

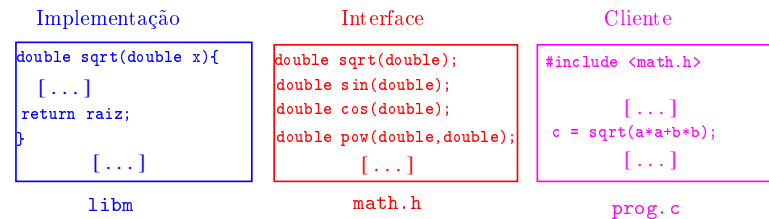
Melhores momentos

Interfaces

Uma **interface** (= *interface*) é uma fronteira entre entre a **implementação** de um biblioteca e o **programa que usa** a biblioteca.

Um **cliente** (= *client*) é um programa que chama alguma função da biblioteca.

AULA 12



Interfaces

Não sabemos como a pilha está implementada.

```
char *infixaParaPosfixa(char *inf) {
  [ ... ]
  stackInit(n) /* inicializa a pilha */

  [ ... ]
  stackPush(inf[i]);
  while((x = stackPop()) != '(')
    posf[j++] = x;
  break;
```

```
[ ... ]
  while (!stackEmpty()
    && (x = stackTop()) != '(')
    posf[j++] = stackPop();
  stackPush(inf[i]);
[ ... ]
  while (!stackEmpty())
    posf[j++] = stackPop()
posf[j] = '\0'; /* fim expr polonesa */
stackFree();
```

Pausa para nossos comerciais

O **Vinicius** dará uma aula amanhã sobre ferramentas para revelar **memory leaking** e **outros erros** em nossos programas.

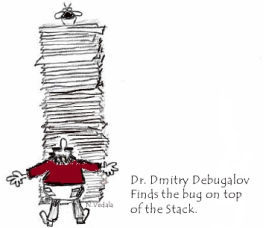
Quando: terça-feira, 30/SET

Horário: das 12h às 13h

Onde: auditório do CCSL

AULA 13

Pilha implementada em lista encadeada



Fonte: <http://www.dumpanalysis.org/>

PF 6.3, S 4.4

<http://www.ime.usp.br/~pf/algoritmos/aulas/pilha.html>

Interface stack.h

```
/*
 * stack.h
 * INTERFACE: funcoes para manipular uma
 * pilha
 */
void stackInit(int);
int stackEmpty();
void stackPush(Item);
Item stackPop();
Item stackTop();
void stackFree();
void stackDump();
```

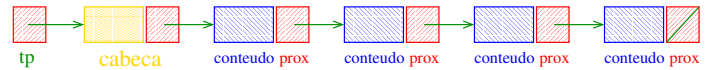
Implementação stack.c

```
void
stackInit(int n)
{
    tp = mallocSafe(sizeof *tp);
    tp->prox = NULL;
}

int
stackEmpty()
{
    return tp->prox == NULL;
}
```

Pilha implementada em uma lista encadeada

A pilha será armazenada em uma **lista encadeada** com **cabeça**.



O ponteiro **tp** aponta para a **cabeça** da lista.

tp->prox->conteudo é o elemento do **topo** da pilha.

A pilha está **vazia** se "**tp->prox == NULL**".

A pilha está **cheia** se ... acabou a memória disponível.

Implementação stack.c

```
#include <stdlib.h>
#include <stdio.h>
#include "item.h"
/*
 * PILHA: implementacao em lista encadeada
 */
typedef struct stackNode* Link;
struct stackNode{
    Item conteudo;
    Link prox;
};
static Link tp;
```

Implementação stack.c

```
void
stackPush(Item item)
{
    Link nova = mallocSafe(sizeof *nova);

    nova->conteudo = item;
    nova->prox = tp->prox;
    tp->prox = nova;
}
```


PilhaS implementadaS em lista encadeada



Fonte: <http://www.dumpanalysis.org/>

PF 6.3, S 4.4

<http://www.ime.usp.br/~pf/algoritmos/aulas/pilha.html>

PilhaS implementadaS em listaS encadeadaS

Para **cada pilha** há um ponteiro **tp** para a lista.

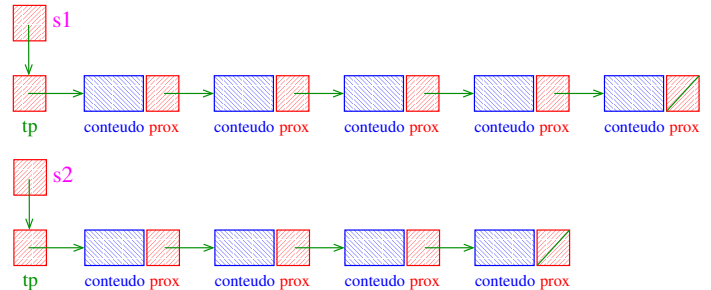
tp->conteudo é o elemento do **topo** da pilha.

Uma pilha **s** está **vazia** se "**s->tp == NULL**".

Uma pilha está **cheia** se ... acabou a memória disponível.

PilhaS implementadaS em listaS encadeadaS

As pilhas serão armazenada em **listaS encadeadaS** sem **cabeça**.



Interface stack.h

```
/*
 * stack.h
 * INTERFACE: funcoes para manipular uma
 * pilha
 */
typedef struct stack *Stack;
Stack stackInit(int);
int stackEmpty(Stack);
void stackPush(Stack, Item);
Item stackPop(Stack);
Item stackTop(Stack);
void stackFree(Stack);
void stackDump(Stack);
```

Infixa para posfixa novamente...

Recebe uma expressão infixa **inf** e devolve a correspondente expressão **posfixa**.

```
char *infixaParaPosfixa(char *inf) {
    char *posf; /* expressao polonesa */
    int n = strlen(inf);
    int i; /* percorre infixa */
    int j; /* percorre posfixa */
    Stack s; /* PILHA */

    /*aloca area para expressao polonesa*/
    posf = mallocSafe((n+1)*sizeof(char));
    /* 0 '+1' eh para o '\0' */
```

case '('

```
s = stackInit(n) /* inicializa a pilha */

/* examina cada item da infixa */
for (i = j = 0; i < n; i++) {
    switch (inf[i]) {
        char x; /* item do topo da pilha */
        case '(':
            stackPush(s, inf[i]);
            break;
```

case ')':

```
case ')':
    while((x = stackPop(s)) != '(')
        posf[j++] = x;
    break;
```

case '+', case '-':

```
case '*':
case '/':
    while (!stackEmpty(s)
        && (x = stackTop(s)) != '(')
        posf[j++] = stackPop(s);
    stackPush(s, inf[i]);
    break;
```

case '*', case '/':

```
case '*':
case '/':
    while (!stackEmpty()
        && (x = stackTop(s)) != '('
        && x != '+' && x != '-')
        posf[j++] = stackPop(s);
    stackPush(s, inf[i]);
    break;
```

default

```
default:
    if(inf[i] != ' ')
        posf[j++] = inf[i];
} /* fim switch */
} /* fim for (i=j=0...) */
```

Finalizações

```
/* desempilha todos os operandos que
   restaram */
while (!stackEmpty(s))
    posf[j++] = stackPop(s)
posf[j] = '\0'; /* fim expr polonesa */
stackFree(s);
return posf;
} /* fim funcao */
```

Implementação stack.c

```
#include "item.h"
/* PILHA: implementacao em lista encadeada
   */
typedef struct stackNode* Link;
struct stackNode{
    Item conteudo;
    Link prox;
};
struct stack {
    Link tp;
};
typedef struct stack *Stack;
```