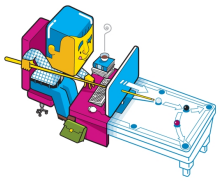
AULA 12

Interfaces



Fonte: http://allfacebook.com/

Before I built a wall I'd ask to know
What I was walling in or walling out,
And to whom I was like to give offence.
Something there is that doesn't love a wall,
That wants it down.

Robert Frost, *Mending Wall*

The Practice of Programming
B.W.Kernigham e R. Pike

S 3.1, 4.2, 4.3, 4.4

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.

Implementação

```
double sqrt(double x){
  [...]
return raiz;
}
      [...]
```

libm

Interface

```
double sqrt(double);
double sin(double);
double cos(double);
double pow(double,double);
      [...]
```

math.h

Cliente

```
#include <math.h>
      [...]
 c = sqrt(a*a+b*b);
      [...]
```

prog.c

Interfaces

Para cada função na biblioteca o cliente precisa saber

- o seu nome, os seus argumentos e os tipos desses argumentos;
- o tipo do resultado que é retornado.

Só a quem implementa interessa os detalhes de implementação.

Implementação	Interface	Cliente
Responsável por como as funções funcionam	Os dois lados concordam sobre os protótipos das funções	Responsável por como usar as funções
lib	xxx.h	yyy.c

Interfaces

Entre as decisões de projeto estão

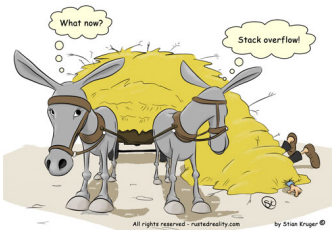
Interface: quais serviços serão oferecidos? A interface é um "contrato" entre o usuário e o projetista.

Ocultação: qual informação é visível e qual é privada? Uma interface deve prover acesso aos componente enquanto esconde detalhes de implementação que podem ser alterados sem afetar o usuário.

Recursos: quem é responsável pelo gerenciamento de memória e outros recursos?

Erros: quem detecta e reporta erros e como?

Interfaces para pilhas



Fonte: http://rustedreality.com/stack-overflow/

S 3.1, 4.2, 4.3, 4.4

Interface item.h

```
/* item.h */
typedef char Item;
```

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();
```

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 */

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

case '('

```
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(inf[i]);
        break;
```

case ')'

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

case '+', case '-'

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

case '*', case '/'

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

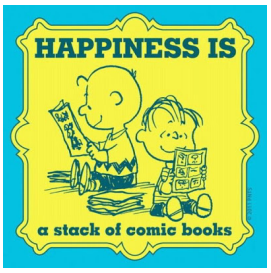
default

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

Finalizações

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

Pilha implementada em um vetor



Fonte: http://powsley.blogspot.com.br/

PF 6.1, S 4.4
http://www.ime.usp.br/~pf/algoritmos/aulas/pilha.html

Implementação stack.c

```
#include <stdlib.h>
#include <stdio.h>
#include "item.h"
/*
 * PILHA: implementacao em vetor
 */
static Item *s; /* pilha */
static int t;
/* t eh o indice do topo da pilha, s[t]
 * eh a 1a.  posicao vaga da pilha
 */
```

Implementação stack.c

```
void
stackInit(int n)
{
  s = (Item*) malloc(n*sizeof(Item));
  t = 0;
}

int
stackEmpty()
{
  return t == 0;
}
```

Implementação stack.c

```
void
stackPush(Item item)
{
   s[t++] = item;
}

Item
stackPop()
{
   return s[--t];
}
```

Implementação stack.c

```
Item
stackTop()
{
   return s[t-1];
}

void
stackFree()
{
   free(s);
}
```

Implementação stack.c

```
void
stackDump()
{
   int k;

   fprintf(stdout,"pilha :  ");
   if (t == 0) fprintf(stdout, "vazia.");
   for (k = t-1; k >= 0; k--)
       fprintf(stdout, "%c ", s[k]);
   fprintf(stdout,"\n");
}
```

Compilação

```
cria o obj stack.o
> gcc -Wall -O2 -ansi -pedantic
-Wno-unused-result -c stack.c

cria o obj polonesa.o
> gcc -Wall -O2 -ansi -pedantic
-Wno-unused-result
   -c polonesa.c

cria o executável polonesa
> gcc stack.o polonesa.o -o polonesa
```

Makefile

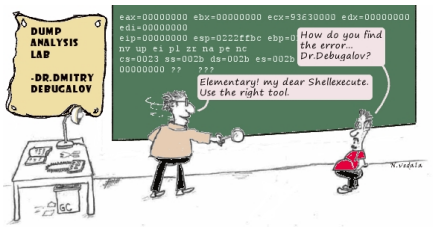
Hmmm. Ler o tópico Makefile no fórum.

```
polonesa:  polonesa.o stack.o
       gcc polonesa.o stack.o -o polonesa

polonesa.o:  polonesa.c stack.h
       gcc -Wall -O2 -ansi -pedantic \
       -Wno-unused-result -c polonesa.c

stack.o:  stack.c stack.h item.h
       gcc -Wall -O2 -ansi -pedantic \
       -Wno-unused-result -c stack.c
```

Leiaute da memória de um programa em C

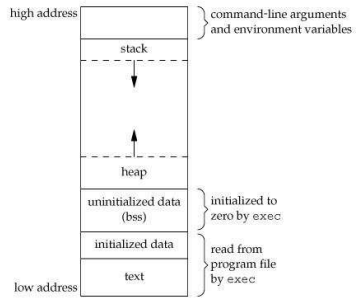


Fonte: http://meta.stackexchange.com/

http://www.geeksforgeeks.org/memory-layout-of-c-program/
http://cs-fundamentals.com/c-programming/

Leiaute da memória

Representação típica da memória utilizada por um programa.



Fonte: http://cs-fundamentals.com/c-programming/

Leiaute da memória

A memória utilizada por um programa em partes chamadas de segmentos (**memory segments**):

- *stack*: variáveis locais, parâmetros das funções, endereços de retorno;
- *heap*: para alocação de memória dinamicamente, administrada por `malloc()` `free()` e seus amigos;
- BSS: variáveis estáticas (*static*) e globais não inicializadas (`static int i;`);
- *data*: variáveis estáticas (*static*) inicializadas (`static int i = 0;`);
- *text*: código do programa.

Size

```
meu_prompr> size polonesa
text   data   bss   dec    hex   filename
3392   316    56    3764   eb4   polonesa
```