Estrutura de Dados

Hamilton José Brumatto

Bacharelado em Ciências da Computação - UESC

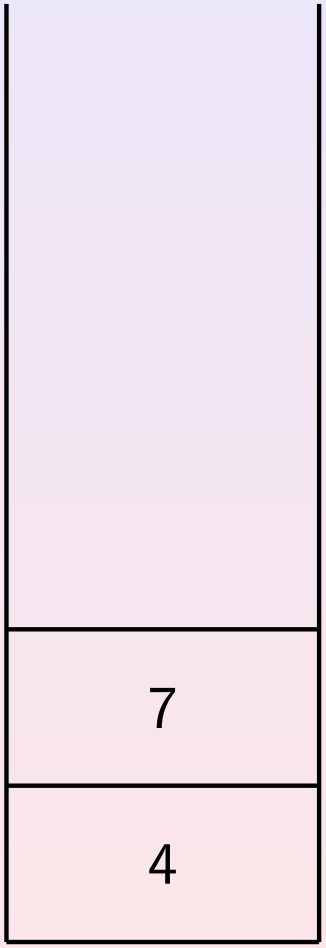
13 de maio de 2016

Pilhas

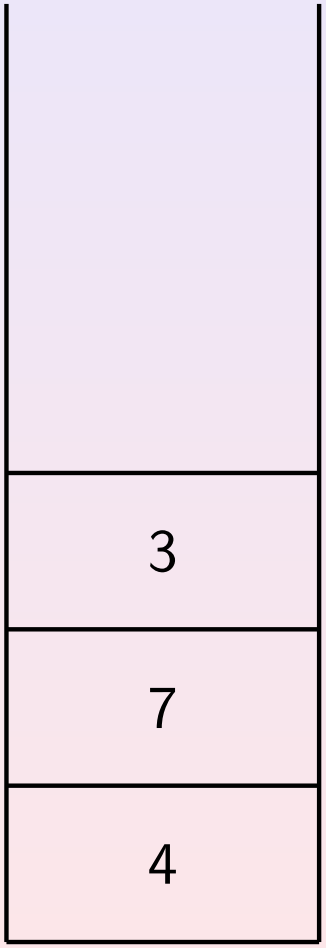
Pilha:

- Um conjunto ordenado de itens
- Novos itens podem ser inseridos ou excluídos em uma extremidade: *Topo*
- Pilha é uma estrutura de dados dinâmica: a operação de inserir e remover itens faz parte da estrutura.
- A representação de uma pilha é um sequência vertical de objetos, e o topo é o objeto mais acima.

Exemplo de Pilha e operações:

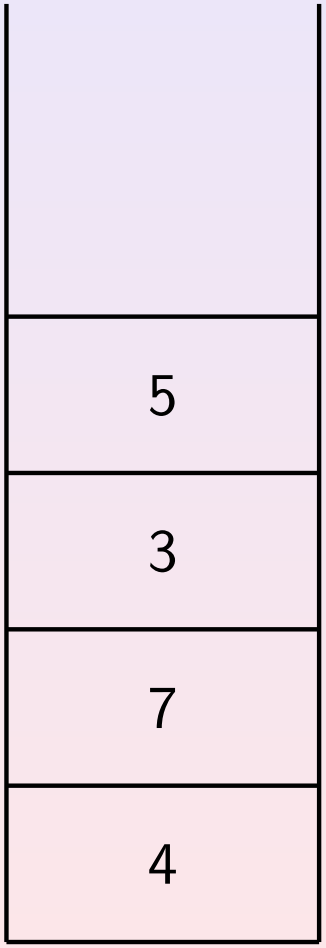


Exemplo de Pilha e operações:



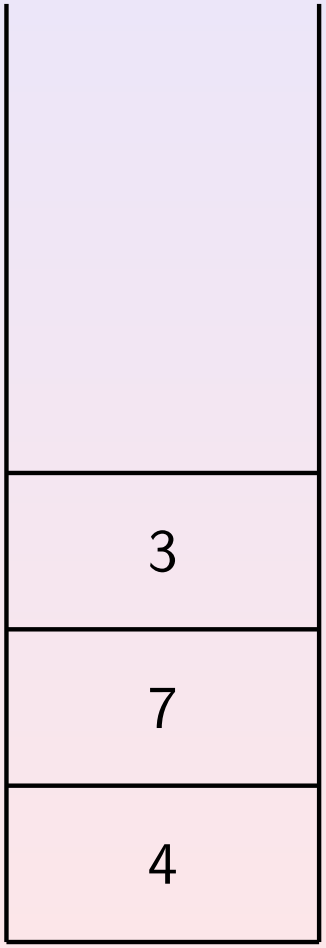
Inserindo 3

Exemplo de Pilha e operações:

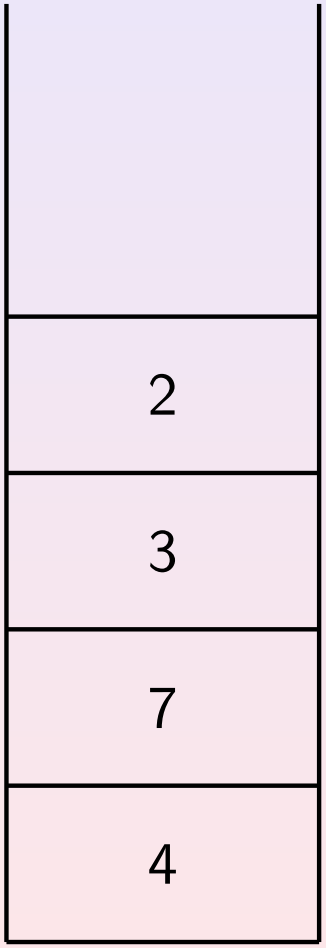


Inserindo 5

Exemplo de Pilha e operações:

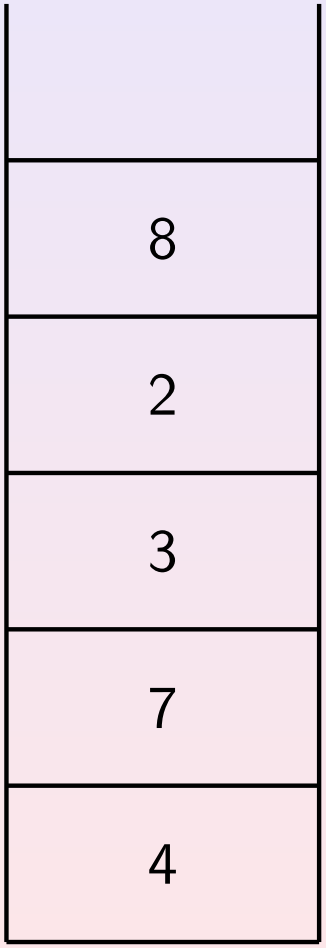


Exemplo de Pilha e operações:



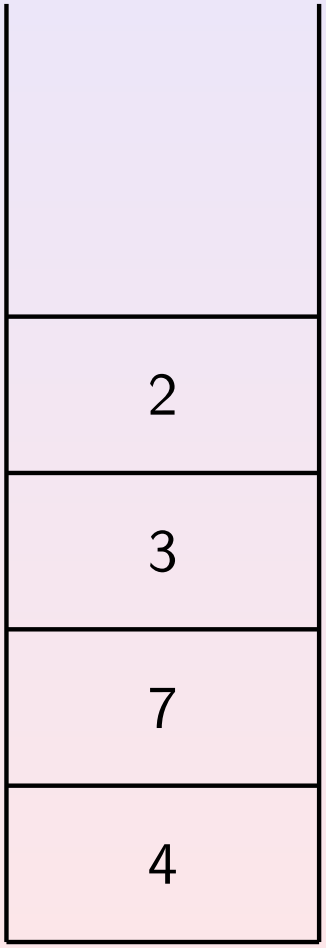
Inserindo 2

Exemplo de Pilha e operações:



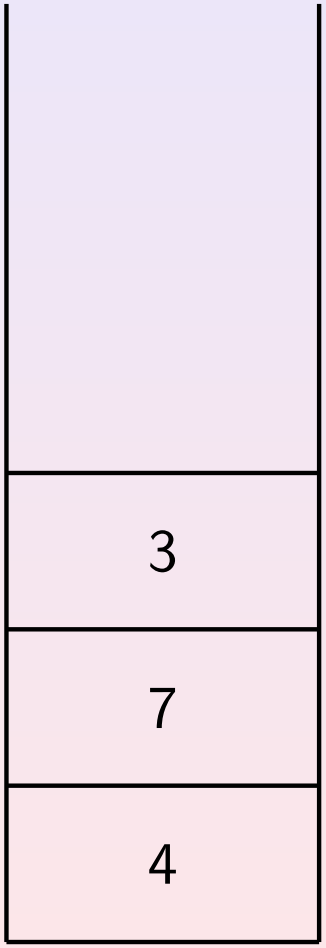
Inserindo 8

Exemplo de Pilha e operações:



Removendo

Exemplo de Pilha e operações:



Operando a Pilha

- A pilha somente mantém o estado de objetos contidos, e não o histórico de movimentos.
- Na animação dos slides anteriores, três estados são totalmente idênticos:
  - Após inserir o número 3.
  - Após remover, quando saiu o número 5.
  - E na última remoção, quando saiu o número 2.
- O que determina o estado da pilha é o conjunto de objetos lá existentes.
- A operação que remove um item não especifica qual item será removido, pois será o item que está no topo.

Empilhar / Desempilhar

- Duas operações são realizadas sobre a pilha:
  - A operação de inserir na pilha é: *Empilhar(Obj)*.
  - A operação de remover da pilha é: *Desempilhar()* $\rightarrow$ *Obj*
- Estas operações são conhecidas, também, como *Push(Obj)* e *Pop()*, respectivamente.
- Nos slides anteriores, a sequência de operações foram:
  Empilhar(3)
  Empilhar(5)
  Desempilhar() $\rightarrow$ 5
  Empilhar(2)
  Empilhar(8)
  Desempilhar() $\rightarrow$ 8
  Desempilhar() $\rightarrow$ 2

FILO ou LIFO

- Dada as operações sobre a pilha e como são inseridos os objetos, tem-se:
- O último objeto a ser inserido será o primeiro a ser removido.
- *LIFO* - Last In, First Out.
- Ou então, o primeiro objeto que foi inserido na pilha será o último a ser removido.
- *FILO* - First In, Last Out.

Chamadas de funções em um processo

- Em um programa em execução, em uma chamada de função/procedimento, o processo deve guardar as variáveis locais da função que estava rodando, e abrir espaço para variáveis locais da nova função.
- Como as chamadas podem ser recursivas, cada função chamada realiza este procedimento.
- As variáveis locais são recuperadas ao sair da função.
- Como as chamadas de funções são, LIFO, a pilha é a melhor opção para guardar as variáveis locais.

Um programa recursivo

```
1:  #include <stdio.h>
2:
3:  int fatorial(int n) {
4:    if(n==1) return 1;
5:    else return n*fatorial(n-1);
6:  }
7:
8:  int main(int c, char **args) {
9:    printf("Fatorial de 5: %d\n",fatorial(5));
10:  return 0;
11:}
```

Debugando o processo

```
Breakpoint 1, main (c=1, args=0x7fffffffdb18) at fatorial.c:9
9 printf("Fatorial de 5:  %d\n",fatorial(5));

(gdb)
```

Debugando o processo

```
Breakpoint 1, main (c=1, args=0x7fffffffdb18) at fatorial.c:9
9 printf("Fatorial de 5:  %d\n",fatorial(5));

(gdb) backtrace
#0 main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb)
```

Debugando o processo

```
Breakpoint 1, main (c=1, args=0x7fffffffdb18) at fatorial.c:9
9 printf("Fatorial de 5:  %d\n",fatorial(5));

(gdb) backtrace
#0 main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) s
fatorial (n=5) at fatorial.c:4
4 if(n==1) return 1;

(gdb)
```

Debugando o processo

```
Breakpoint 1, main (c=1, args=0x7fffffffdb18) at fatorial.c:9
9 printf("Fatorial de 5:  %d\n",fatorial(5));

(gdb) backtrace
#0 main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) s
fatorial (n=5) at fatorial.c:4
4 if(n==1) return 1;

(gdb) backtrace
#0 fatorial (n=5) at fatorial.c:4
#1 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb)
```

Debugando o processo

```
Breakpoint 1, main (c=1, args=0x7fffffffdb18) at fatorial.c:9
9 printf("Fatorial de 5:  %d\n",fatorial(5));

(gdb) backtrace
#0 main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) s
fatorial (n=5) at fatorial.c:4
4 if(n==1) return 1;

(gdb) backtrace
#0 fatorial (n=5) at fatorial.c:4
#1 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
5 else return n*fatorial(n-1);

(gdb)
```

Debugando o processo

```
(gdb) backtrace
#0 main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) s
fatorial (n=5) at fatorial.c:4
4 if(n==1) return 1;

(gdb) backtrace
#0 fatorial (n=5) at fatorial.c:4
#1 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
5 else return n*fatorial(n-1);

(gdb) s
fatorial (n=4) at fatorial.c:4
4 if(n==1) return 1;

(gdb)
```

Debugando o processo

```
(gdb) s
fatorial (n=5) at fatorial.c:4
4 if(n==1) return 1;

(gdb) backtrace
#0 fatorial (n=5) at fatorial.c:4
#1 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
5 else return n*fatorial(n-1);

(gdb) s
fatorial (n=4) at fatorial.c:4
4 if(n==1) return 1;

(gdb) backtrace
#0 fatorial (n=4) at fatorial.c:4
#1 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#2 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb)
```

Debugando o processo

```
(gdb) backtrace
#0 fatorial (n=5) at fatorial.c:4
#1 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
5 else return n*fatorial(n-1);

(gdb) s
fatorial (n=4) at fatorial.c:4
4 if(n==1) return 1;

(gdb) backtrace
#0 fatorial (n=4) at fatorial.c:4
#1 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#2 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
 else return n*fatorial(n-1);

(gdb)
```

Debugando o processo

```
(gdb) n
5 else return n*fatorial(n-1);

(gdb) s
fatorial (n=4) at fatorial.c:4
4 if(n==1) return 1;

(gdb) backtrace
#0 fatorial (n=4) at fatorial.c:4
#1 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#2 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
 else return n*fatorial(n-1);

(gdb) s
fatorial (n=3) at fatorial.c:4
4 if(n==1) return 1;

(gdb)
```

Debugando o processo

```
4 if(n==1) return 1;

(gdb) backtrace
#0 fatorial (n=4) at fatorial.c:4
#1 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#2 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
 else return n*fatorial(n-1);

(gdb) s
fatorial (n=3) at fatorial.c:4
4 if(n==1) return 1;

(gdb) backtrace
#0 fatorial (n=3) at fatorial.c:4
#1 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#3 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb)
```

Debugando o processo

```
#0 fatorial (n=4) at fatorial.c:4
#1 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#2 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
 else return n*fatorial(n-1);

(gdb) s
fatorial (n=3) at fatorial.c:4
4 if(n==1) return 1;

(gdb) backtrace
#0 fatorial (n=3) at fatorial.c:4
#1 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#3 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
5 else return n*fatorial(n-1);

(gdb)
```

Debugando o processo

```
(gdb) n
 else return n*fatorial(n-1);

(gdb) s
fatorial (n=3) at fatorial.c:4
4 if(n==1) return 1;

(gdb) backtrace
#0 fatorial (n=3) at fatorial.c:4
#1 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#3 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
5 else return n*fatorial(n-1);

(gdb) s
fatorial (n=2) at fatorial.c:4
4 if(n==1) return 1;

(gdb)
```

Debugando o processo

```
(gdb) backtrace
#0 fatorial (n=3) at fatorial.c:4
#1 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#3 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
5 else return n*fatorial(n-1);

(gdb) s
fatorial (n=2) at fatorial.c:4
4 if(n==1) return 1;

(gdb) backtrace
#0 fatorial (n=2) at fatorial.c:4
#1 0x0000000000400555 in fatorial (n=3) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#3 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#4 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb)
```

Debugando o processo

```
#2 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#3 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
5 else return n*fatorial(n-1);

(gdb) s
fatorial (n=2) at fatorial.c:4
4 if(n==1) return 1;

(gdb) backtrace
#0 fatorial (n=2) at fatorial.c:4
#1 0x0000000000400555 in fatorial (n=3) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#3 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#4 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
5 else return n*fatorial(n-1);

(gdb)
```

Debugando o processo

```
5 else return n*fatorial(n-1);

(gdb) s
fatorial (n=2) at fatorial.c:4
4 if(n==1) return 1;

(gdb) backtrace
#0 fatorial (n=2) at fatorial.c:4
#1 0x0000000000400555 in fatorial (n=3) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#3 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#4 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
5 else return n*fatorial(n-1);

(gdb) s
fatorial (n=1) at fatorial.c:4
4 if(n==1) return 1;

(gdb)
```

Debugando o processo

```
#1 0x0000000000400555 in fatorial (n=3) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#3 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#4 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
5 else return n*fatorial(n-1);

(gdb) s
fatorial (n=1) at fatorial.c:4
4 if(n==1) return 1;

(gdb) backtrace
#0 fatorial (n=1) at fatorial.c:4
#1 0x0000000000400555 in fatorial (n=2) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=3) at fatorial.c:5
#3 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#4 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#5 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb)
```

Debugando o processo

```
#4 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
5 else return n*fatorial(n-1);

(gdb) s
fatorial (n=1) at fatorial.c:4
4 if(n==1) return 1;

(gdb) backtrace
#0 fatorial (n=1) at fatorial.c:4
#1 0x0000000000400555 in fatorial (n=2) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=3) at fatorial.c:5
#3 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#4 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#5 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
6 }

(gdb)
```

Debugando o processo

```
(gdb) backtrace
#0 fatorial (n=1) at fatorial.c:4
#1 0x0000000000400555 in fatorial (n=2) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=3) at fatorial.c:5
#3 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#4 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#5 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
6 }

(gdb) backtrace
#0 fatorial (n=1) at fatorial.c:6
#1 0x0000000000400555 in fatorial (n=2) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=3) at fatorial.c:5
#3 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#4 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#5 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb)
```

Debugando o processo

```
#1 0x0000000000400555 in fatorial (n=2) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=3) at fatorial.c:5
#3 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#4 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#5 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
6 }

(gdb) backtrace
#0 fatorial (n=1) at fatorial.c:6
#1 0x0000000000400555 in fatorial (n=2) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=3) at fatorial.c:5
#3 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#4 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#5 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
6 }

(gdb)
```

Debugando o processo

```
6 }

(gdb) backtrace
#0 fatorial (n=1) at fatorial.c:6
#1 0x0000000000400555 in fatorial (n=2) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=3) at fatorial.c:5
#3 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#4 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#5 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
6 }

(gdb) backtrace
#0 fatorial (n=2) at fatorial.c:6
#1 0x0000000000400555 in fatorial (n=3) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#3 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#4 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb)
```

Debugando o processo

```
#0 fatorial (n=1) at fatorial.c:6
#1 0x0000000000400555 in fatorial (n=2) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=3) at fatorial.c:5
#3 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#4 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#5 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
6 }

(gdb) backtrace
#0 fatorial (n=2) at fatorial.c:6
#1 0x0000000000400555 in fatorial (n=3) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#3 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#4 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
6 }

(gdb)
```

Debugando o processo

```
(gdb) n
6 }

(gdb) backtrace
#0 fatorial (n=2) at fatorial.c:6
#1 0x0000000000400555 in fatorial (n=3) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#3 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#4 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
6 }

(gdb) backtrace
#0 fatorial (n=3) at fatorial.c:6
#1 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#3 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb)
```

Debugando o processo

```
(gdb) backtrace
#0 fatorial (n=2) at fatorial.c:6
#1 0x0000000000400555 in fatorial (n=3) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#3 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#4 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
6 }

(gdb) backtrace
#0 fatorial (n=3) at fatorial.c:6
#1 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#3 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
6 }

(gdb)
```

Debugando o processo

```
#3 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#4 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
6 }

(gdb) backtrace
#0 fatorial (n=3) at fatorial.c:6
#1 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#3 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
6 }

(gdb) backtrace
#0 fatorial (n=4) at fatorial.c:6
#1 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#2 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb)
```

Debugando o processo

```
(gdb) n
6 }

(gdb) backtrace
#0 fatorial (n=3) at fatorial.c:6
#1 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#3 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
6 }

(gdb) backtrace
#0 fatorial (n=4) at fatorial.c:6
#1 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#2 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
6 }

(gdb)
```

Debugando o processo

```
#0 fatorial (n=3) at fatorial.c:6
#1 0x0000000000400555 in fatorial (n=4) at fatorial.c:5
#2 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#3 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
6 }

(gdb) backtrace
#0 fatorial (n=4) at fatorial.c:6
#1 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#2 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
6 }

(gdb) backtrace
#0 fatorial (n=5) at fatorial.c:6
#1 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb)
```

Debugando o processo

```
#3 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
6 }

(gdb) backtrace
#0 fatorial (n=4) at fatorial.c:6
#1 0x0000000000400555 in fatorial (n=5) at fatorial.c:5
#2 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
6 }

(gdb) backtrace
#0 fatorial (n=5) at fatorial.c:6
#1 0x0000000000400574 in main (c=1, args=0x7fffffffdb18) at fatorial.c:9

(gdb) n
Fatorial de 5:  120
main (c=1, args=0x7fffffffdb18) at fatorial.c:10
10 return 0;
```

Análise da sintaxe de níveis de parêntesis

Considere a expressão:
$7 - ((X\dot{(}(X + Y)/(J - 3)) + Y)/(4 - 2,5))$

- Existe um número igual de parêntesis esquerdos e direitos.
- Todo parêntese da direita está precedido por um parêntese da esquerda correspondente.

Em uma análise se a expressão atende aos dois critérios podemos realizar uma análise com base em pilha.

Profundidade de Parêntesis

- Um parêntese de esquerda realiza uma abertura de nível.
- Um parêntese de direita realiza um fechamento de nível.
- A profundidade de nível em um determinado momento representa o número de parêntesis de esquerda abertos, mas ainda não fechados.
- O exemplo abaixo apresenta a contagem de nível para a expressão anterior

```
7 - ( ( X . ( ( X + Y ) / ( J - 3 ) ) + Y ) / ( 4 - 2,5 ) )
0 0 1 2 2 2 3 4 4  4  4 3 3 4 4 4 4 3 2 2  2 1 1 2 2 2  2  1 0
```

- A expressão pode ser mais complexa, com tipos diferentes de parêntesis: ([{...
- Neste caso, o fechamento de um nível deve ser correspondente ao mesmo símbolo que abriu, senão a expressão não é válida.

Análise de expressão com base em pilha

Contagem de parêntesis

valid = verdade;
p = pilha vazia;
enquanto (não terminou expressão) {
  simb ← pega o próximo símbolo da expressão;
  se (simb = '(' **ou** simb = '[' **ou** simb = '{')
    empilhar(p,simb);
  se (simb = ')' **ou** simb = ']' **ou** simb = '}')
    se(ehvazia(p)) valid = falso;
    senão {
      s = desempilhar(p);
      se(**não**(s complemento simb)) valid = falso;
    }
}
se(**não**(ehvazia(p))) valid = false;
se(valid) **imprime** "Expressão válida";
senão imprime "Expressão não válida";

Implementando Pilha

- A implementação de pilhas necessita de 2 operações primitvas e 1 de apoio:
  - A função *Empilhar(pilha,obj)*: Atualiza a posição do topo da pilha e insere um objeto no topo.
  - A função *Desempilhar(pilha)* $\rightarrow$ *obj*: Remove um objeto do topo da pilha, atualiza a posição do topo e retorna o objeto removido.
  - A função *ehVazia(pilha)* $\rightarrow$ *T/F*: Não é uma operação primitiva, mas auxilia retornando a informação sobre o status da pilha.
- A função *ehVazia* não é necessária, por exemplo, quando se utiliza um modelo de captura de exceções. Tentar uma operação Desempilhar em uma pilha vazia poderia gerar uma exceção.

Vetores

Estrutura e tipos para a pilha

```
#define TAMANHOPILHA 100
typedef int obj_t;
typedef int boolean;
enum{falso, verdade};

typedef struct pilha {
  obj_t itens[TAMANHOPILHA];
  int topo;
} pilha;

boolean empilhar(pilha *p, obj_t obj);
obj_t desempilhar(pilha *p);
boolean ehvazia(pilha p);
```

Operações: Empilhar

```
boolean empilhar(pilha *p, obj_t obj) {
  boolean ret = falso;
  if(p->topo < TAMANHOPILHA) {
    p->itens[p->topo++]=obj;
    ret = verdade;
  }
  return ret;
}
```

Operações: Desempilhar e ehVazia

```
obj_t desempilhar(pilha *p) {
  assert(p-> topo > 0);
  return p->itens[--p->topo];
}

boolean ehvazia(pilha p) {
  return (p.topo ? falso : verdade);
}
```

Testando

```
int main(int argc, char **args) {
 pilha p;
 p.topo = 0;
 obj_t o;
 empilhar(&p,5); status(p);
 empilhar(&p,3); status(p);
 empilhar(&p,2); status(p);
 o = desempilhar(&p); printf("%d -- > ",o); status(p);
 empilhar(&p,4); status(p);
 o = desempilhar(&p); printf("%d -- > ",o); status(p);
 o = desempilhar(&p); printf("%d -- > ",o); status(p);
 empilhar(&p,8); status(p);
 while(!ehvazia(p)) desempilhar(&p); status(p);
 return 0;
}
```

Testando

```
void status(pilha p) {
  int i=0;
  printf("%d : ",p.topo);
  for(i=0; i<p.topo; i++) printf("%d ",p.itens[i]);
  printf("\n");
  return;
}
```

Saída

```
1 :  5
2 :  5 3
3 :  5 3 2
2 --> 2 :  5 3
3 :  5 3 4
4 --> 2 :  5 3
3 --> 1 :  5
2 :  5 8
0 :
```

Implementando via Lista Ligada

- Lista Ligada oferece duas vantagens:
  - Não há limite de empilhamento (exceto o próprio limite de memória)
  - Não há a necessidade de uma variável para indicar o topo
- O topo é a própria cabeça da lista, inserimos e removemos na posição da cabeça.
- A pilha está vazia quando a cabeça é NULL

Tipos e Protótipos

```
typedef int obj_t;

typedef struct pilha {
  obj_t item;
  struct pilha *prox;
} pilha;

void empilhar(pilha **p, obj_t obj);
obj_t desempilhar(pilha **p);
```

Operações

```
void empilhar(pilha **p, obj_t obj) {
  pilha *np;
  np = malloc(sizeof(pilha));
  np->prox = (*p);
  np->item = obj;
  (*p) = np;
}

obj_t desempilhar(pilha **p) {
  obj_t o;
  pilha *d;
  assert((*p) != NULL);
  o = (*p)->item;
  d = (*p);
  (*p) = (*p)->prox;
  o = d->item;
  free(d);
  return o;
}
```

Testando

```
int main(int argc, char **args) {
 pilha *p;
 p = NULL;
 obj_t o;
 empilhar(&p,5); status(p);
 empilhar(&p,3); status(p);
 empilhar(&p,2); status(p);
 o = desempilhar(&p); printf("%d -- > ",o); status(p);
 empilhar(&p,4); status(p);
 o = desempilhar(&p); printf("%d -- > ",o); status(p);
 o = desempilhar(&p); printf("%d -- > ",o); status(p);
 empilhar(&p,8); status(p);
 while(p!=NULL) desempilhar(&p); status(p);
 return 0;
}
```

Testando

```
void status(pilha *p) {
  while(p != NULL) {
    printf("%d ",p- >item);
    p=p- >prox;
  }
  printf(".\n");
  return;
}
```

Saída

```
5 .
3 5 .
2 3 5 .
2 --> 3 5 .
4 3 5 .
4 --> 3 5 .
3 --> 5 .
8 5 .
.
```

Realizar a lista 5 de exercícios