Estrutura de Dados

Hamilton José Brumatto

Bacharelado em Ciências da Computação - UESC

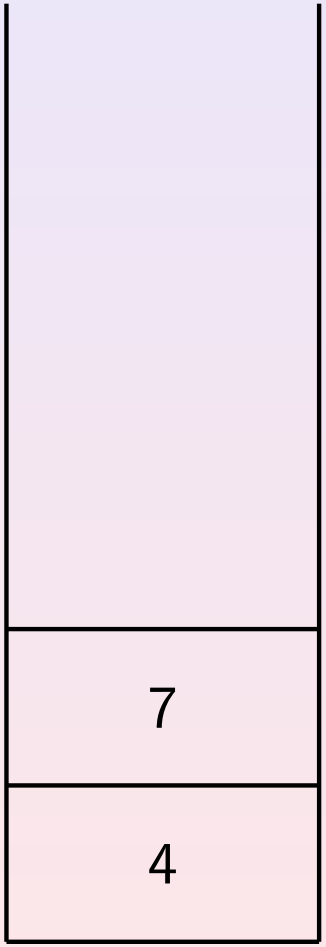
3 de fevereiro de 2024

Pilhas

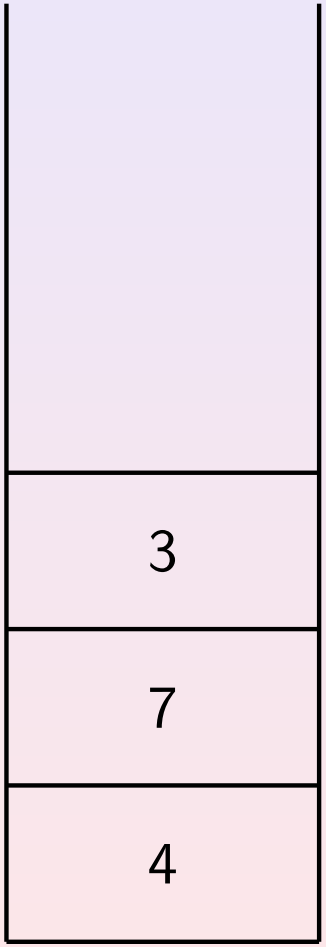
Pilha: "*stack*"

- 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:

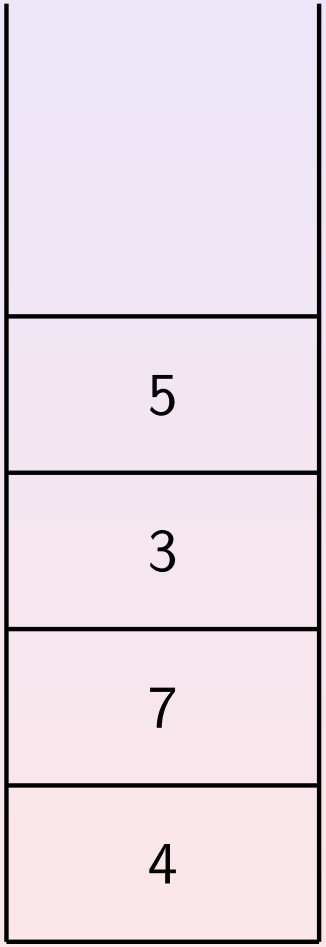


Exemplo de Pilha e operações:



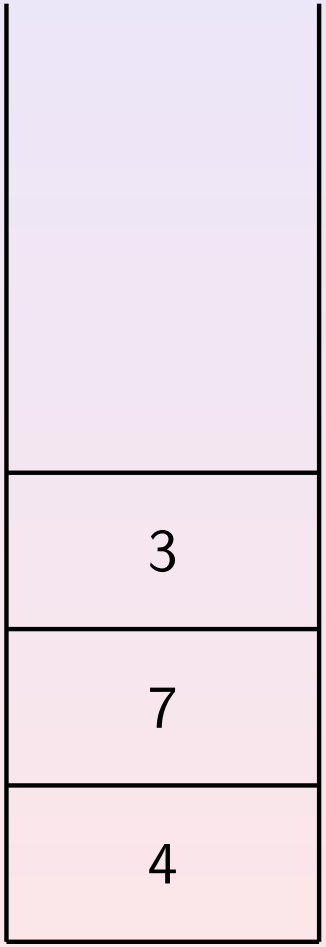
Inserindo 3

Exemplo de Pilha e operações:



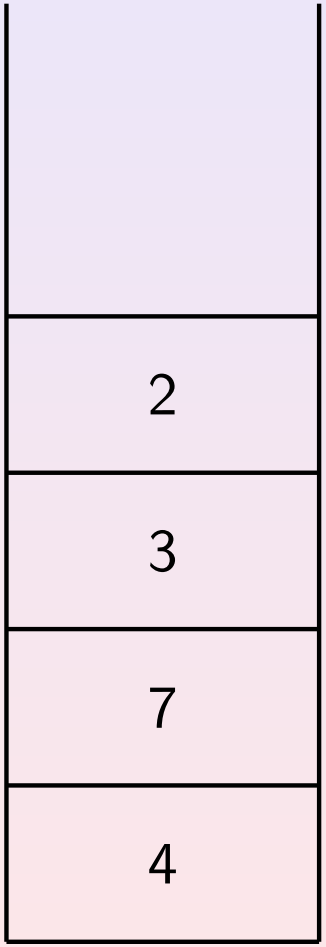
Inserindo 5

Exemplo de Pilha e operações:



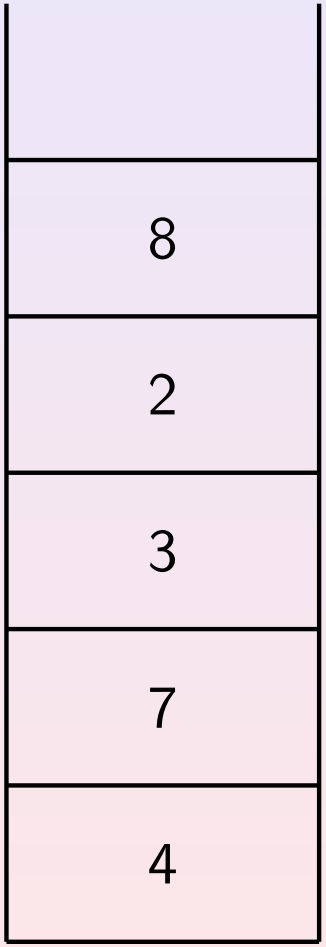
Removendo

Exemplo de Pilha e operações:



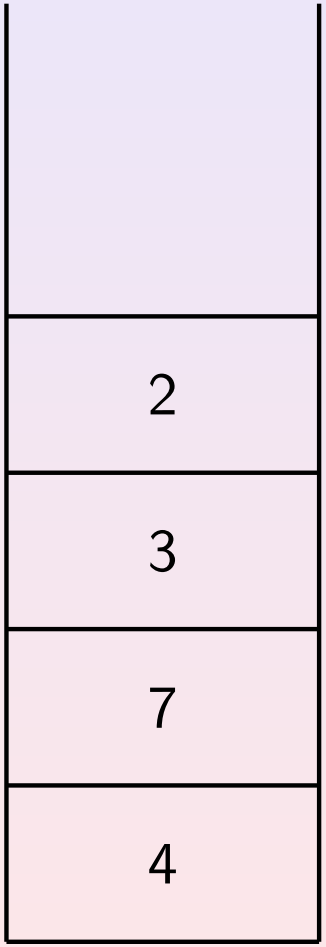
Inserindo 2

Exemplo de Pilha e operações:



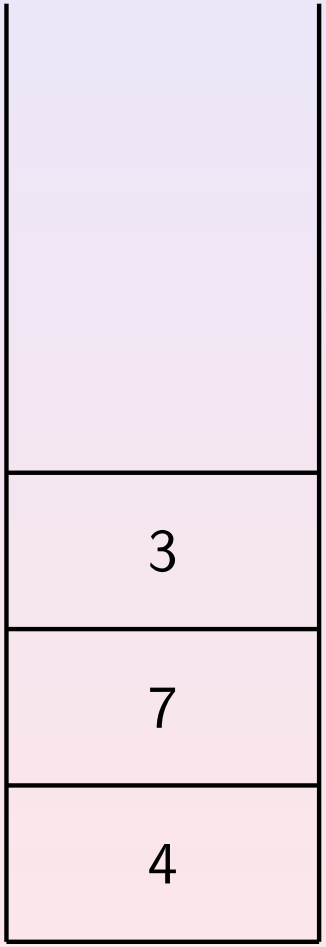
Inserindo 8

Exemplo de Pilha e operações:



Removendo

Exemplo de Pilha e operações:



Removendo

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

- Três operações são realizadas sobre a pilha:
  - A operação de inserir na pilha é: *eEmpilhar(Obj)*.
  - A operação de remover da pilha é: *desempilhar()*.
  - A operação que informa qual o valor do topo é: *topo()* $\rightarrow$ *obj*
- Estas operações são conhecidas, também, como *push(Obj)*, *pop()* e *obj top()*, respectivamente.
- Nos slides anteriores, a sequência de operações foram:
  empilhar(3)
  empilhar(5)
  desempilhar()
  empilhar(2)
  empilhar(8)
  desempilhar()
  desempilhar()

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.

Implementando Pilha

- A implementação de pilhas necessita de 2 operações primitvas e 21 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 *tamanho()* $\rightarrow$ *int*: Não é uma operação primitiva ou essencial, mas auxilia o usuário retornando a informação de quantos elementos há na pilha.

Implementação usando vetores

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

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

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

Operações: Empilhar

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

Operações: Desempilhar

```
boolean desempilhar(pilha *p) {
  boolean ret = falso;
  if(p->tamanho > 0) {
    p->tamanho--;
    ret= verdade;
  }
  return ret;
}
```

Operações: Topo, ehVazia e Tamanho

```
obj_t topo(pilha p) {
  obj_t val=0;
  if(p.tamanho > 0)
    val = p.itens[p.tamanho -1];
  return val;
}

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

int tamanho(pilha p) {
  return p.tamanho;
}
```

Testando

```
int main() {
 pilha p;
 p.tamanho = 0;
 obj_t o;
 empilhar(p,5); printf("%d --> %d\n",tamanho(p),topo(p));
 empilhar(p,4); printf("%d --> %d\n",tamanho(p),topo(p));
 empilhar(p,3); printf("%d --> %d\n",tamanho(p),topo(p));
 empilhar(p,2); printf("%d --> %d\n",tamanho(p),topo(p));
 empilhar(p,1); printf("%d --> %d\n",tamanho(p),topo(p));
 desempilhar(p); printf("%d --> %d\n",tamanho(p),topo(p));
 empilhar(p,7); printf("%d --> %d\n",tamanho(p),topo(p));
 empilhar(p,8); printf("%d --> %d\n",tamanho(p),topo(p));
 while(!ehvazia(p)) {
  desempilhar(p); printf("%d --> %d\n",tamanho(p),topo(p));
 } return 0;
}
```

Resultado

```
1 --> 5
2 --> 4
3 --> 3
4 --> 2
5 --> 1
4 --> 2
5 --> 7
5 --> 7
4 --> 2
3 --> 3
2 --> 4
1 --> 5
0 --> 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
- Como a pilha pode ser NULL, é importante que as funções retornem a atualização da cabeça da pilha.

Tipos e Protótipos

```
typedef int obj_t;
typedef int bool;
enum {false,true};

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

typedef struct pilha {
  item_pilha *cabeca;
} pilha;

void iniciar(pilha *p); pilha *empilhar(pilha *p, obj_t obj);
pilha *desempilhar(pilha *p);
obj_t topo(pilha *p);
bool ehvazia(pilha *p);
int tamanho(pilha *p);
```

Operações: Empilhar e Desempilhar

```
bool empilhar(pilha *p, obj_t obj) {
  bool res = false;
  item_pilha *cel = malloc(sizeof(item_pilha));
  if(cel != NULL)
    cel->item = obj;
    cel->prox = p;
    p->cabeca = cel;
    res = true;
  }
  return res;
}

bool desempilhar(pilha *p) {
  bool res = false;   item_pilha *cel = p-¿cabeca;
  if(cel) {
    p->cabeca = p->cabeca->prox;
    free(cel);
    res = true;
   }
  return res;
}
```

Operações: Iniciar, Topo, ehVazia e Tamanho

```
void iniciar(pilha *p) {
  p->cabeca = NULL;
}

obj_t topo(pilha *p) {
  obj_t val;
  if(p.cabeca) val = p.cabeca->item;
  return val;
}

bool ehvazia(pilha *p) {
  return (p.cabeca ? false : true);
}

int tamanho(pilha *p) {
  item_pilha *i = p.cabeca;
  int cont = 0;
  for(cont=0; i; cont++, i=i->prox);
  return cont;
}
```

Testando

```
int main(int argc, char **args) {
  pilha *p; iniciar(&p);
  empilhar(p,5); printf("%d -- > %d\n", tamanho(p), topo(p));
  empilhar(p,4); printf("%d -- > %d\n", tamanho(p), topo(p));
  empilhar(p,3); printf("%d -- > %d\n", tamanho(p), topo(p));
  empilhar(p,2); printf("%d -- > %d\n", tamanho(p), topo(p));
  desempilhar(p); printf("%d -- > %d\n", tamanho(p), topo(p));
  desempilhar(p); printf("%d -- > %d\n", tamanho(p), topo(p));
  empilhar(p,7); printf("%d -- > %d\n", tamanho(p), topo(p));
  empilhar(p,8); printf("%d -- > %d\n", tamanho(p), topo(p));
  while(!ehvazia(p)) {
    desempilhar(p); printf("%d -- > %d\n", tamanho(p), topo(p));
  } return 0;
}
```

Resultado

```
1 --> 5
2 --> 4
3 --> 3
4 --> 2
3 --> 3
2 --> 4
3 --> 7
4 --> 8
3 --> 7
2 --> 4
1 --> 5
0 --> 0
```

Recursividade e Pilhas

- Recursividade em computação é a capacidade de uma função de chamar a si própria.
- A recursividade precisa de um limite, pois se a função chamar a si própria indefinidamente, a computação não termina.
- O limite é chamado de base da recursividade. No início da função sempre se verifica se já terminou a recursão.
- A função recursiva que inclui a chamada recursiva é o passo da recursividade.
- Sim, funções recursivas são implementações de indução finita!
- Portanto, a chamada recursiva é a hipótese indutiva da recursividade.

Recursividade e Pilhas: Exemplo `fatorial(n)`

- Como trouxemos à tona o tema indução, vamos usar um exemplo baseado na indução: Fatorial.
- Base: $1! = 1$
- Hipótese da Indução: $(n-1)!$ é conhecido.
- Passo: $n! = n \cdot (n-1)!$

Recursividade e Pilhas: Exemplo `fatorial(n)`

- Base: $1! = 1$

Este é o limite da recursividade, a base, o teste que devemos fazer antes de continuar:

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

Recursividade e Pilhas: Exemplo `fatorial(n)`

- Passo: $n! = n \cdot (n-1)!$

O passo utiliza a hipótese de indução, ou seja, faz uma chamada recursiva:

```
return = n * fatorial(n-1);
```

Recursividade e Pilhas: Exemplo `fatorial(n)`

```
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:}
```

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.

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;
```

Pilhas simulando chamadas recursivas

- As chamadas recursivas são realizadas empilhando-se as variáveis locais.
- Podemos simular isto criando uma pilha e empilhando as "variáveis locais" até a base da recursividade.
- O passo ocorre desempilhando os valores.
- Na função fatorial, o valor empilhado era justamente n, o valor da chamada.
- Ao desempilhar, no passo, fazíamos $n * fat..$, ou seja, multiplicamos o valor empilhado com o fatorial já calculado.
- A base era 1, fatorial de $1 = 1$.

Pilhas simulando chamadas recursivas

```
int fatorial(int n) {
  pilha *p = NULL;
  int fat;

  while(n!=1) {
    p = empilhar(p,n);
    n--;
  }
  fat = 1; // 1!
  while(!ehvazia(p)) {
    fat *= topo(p);     p = desempilhar(p);
  }
  return fat;
}
```

Tudo é uma iteração!

- Em tempo! Toda chamada recursiva pode ser transformada em um loop iterativo, independente de pilhas.
- Um loop (for ou while), é também uma indução finita.
- Mas este tema não vamos explorar aqui, fica para outra oportunidade.
- Apreciem, apenas, a versão iterativa do fatorial.

Fatorial: versão iterativa

```
int fatorial(int n) {
  int i, fat = 1;
  for (i = 2; i <= n; i++)
    fat = fat * i;
  return fat;
}
```