



Estruturas de Dados-I

Tipos Abstratos de Dados (TAD)

Neste tópico abordaremos a técnica de compilação em separado de programas e o conceito de Tipos Abstratos de Dados (TAD)

Prof. Ciro Cirne Trindade



Módulos e compilação em separado

- Um programa em C pode ser dividido em vários arquivos-fontes
- Em aplicações grandes é comum identificar funções afins e agrupá-las por arquivos que chamamos de **módulos**
 - Assim, a implementação de um programa pode ser composta por um ou mais módulos
 - Facilita a divisão de uma tarefa complexa em tarefas menores (dividir para conquistar)
 - Reaproveitamento de código



Módulos e compilação em separado

- No caso de um programa com vários módulos, cada um deles deve ser compilado separadamente, gerando um arquivo objeto (extensão .o) para cada módulo
- Após a compilação de todos os módulos, uma outra ferramenta, denominada ligador ou linkeditor, é usada para juntar todos os arquivos objetos em um único executável



Módulos e compilação em separado

- Para ilustrar o uso de módulos em C, vamos considerar a existência de um arquivo *my_io.c* que contém a implementação das funções de entrada de dados `read_line` e `read_int`



Programa *my_io.c* (1/2)

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_LENGTH 120

void read_line (char line[], int max_length)
{
    int i = 0;
    char ch;

    while ((ch = getchar()) != '\n') {
        if (i < max_length)
            line[i++] = ch;
    }
    line[i] = '\0';
}
```



Programa *my_io.c* (2/2)

```
bool read_int (int * val)
{
    char line[MAX_LENGTH + 1];
    read_line(line, MAX_LENGTH);
    return (sscanf(line, "%d", val) == 1);
}
```



Programa *main.c*

```
#include <stdio.h>
#include <stdbool.h>
void read_line (char *, int);
bool read_int (int *);
int main() {
    int n;
    printf("Tamanho da string (min. 5): ");
    while (!read_int(&n) || n < 5) {
        printf("Você não digitou um inteiro maior ou igual
               a 5!\nTamanho da string (min. 5): ");
    }
    char str[n + 1];
    printf("Informe uma string (máx. %d): ", n);
    read_line(str, n);
    printf("String informada: %s\n", str);
    return 0;
}
```

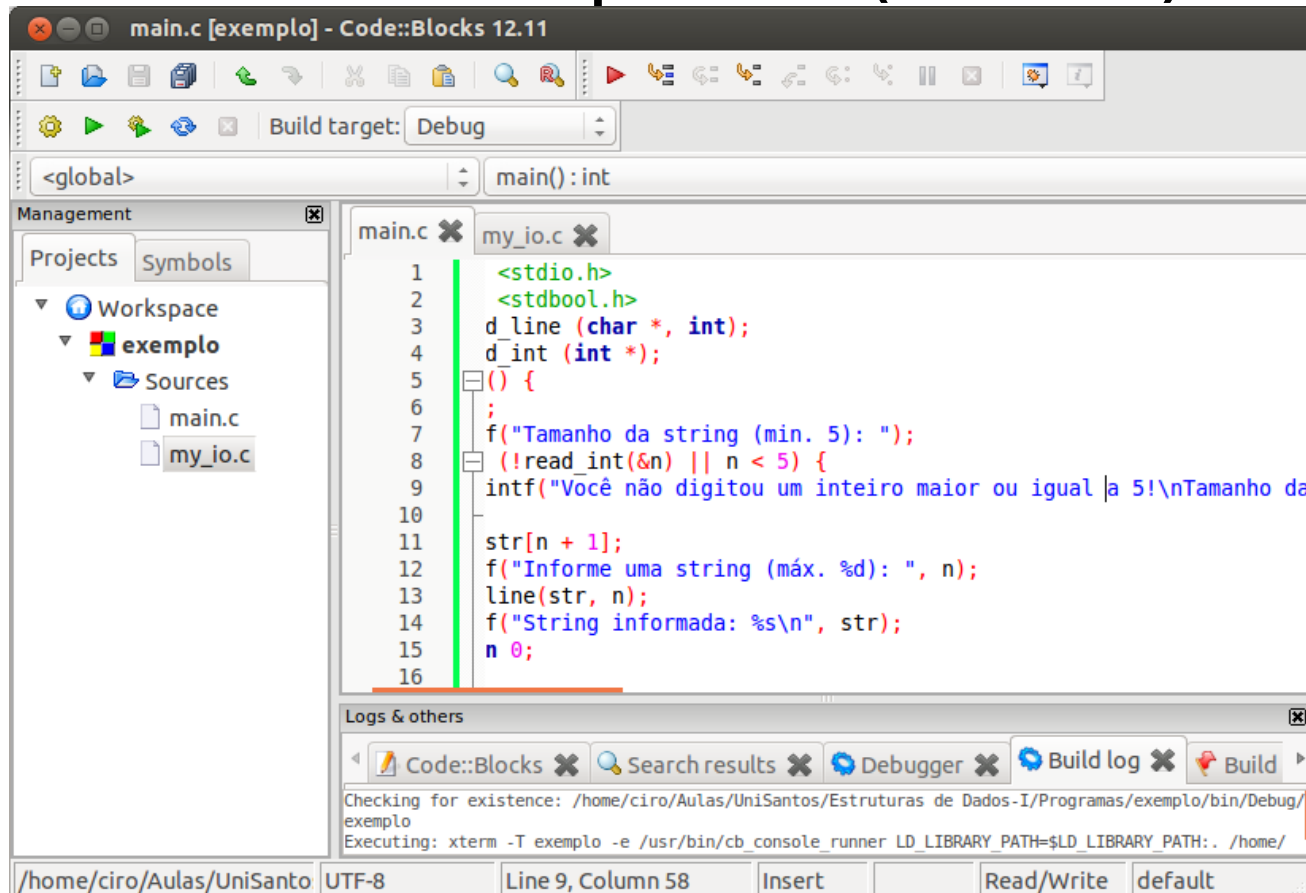


Módulos e compilação em separado

- A partir desses dois arquivos-fontes (*my_io.c* e *main.c*), podemos gerar um programa executável compilando cada um separadamente e depois ligando-os em um único arquivo executável
- Por exemplo:
 - `gcc -c my_io.c`
 - `gcc -c prog1.c`
 - `gcc -o prog1 prog1.o my_io.o`

Múltiplos módulos no Code Blocks

- É necessário criar um projeto
- Incluir todos os arquivos (.c e .h) no projeto





Módulos e compilação em separado

- Para que outros programas utilizem o módulo *my_io.c*, eles precisam conhecer os protótipos das funções oferecidas por *my_io.c*
- No exemplo anterior isso foi resolvido por meio da repetição dos protótipos no início do arquivo *main.c*
- Isto é inviável para módulos com muitas funções ou se desejarmos usar funções de muitos módulos



Módulos e compilação em separado

- Para contornar esse problema, todo módulo de funções C costuma estar associado a um arquivo de cabeçalho (.h) que contém apenas os protótipos das funções oferecidas pelo módulo e, às vezes, os tipos de dados exportados
- Este arquivo caracteriza a interface do módulo
- Em geral possui o mesmo nome do módulo só que com a extensão .h



my_io.h

```
/* my_io.h
 * Funções oferecidas pelo módulo my_io.c */
#ifndef _MY_IO_H
#define _MY_IO_H
#define MAX_LENGTH 120 // comprimento máximo da entrada
#include <stdbool.h>

/* Função read_line: lê uma linha do teclado cujo tamanho máximo é
 * dado pelo 2º argumento e a armazena no 1º argumento */
void read_line (char *, int);

/* Função read_int: lê uma linha do teclado e armazena em seu
 * parâmetro se a linha contiver um inteiro e devolve verdadeiro,
 * caso contrário, devolve falso */
bool read_int (int *);

#endif
```

Retire a definição
desta constante do
my_io.c



Módulos e compilação em separado

- É uma boa prática comentar as funções oferecidas por um módulo
- Para evitar que o mesmo arquivo `.h` seja incluído por mais de um módulo de uma aplicação, costuma-se definir uma constante que identifica o módulo e verificar se esta constante ainda não foi definida
- Por convenção o nome da constante é `_NOME-DO-MÓDULO_H`



Módulos e compilação em separado

- Agora, em vez de repetir manualmente os protótipos das funções, basta apenas incluir o arquivo .h

- Por exemplo:

```
#include <stdio.h>
#include "my_io.h"
int main() {
    ...
}
```

Os arquivos de cabeçalho das funções da biblioteca padrão de C são incluídos da forma

```
#include <arquivo.h>
```

Os arquivos de cabeçalho de nossos módulos são geralmente incluídos da forma

```
#include "arquivo.h"
```

make

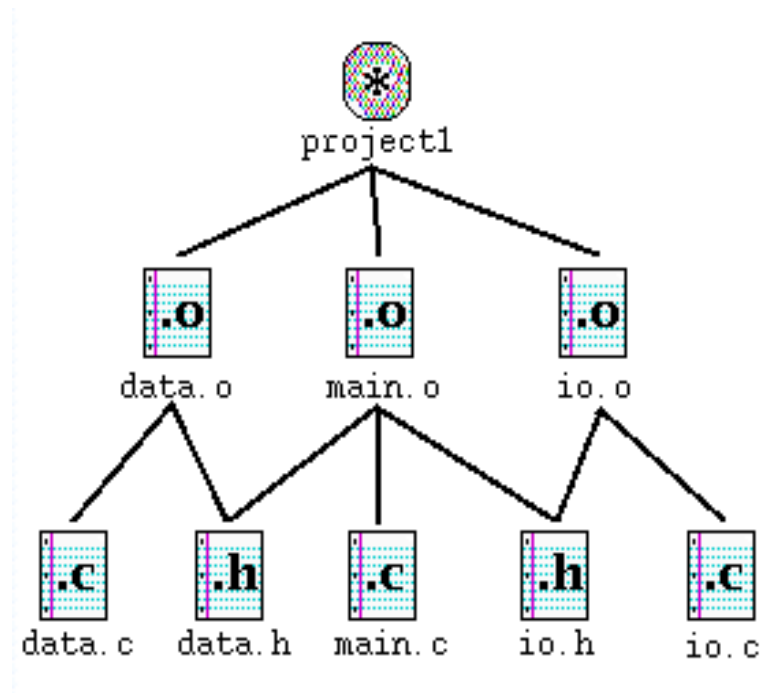
- *make* é uma ferramenta que controla a geração de arquivos executáveis e objetos a partir dos programas-fontes da aplicação
- O *make* obtém as informações de como construir a aplicação a partir de um arquivo chamado *makefile*, que lista cada arquivo objeto ou executável e como gerá-lo a partir de outros arquivos

Dependências

- O *make* cria o programa a partir de suas **dependências** descritas no arquivo *makefile*
- Por exemplo, para criar um arquivo objeto, *programa.o*, é necessário pelo menos o arquivo *programa.c* (pode haver outras dependências, como arquivos de cabeçalho *.h*)

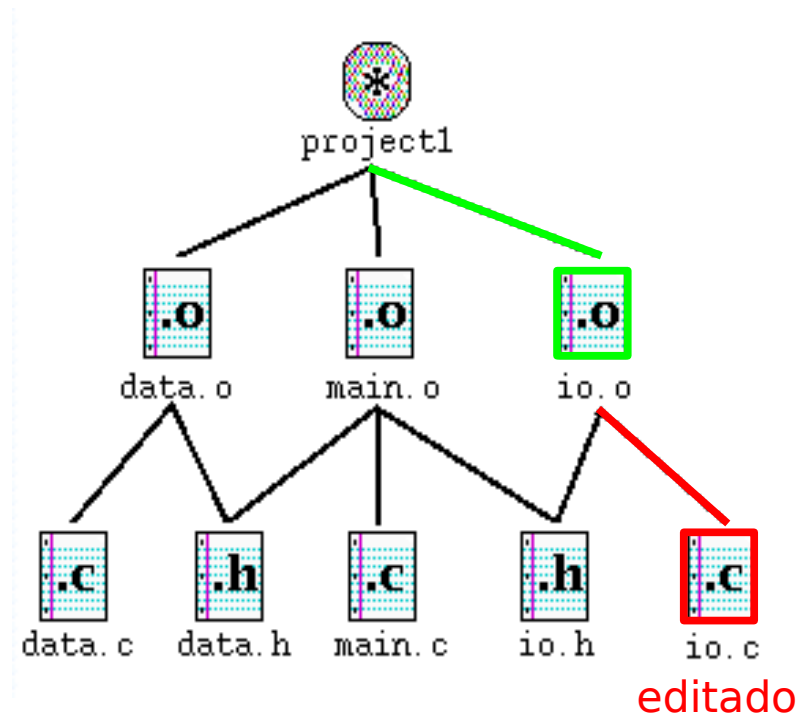
make

- Grafo de dependências



make


- Grafo de dependências



makefile

- O arquivo *makefile* ou *Makefile* determina o relacionamento entre arquivos-fontes, objetos e executável
- Uma dependência no *makefile* é definida da seguinte forma

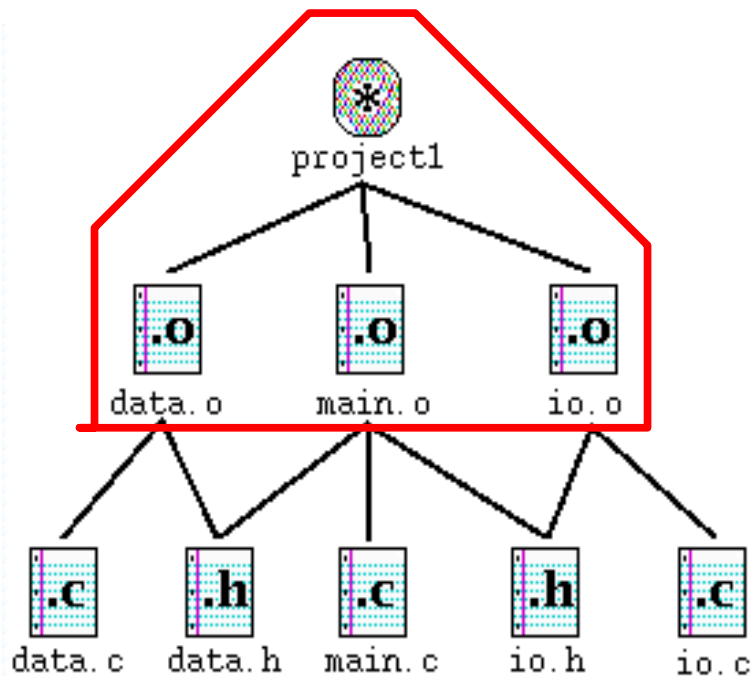
```
target: dependência1 dependência2 ...
```

 comando para gerar target

tabulação

makefile

■ Exemplo de um makefile



```
project1: main.o data.o io.o
```

```
gcc -o project1 main.o data.o io.o
```

```
main.o: main.c data.h io.h
```

```
gcc -c main.c
```

```
data.o: data.c data.h
```

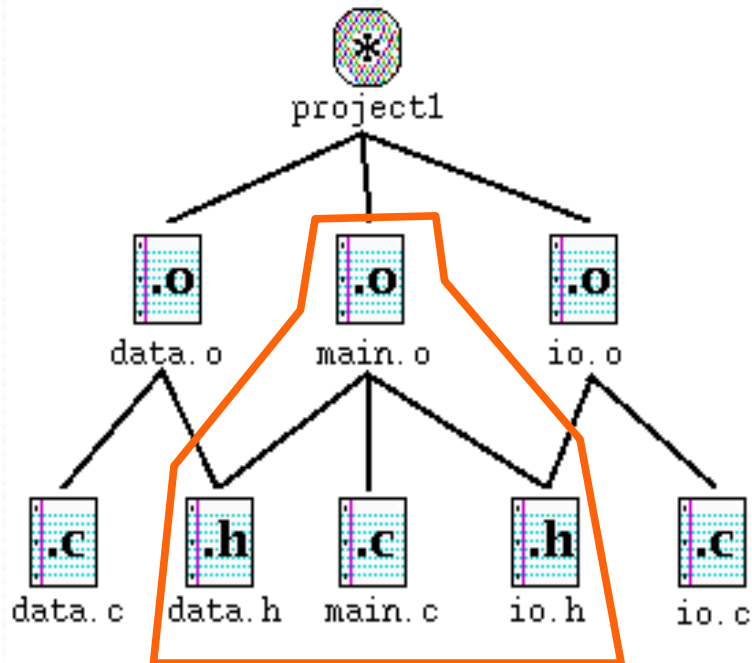
```
gcc -c data.c
```

```
io.o: io.c io.h
```

```
gcc -c io.c
```

makefile

■ Exemplo de um makefile



```
project1: main.o data.o io.o
```

```
gcc -o project1 main.o data.o io.o
```

```
main.o: main.c data.h io.h
```

```
gcc -c main.c
```

```
data.o: data.c data.h
```

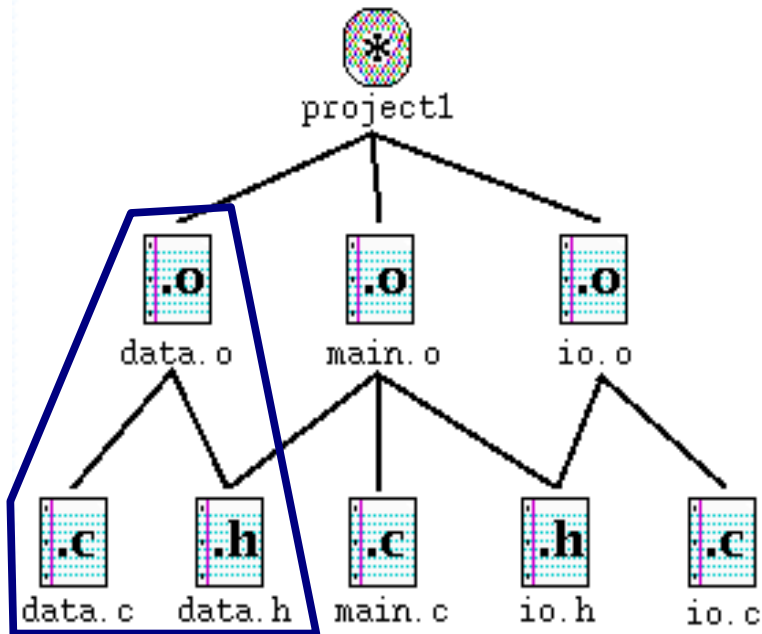
```
gcc -c data.c
```

```
io.o: io.c io.h
```

```
gcc -c io.c
```

makefile

■ Exemplo de um makefile



```
project1: main.o data.o io.o
```

```
gcc -o project1 main.o data.o io.o
```

```
main.o: main.c data.h io.h
```

```
gcc -c main.c
```

```
data.o: data.c data.h
```

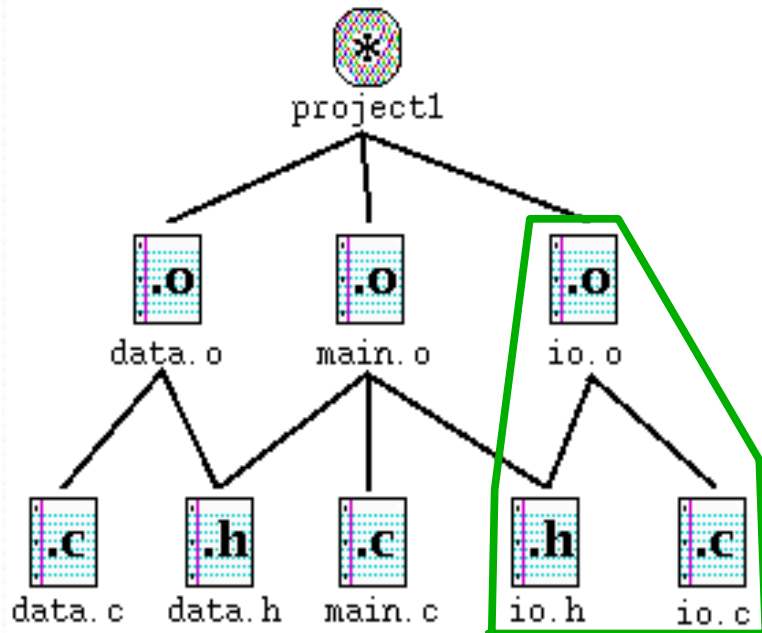
```
gcc -c data.c
```

```
io.o: io.c io.h
```

```
gcc -c io.c
```

makefile

■ Exemplo de um makefile



```
project1: main.o data.o io.o
```

```
gcc -o project1 main.o data.o io.o
```

```
main.o: main.c data.h io.h
```

```
gcc -c main.c
```

```
data.o: data.c data.h
```

```
gcc -c data.c
```

```
io.o: io.c io.h
```

```
gcc -c io.c
```



makefile

- Após criar seu *makefile* e os arquivos-fontes de sua aplicação, basta digitar `make` no terminal
- Se você não quiser chamar seu arquivo de dependências de *makefile* ou *Makefile*, você pode digitar
 - `make -f mymakefile`
- A ordem das dependências é importante, pois o *make* tentará criar ou atualizar a primeira dependência listada



Exercícios

- 1) Acrescente ao módulo *my_io* as funções `read_float` e `read_double`.
- 2) Implemente uma aplicação que use o módulo *my_io*. Forneça o makefile da aplicação.

Tipos Abstratos de Dados

- Quando um módulo define um novo tipo de dado e o conjunto de operações para manipular esse tipo, dizemos que o módulo representa um tipo abstrato de dado (TAD)
- A ideia central de um TAD é esconder de quem usa um determinado tipo a forma concreta com que ele foi implementado
 - Um TAD é descrito pela finalidade do tipo e de suas operações, e não pela forma como está implementado

Tipos Abstratos de Dados

- A interface de um TAD consiste, basicamente, na definição do nome do tipo e do conjunto de funções exportadas para sua criação e manipulação
- É comum tipos distintos oferecerem operações similares
- Por exemplo, qualquer TAD deve oferecer uma função para sua criação ou inicialização

Tipos Abstratos de Dados

- Para permitir o uso de tipos distintos pela mesma aplicação, precederemos os nomes das funções exportadas por um prefixo que identifica a qual tipo as funções se aplicam
- Por exemplo:
 - A função para inicializar um TAD `ponto` pode ser chamada `pto_init`
 - A função para inicializar um TAD `circulo` pode se chamar `circ_init`

Exemplo: TAD ponto

- Vamos considerar a criação de um tipo de dado para representar um ponto no R^2
- Para isso, devemos definir um TAD chamado **ponto** e o conjunto de funções que operam sobre esse tipo
- Vamos considerar as seguintes operações:
 - **pto_init**: operação que inicializa um ponto com as coordenadas x e y
 - **pto_distancia**: operação que calcula a distância entre 2 pontos



ponto.h

```
/* ponto.h
 * TAD para representar um ponto no  $R^2$  */

#ifdef _PONTO_H
#define _PONTO_H

/* Definição do tipo Ponto */
typedef struct {
    int x;
    int y;
} ponto;

/* Função pto_init
 * Inicializa as coordenadas x,y de um ponto */
void pto_init(ponto *, int, int);

/* Função pto_distancia
 * Devolve a distância entre 2 pontos */
double pto_distancia(ponto, ponto);

#endif
```



ponto.c

```
/* ponto.c
 * Implementa as operações que manipulam o TAD ponto */

#include <math.h>
#include "ponto.h"

void pto_init(ponto * p, int x, int y) {
    p->x = x;
    p->y = y;
}

double pto_distancia(ponto p1, ponto p2) {
    int dx = p2.x - p1.x;
    int dy = p2.y - p1.y;
    return sqrt(dx * dx + dy * dy);
}
```



Exercícios

- 1) Implemente um programa em C que dados dois pontos no R^2 , calcule e imprima a distância entre eles. Utilize o módulo *ponto.c* implementado anteriormente. Forneça também o makefile da aplicação.

Exercícios

2) Defina um TAD `circulo` definido pela estrutura abaixo:

```
typedef struct {  
    ponto centro;  
    double raio;  
} circulo;
```

Este TAD deve exportar as seguintes operações:

- a) `void circ_init(circulo * c, int x, int y, double r)`: operação que inicializa um círculo `c` com centro no ponto `x,y` e raio `r`
- b) `double circ_area(circulo c)`: operação que calcula e devolve a área do círculo `c` (πR^2)
- c) `bool circ_contains(circulo c, ponto p)`: operação que verifica se um ponto `p` está dentro do círculo `c` (a distância do ponto ao centro do círculo é menor ou igual a seu raio)



Exercícios

3) Implemente um programa que dado um círculo e um ponto, informe a área do círculo e se o ponto está dentro do círculo. Forneça também o makefile da aplicação.

Referências

- CELES, W.; CERQUEIRA, R.; RANGEL, J.L..
Introdução a Estruturas de Dados. Elsevier, 2004.
- TENENBAUM, A.M.; LANGSAM, Y.;
AUGENSTEIN, M.J..
Estruturas de Dados Usando C. Makron Books, 1995.