TAD: Tipo Abstrato de Dados

Marcio Santi

Linguagem C++

Módulos e Compilação em Separado

- Um programa em linguagem C pode ser dividido em vários arquivos fontes com extensão C;
- Módulo: arquivo que implementa funções que representam apenas parte da implementação do programa, ou seja um programa pode ser composto por um ou mais módulos;
- Cada módulo é compilado separadamente, gerando, para cada módulo, um arquivo objeto;
- Após a compilação individual de cada módulo, esses objetos são *ligados* ou ***linkados*** pelo *ligador* ou ***link-editor*** ;

Exemplo

str.h

```
int comp(char* s);
void cop(char* d, char* o);
void conc(char* d, char* o);
```

str.c

```
/* implementação das fçs
```

prg.c

```
#include <stdio.h>
#include "str.h"
int main( void )
{
   // programa que usa as fçs
}
```

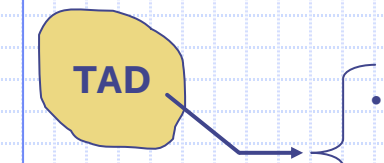
```
> gcc –c str.c
> gcc –c prg.c
> gcc –o prg.exe str.o prg.o
```

- O mesmo arquivo str.c pode ser utilizado por outros programas ou módulos de programação que queiram utilizar funções utilitárias para a manipulação de strings

Tipos Abstratos de Dados

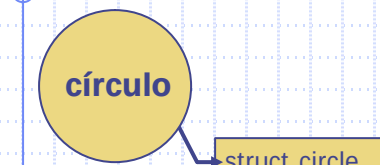
- Em geral um módulo agrupa vários tipos e funções com funcionalidades relacionadas caracterizando uma finalidade bem definida
- Quando um módulo define um novo tipo de dado e o conjunto de funções para manipular dados desse tipo, diz-se que esse módulo representa um TAD (tipo abstrato de dados)
- Abstrato nesse contexto quer dizer: "esquecida a forma de implementação".
- Assim, um TAD é definido pela finalidade do tipo e das operações associadas a este tipo (funções que manipulam variáveis do tipo) e não pela forma como está implementado.

Tipos Abstratos de Dados



- **atributos**
  - dados que o representam
- **interface**
  - operações associadas
  - como é manipulado

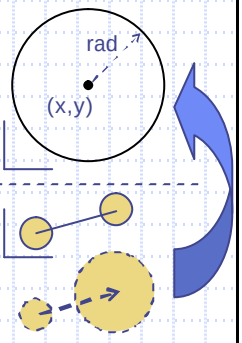
Exemplo: Editor Gráfico (classe círculo)




```
struct circle
{
   double x, y;
   double rad;
   void translation( );
   void scale( );
   double area( );
   double perim( );
}
```

atributos

interface



Exemplo: *stack*

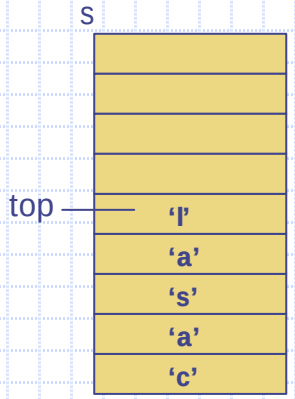
push: insere um elemento na pilha

pop: retira um elemento na pilha

FILO: first in, last out

```
struct stack
{
    int top;
    char s[max_len];
}
```

Sempre no topo



Exemplo: pilha de inteiros (*header file*)

stk.h

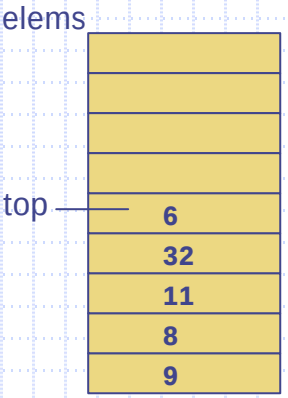
```
#ifndef stack_h
#define stack_h

#define MAX 50

struct Stack {
  int top;
  int elems[MAX];
};

void push(struct Stack * s, int i);
int  pop(struct Stack * s);
int  empty (struct Stack * s);
struct Stack *  createStack(void);

#endif
```



Exemplo: pilha de inteiros (*source code*)

stk.c

```
#include <stdlib.h>
#include "stack-c.h"
void push(struct Stack*s, int i) { s->elems[s->top++] = i; }
int  pop(struct Stack*s)         { return s->elems[--(s->top)]; }
int  empty(struct Stack*s)       { return s->top == 0; }

struct Stack* createStack(void)
{
   struct Stack* s = (struct  Stack*)malloc(sizeof(struct Stack));
   s->top = 0;
   return s;
}
```

Cliente: Calculadora RPN

rpn.c

```
#include <stdlib.h>
#include <stdio.h>
#include "stack-c.h"
int getop(struct Stack* s, int* n1, int* n2)
{
   if (empty(s))  {
      printf("empty stack!\n");
      return 0;
   }
   *n2 = pop(s);
   if (empty(s))  {
      push(s, *n2);
      printf("two operands needed!\n");
      return 0;
   }
   *n1 = pop(s);
   return 1;
}
// continua na próxima transparência
```

Cliente: Calculadora RPN

rpn.c

```
int main(void)
{
  struct Stack* s = createStack();
  while (1)
  {
    char str[31];
    int i;
    printf("> ");
    gets(str);
    if (sscanf(str, " %d", &i)==1)push(s, i);
    else
    {
      int n1, n2;
      char c;
      sscanf(str, "%c", &c);
      switch(c) {
      }
    }
  }
}
```

```
{  // implementação do switch
      case '+':
        if (getop(s, &n1, &n2))push(s, n1+n2);
        break;
      case '-':
        if (getop(s, &n1, &n2))push(s, n1-n2);
        break;
      case '/':
        if (getop(s, &n1, &n2))push(s, n1/n2);
        break;
      case '*':
        if (getop(s, &n1, &n2))push(s, n1*n2);
        break;
      case 'q':
        return 0;
      default:
        printf("error\n");
}
```

```
{ // para imprimir a pilha
    int i;
    for (i=0; i<s->top; i++)
      printf("%d:%6d\n", i, s->elems[i]);
  }
```

Exercício

- Implementar, em linguagem C, a calculadora RPN apresentada anteriormente utilizando-se dos conceitos de:
  - programação modularizada
  - tipo abstratos de dados
  - Conceito cliente / servidor