dado;

    pilha->topo = pilha->topo->prox;
    free(aux);

    pilha->tam--;
    return rem;
}
```



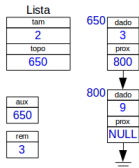
Removendo Elementos

8ª Operação:
Remoção da Pilha

```
int pop(Pilha *pilha)
{
    cel *aux = pilha->topo;
    int rem = aux->dado;

    pilha->topo = pilha->topo->prox;
    free(aux);

    pilha->tam--;
    return rem;
}
```



Removendo Elementos

8ª Operação:
Remoção da Pilha

```
int pop(Pilha *pilha)
{
    cel *aux = pilha->topo;
    int rem = aux->dado;

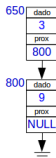
    pilha->topo = pilha->topo->prox;
    free(aux);

    pilha->tam--;
    return rem;
}
```

Lista

tam
2
topo
800

aux
650
rem
3



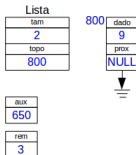
Removendo Elementos

8ª Operação:
Remoção da Pilha

```
int pop(Pilha *pilha)
{
    cel *aux = pilha->topo;
    int rem = aux->dado;

    pilha->topo = pilha->topo->prox;
    free(aux);

    pilha->tam--;
    return rem;
}
```



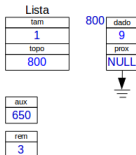
Removendo Elementos

8ª Operação:
Remoção da Pilha

```
int pop(Pilha *pilha)
{
    cel *aux = pilha->topo;
    int rem = aux->dado;

    pilha->topo = pilha->topo->prox;
    free(aux);

    pilha->tam--;
    return rem;
}
```



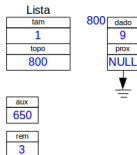
Removendo Elementos

8ª Operação:
Remoção da Pilha

```
int pop(Pilha *pilha)
{
    cel *aux = pilha->topo;
    int rem = aux->dado;

    pilha->topo = pilha->topo->prox;
    free(aux);

    pilha->tam--;
    return rem;
}
```



1 Introdução

2 Operações


- Inicialização
- Inserindo na Lista
- Remoções


3 Análise da Complexidade

- Inserção e remoção sempre realizam operações básicas para atualizar o topo da pilha:
 - São operações de tempo constante e gastam $O(1)$.
- Consultar toda a pilha percorre os elementos armazenados. Uma pilha contém n elementos:
 - logo o tempo de execução é $O(n)$.
- A operação de esvaziamento da pilha remove todos os elementos:
 - logo o tempo de execução é $O(n)$.


Obrigado pela atenção!!!
thiago.tavares@ifsuldeminas.edu.br





 ASCENCIO, A.; CAMPOS, E. de. *Fundamentos da programação de computadores: algoritmos, Pascal, C/C++ e Java*. Pearson Prentice Hall, 2008. ISBN 9788576051480. Disponível em: <<https://books.google.com.br/books?id=p-mTPgAACAAJ>>.


 C: A Reference Manual. Pearson Education, 2007. ISBN 9788131714409. Disponível em: <<https://books.google.com.br/books?id=Wt2NEypdGNIC>>.


 DAMAS, L. *LINGUAGEM C*. LTC. ISBN 9788521615194. Disponível em: <<https://books.google.com.br/books?id=22-vPgAACAAJ>>.

 FEOFILOFF, P. *Algoritmos Em Linguagem C*. CAMPUS - RJ, 2009. ISBN 9788535232493. Disponível em: <<http://books.google.com.br/books?id=LfUQai78VQgC>>.

 KERNIGHAN, B.; RITCHIE, D. *C: a linguagem de programação padrão ANSI*. Campus, 1989. ISBN 9788570015860. Disponível em: <<https://books.google.com.br/books?id=aVWrQwAACAAJ>>.

 LOPES, A.; GARCIA, G. *Introdução à programação: 500 algoritmos resolvidos*. Campus, 2002. ISBN 9788535210194. Disponível em: <<https://books.google.com.br/books?id=Rd-LPgAACAAJ>>.

 MIZRAHI, V. *Treinamento em linguagem C*. Pearson Prentice Hall, 2008. ISBN 9788576051916. Disponível em: <<https://books.google.com.br/books?id=7xt7PgAACAAJ>>.

 SCHILDT, H.; MAYER, R. *C completo e total*. Makron, 1997. ISBN 9788534605953. Disponível em: <<https://books.google.com.br/books?id=Pbl0AAAACAAJ>>.