

Visualizando a memória:

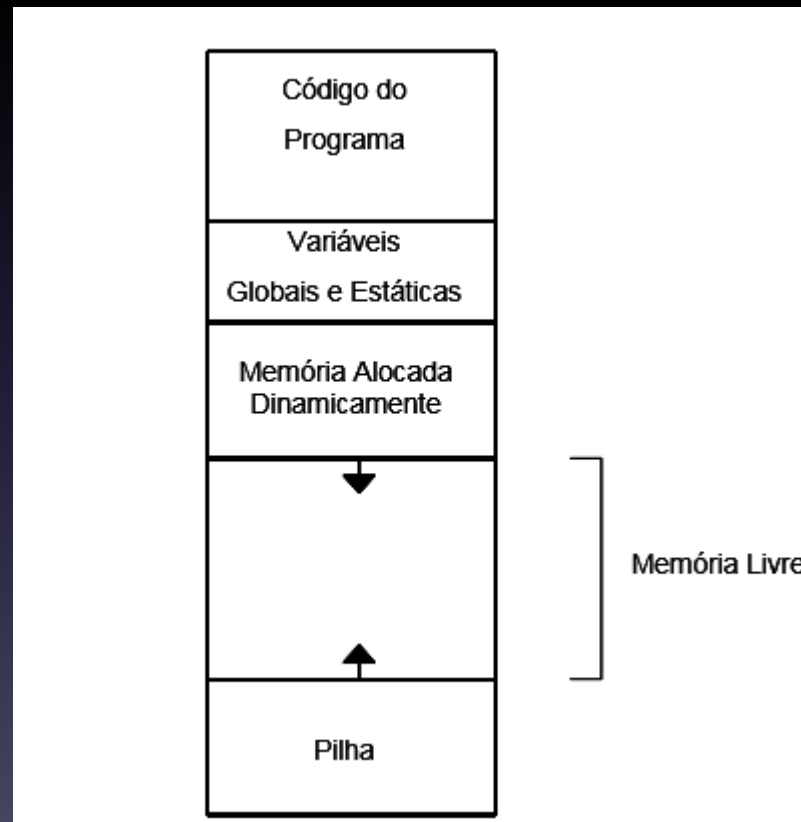
Depurando programas em C

Projetando tipos abstratos de dados

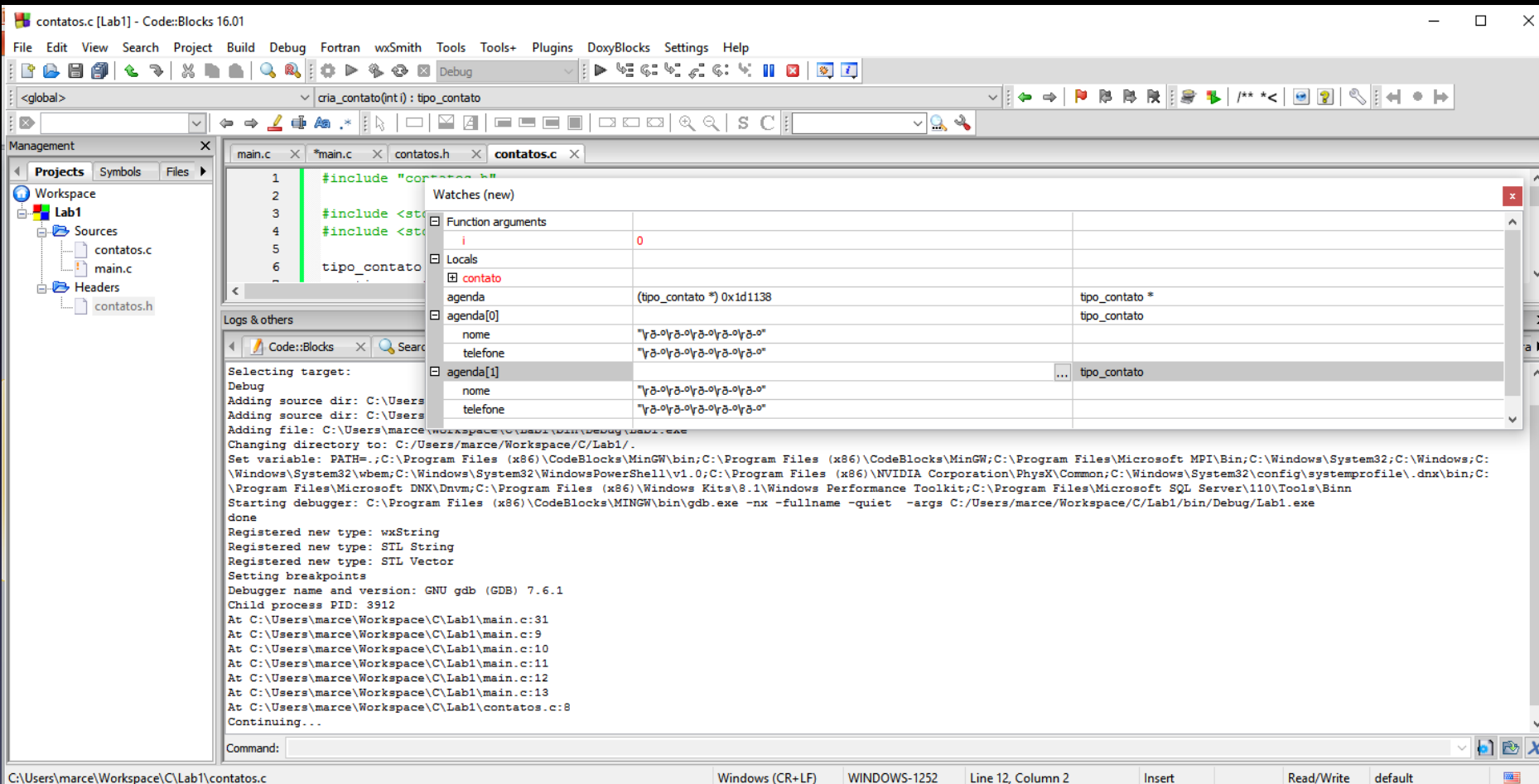
Marcelo Maia

2016

Organização esquemática da memória



Mostrando o depurador



C:\Users\marce\Workspace\C\Lab1\contatos.c

Windows (CR+LF)

WINDOWS-1252

Line 12, Column 2

Insert

Read/Write

default



TAD: Ponto

```
/* TAD: Ponto (x,y) */
/* Tipo exportado */
typedef struct ponto Ponto;
/* Funções exportadas */
/* Função cria
** Aloca e retorna um ponto com coordenadas (x,y)
** */
Ponto* cria_Ponto (float x, float y);
/* Função libera
** Libera a memória de um ponto previamente criado.
** */
void libera_Ponto (Ponto* p);
/* Função acessa
** Devolve os valores das coordenadas de um ponto
** */
void acessa_Ponto (Ponto* p, float* x, float* y);
/* Função atribui
** Atribui novos valores às coordenadas de um ponto
** */
void atribui_Ponto (Ponto* p, float x, float y);
/* Função distancia
** Retorna a distância entre dois pontos
** */
float distancia_Pontos (Ponto* p1, Ponto* p2);
```

Usando o TAD Ponto

```
#include <stdio.h>
#include "ponto.h"

int main (void)
{
    Ponto* p = pto_cria(2.0,1.0);
    Ponto* q = pto_cria(3.4,2.1);

    float d = pto_distancia(p,q);

    printf("Distancia entre pontos: %f\n",d);

    pto_libera(q);
    pto_libera(p);

    return 0;
}
```

Implementando Ponto

```
#include <stdlib.h> /* malloc, free, exit */
#include <stdio.h> /* printf */
#include <math.h> /* sqrt */
#include "ponto.h"
```

```
struct ponto {
    float x;
    float y;
};
```

```
/* TAD: Ponto (x,y) */

/* Tipo exportado */
typedef struct ponto Ponto;

/* Funções exportadas */

/* Função cria
```

Criando e Destruindo

```
Ponto* cria_Ponto (float x, float y) {
    Ponto* p = (Ponto*) malloc(sizeof(Ponto));
    if (p == NULL) {
        printf("Memória insuficiente!\n");
        exit(1);
    }
    p->x = x;
    p->y = y;
    return p;
}

void libera_Ponto (Ponto* p) {
    free(p);
}
```

Acessando

```
void acessa_Ponto (Ponto* p, float* x, float* y) {
    *x = p->x;
    *y = p->y;
}

void atribui_Ponto (Ponto* p, float x, float y) {
    p->x = x;
    p->y = y;
}

float distancia_Pontos (Ponto* p1, Ponto* p2) {
    float dx = p2->x - p1->x;
    float dy = p2->y - p1->y;
    return sqrt(dx*dx + dy*dy);
}
```



```
#ifndef PONTO_H
#define PONTO_H
...
#endif
```

/ código da interface do módulo */*

```
#ifndef PONTO_H_INCLUDED
#define PONTO_H_INCLUDED

/* TAD: Ponto (x,y) */
/* Tipo exportado */
typedef struct ponto Ponto;
/* Funções exportadas */
+ /* Função cria
Ponto* cria_Ponto (float x, float y);
+ /* Função libera
void libera_Ponto (Ponto* p);
+ /* Função acessa
void acessa_Ponto (Ponto* p, float* x, float* y);
+ /* Função atribui
void atribui_Ponto (Ponto* p, float x, float y);
+ /* Função distancia
float distancia_Pontos (Ponto* p1, Ponto* p2);

#endif // PONTO_H_INCLUDED
```

Quais outras funções para Ponto?

- Como fazer a assinatura?
 - Qual nome escolher?
 - Como definir os parâmetros?

```

/* TAD: Círculo */

/* Dependência de módulos */
#include "ponto.h"

/* Tipo exportado */
typedef struct circulo Circulo;

/* Funções exportadas */
/* Função cria
** Aloca e retorna um círculo com centro (x,y) e raio r
*/
Circulo* circ_cria (float x, float y, float r);

/* Função libera
** Libera a memória de um círculo previamente criado
*/
void circ_libera (Circulo* c);

/* Função area
** Retorna o valor da área do círculo
*/
float circ_area (Circulo* c);

/* Função interior
** Verifica se um dado ponto p está dentro do círculo
*/
int circ_interior (Circulo* c, Ponto* p);

```

Implemente Circulo.c

```
#ifndef CIRCULO_H_INCLUDED
#define CIRCULO_H_INCLUDED
#include "Ponto.h"

typedef struct circulo Circulo;

Circulo* cria_Circulo(float x, float y, float r);

void libera_Circulo (Circulo *c);
float area_Circulo (Circulo *c);
int interior_Circulo (Circulo *c, Ponto *p);

#endif // CIRCULO_H_INCLUDED
```

Circulo.c

```
#include <stdlib.h>
#include "circulo.h"

#define PI 3.14159

struct circulo {
    Ponto* p;
    float r;
};
```

```
/* TAD: Circulo */
```

```
/* Dependência de módulos */
#include "ponto.h"
```

```
/* Tipo exportado */
typedef struct circulo Circulo;
```

```

Circulo* circ_cria (float x, float y, float r)
{
    Circulo* c = (Circulo*)malloc(sizeof(Circulo));
    c->p = pto_cria(x,y);
    c->r = r;
    return c;
}


void circ_libera (Circulo* c)
{
    pto_libera(c->p);
    free(c);
}

float circ_area (Circulo* c)
{
    return PI*c->r*c->r;
}

int circ_interior (Circulo* c, Ponto* p)
{
    float d = pto_distancia(c->p,p);
    return (d<c->r);
}

```

Análise Crítica do Exercício em Laboratório

- Opções de criação do tipo abstrato de dados
 - Abstração principal
 - Contato 
 - Lista de elementos
 - Como é o reuso promovido com as opções acima?

```
4 typedef struct {
5     char nome[20];
6     char telefone[20];
7 } contato;
8
9 contato *agenda;
10
11 int cria_agenda () {
12     int nro_contatos, i;
13     printf("Quantos contatos quer cadastrar? ");
14     scanf("%d", &nro_contatos);
15     agenda = malloc(nro_contatos*sizeof(contato));
16     for (i=0; i<nro_contatos; i++) {
17         printf("Entre com o nome %d:", i+1);
18         scanf("%s", agenda[i].nome);
19         printf("Entre com o telefone %d:", i+1);
20         scanf("%s", agenda[i].telefone);
21     }
22     return nro_contatos;
23 }
24
25 void pesquisa_agenda (int tam) {
26     char nome[20];
27     int i =0;
28     printf("Qual nome? ");
29     scanf("%s", nome);
30     while (strcmp(agenda[i].nome, nome) != 0 && i < tam)
31         i++;
32     if (i < tam)
33         printf("O telefone e' %s \n", agenda[i].telefone);
34     else
35         printf("Telefone nao encontrado\n");
36 }
37
38 int main() {
39     int n;
40     n = cria_agenda();
41     pesquisa_agenda(n);
42 }
```

Impacto da escolha da abstração principal no reuso

Opções de projeto do tipo abstrato de dados

- Abstração principal

- Contato

- Aplicações envolvendo

sempre o mesmo

tipo de contato

- Por exemplo: agenda,
cadastro de clientes,
cadastro de fornecedores, ...

```
#ifndef BIBLIOTECA-CONTATO_H_INCLUDED
#define BIBLIOTECA-CONTATO_H_INCLUDED

#include <stdio.h>
#include <stdlib.h>

typedef struct {
    char nome[20];
    char telefone[20];
} tipo_contato;

tipo_contato cria_contato(int i);

tipo_contato *busca_contato(char nome[20], tipo_contato *lista_contatos, int tam);

#endif // BIBLIOTECA-CONTATO_H_INCLUDED
```

- Outra alternativa: Lista de Estruturas Quaisquer

- Aplicações em qualquer domínio de problema
- Sistemas Operacionais, Aplicações móveis, ... Qualquer coisa...