



UNIVERSIDADE FEDERAL DO PIAUÍ - UFPI
CAMPUS SENADOR HELVÍDIO NUNES DE BARROS - CSHNB
CURSO DE SISTEMAS DE INFORMAÇÃO
PICOS - PI

ALOCAÇÃO DINÂMICA

Prof. Ma. Luana Batista da Cruz
luana.b.cruz@nca.ufma.br

Roteiro

2

- Gerenciamento de memória
- Alocação de memória (estática e dinâmica)
- Alocação dinâmica de memória
- Alocação dinâmica de vetores
- Alocação dinâmica de matrizes
- Alocação dinâmica de estruturas

Roteiro

3

- Gerenciamento de memória
- Alocação de memória (estática e dinâmica)
- Alocação dinâmica de memória
- Alocação dinâmica de vetores
- Alocação dinâmica de matrizes
- Alocação dinâmica de estruturas

Gerenciamento de memória

4

- ❑ A memória utilizada por um programa de computador é dividida em:

Memória estática	Código do programa	← Segmento de código
	Variáveis globais e Variáveis estáticas	← Segmento de dados
Memória dinâmica	Variáveis locais (Pilha de execução)	← Stack
	Espaço de memória livre	
	Variáveis alocadas dinamicamente	← Heap

Gerenciamento de memória

5

❑ Segmento de código

- É a parte da memória que armazena o “código de programa”
- É estático em tamanho e conteúdo (de acordo com o executável)
- Geralmente, o bloco de segmento de código é somente leitura
 - As instruções do programa compilado e em execução não pode ser alterado

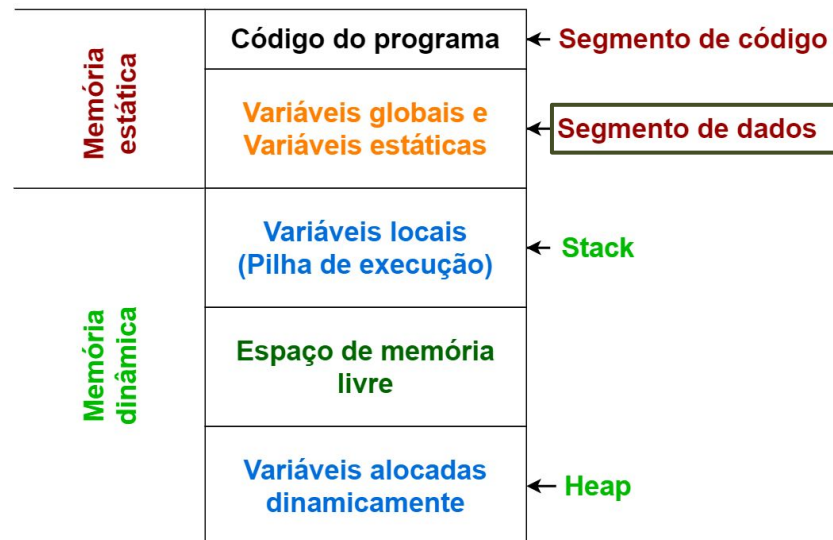
Memória estática	Código do programa	← Segmento de código
	Variáveis globais e Variáveis estáticas	← Segmento de dados
Memória dinâmica	Variáveis locais (Pilha de execução)	← Stack
	Espaço de memória livre	
	Variáveis alocadas dinamicamente	← Heap

Gerenciamento de memória

6

❑ Segmento de dados

- É a parte da memória que armazena as “variáveis globais” e “variáveis estáticas” inicializadas no código do programa
- O tamanho do segmento é calculado de acordo com os valores das variáveis definidas
- O acesso é de leitura-escrita
 - Os valores das variáveis neste segmento podem ser alterados durante a execução do programa

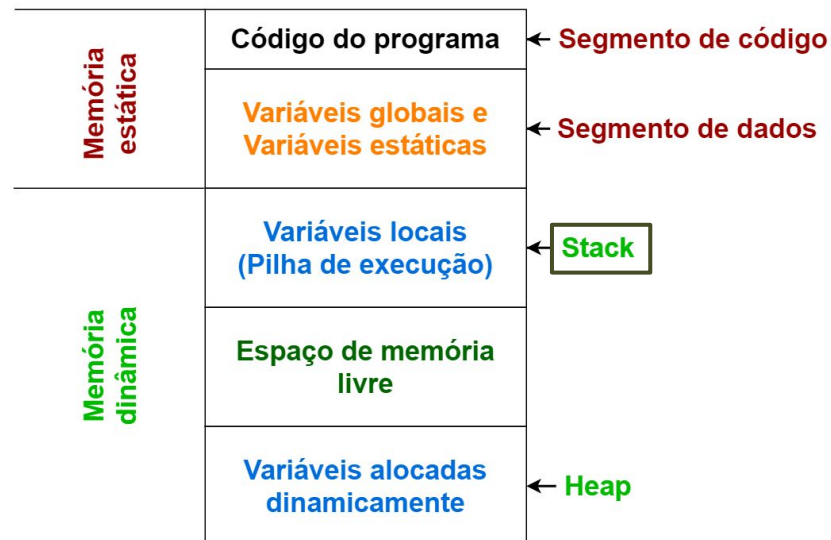


Gerenciamento de memória

7

□ Stack

- É a parte da memória que armazena as “variáveis locais” e chamadas de funções do programa
- Usa a estratégia LIFO (last-in-first-out) para gerenciar a entrada/saída de dados na memória
- O tamanho da Stack é variável e depende do sistema operacional e compilador
- Utilizar mais memória Stack do que disponível provoca um erro de execução:
 - “stack buffer overflow”

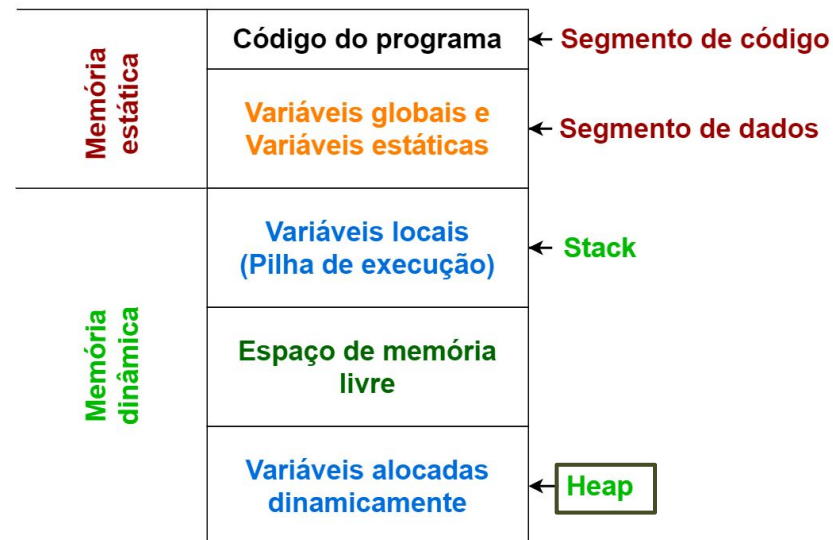


Gerenciamento de memória

8

□ Heap

- É um espaço reservado para alocação dinâmica de memória dos programas
- Memória alocada dinamicamente pode ser usada e liberada a qualquer momento
- A linguagem C fornece funções próprias para lidar com alocação dinâmica de memória
 - Malloc
 - Calloc
 - Realloc
 - Free



Alocação de memória

9

❑ Estática

- Quantidade total de memória utilizada pelos dados é previamente conhecida e definida de modo imutável, no próprio código-fonte do programa
- Durante toda a execução, a quantidade de memória utilizada pelo programa não varia

Alocação de memória

10

❑ Estática

- Implementação simples: vetores (array)
- **Vantagem**
 - Acesso indexado: todos os elementos da estrutura são igualmente acessíveis
- **Desvantagens**
 - Tamanho fixo:
#define TAM 1000
int v[TAM];
 - Alocados em memória de forma estática

Alocação de memória

11

□ Estática

- Ao se determinar o máximo de elementos que o vetor irá conter, pode-se ocorrer um dos seguintes casos:
 - **Subdimensionamento:** haverá mais elementos a serem armazenados do que o vetor é capaz de conter
 - **Superdimensionamento:** na maior parte do tempo, somente uma pequena porção do vetor será realmente utilizada

Alocação de memória

12

❑ Dinâmica

- O programa é capaz de criar novas variáveis enquanto está sendo executado
- Alocação de memória para componentes individuais no instante em que eles começam a existir durante a execução do programa
- Exemplo: vetor alocado dentro do corpo de uma função pode ser usado fora do corpo da função, enquanto estiver alocado
- Deve ser utilizada quando não se sabe ao certo quanto de memória será necessário para o armazenamento das informações
 - Dessa forma evita-se o desperdício de memória ou a falta de memória

Alocação de memória

13

❏ Dinâmica

- Implementação eficiente: ponteiros ou apontadores
- **Vantagens**
 - Tamanho variável
 - Alocados em memória de forma dinâmica
- **Desvantagens, ou restrições:**
 - Capacidade da memória
 - Acesso sequencial

Modeladores (cast)

14

- ❑ Um modelador é aplicado a uma expressão. Ele **força** a mesma a ser de um tipo especificado
- ❑ Sua forma geral é:
 - *(tipo) expressão*

Modeladores (cast)

Exemplo

15

```
#include <stdio.h>
int main () {
    int num;
    float f;
    num=10;
    /* Uso do modelador. Força a transformação de num em um float */
    f= (float) num/7;
    printf ("%f",f);
    return(0);
}
```

Alocação dinâmica de memória

16

- ❑ O padrão C ANSI define apenas 4 funções para o sistema de alocação dinâmica, disponíveis na biblioteca **stdlib.h**:
 - Malloc
 - Calloc
 - Realloc
 - Free

Alocação dinâmica de memória

17

❑ **Malloc (Memory Allocation)**

- Função para requisitar alocação dinâmica de memória
- Recebe como parâmetro o tamanho em bytes de memória a ser alocada
- Retorna um ponteiro genérico para o endereço inicial da área de memória alocada, se houver espaço livre
 - Ponteiro genérico é representado por void*
 - Ponteiro é convertido automaticamente para o tipo apropriado
 - Ponteiro pode ser convertido explicitamente (cast)
- Retorna um endereço nulo, se não houver espaço livre:
 - Representado pelo símbolo NULL

Alocação dinâmica de memória

18

❏ **Malloc (exemplo)**

- Alocação dinâmica de um tipo primitivo inteiro (int)

```
int *p = NULL;  
p = (int *) malloc(sizeof(int));
```

Tipo de dados que o
ponteiro irá receber

Tamanho em bytes do tipo
“int” na linguagem C

Alocação dinâmica de memória

19

❑ Tratamento de erro: malloc (exemplo)

- Imprime mensagem de erro
- Aborta o programa (com a função exit)

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

int main(){
    int *p;
    int a;
    // Determina o valor de a em algum lugar
    p=(int *)malloc(sizeof(int));

    if (p==NULL) {
        printf("*** Erro: Memoria Insuficiente***");
        exit(1); // aborta o programa ao sistema operacional
    }
    return 0;
}
```

Alocação dinâmica de memória

20

❑ **Calloc (Cleared Allocation)**

- A função `calloc()` também serve para alocar memória
- Faz alocação de memória em blocos e **inicializa a memória alocada com zero**
 - A memória alocada com `malloc` não é inicializada (contém lixo). Fica a cargo do programador inicializar a memória alocada

Alocação dinâmica de memória

21

❑ Calloc (exemplo)

- Alocação dinâmica de um tipo primitivo inteiro (int)

```
int *p = NULL;  
p = (int *) calloc(1, sizeof(int));
```

Tipo de dados que o
ponteiro irá receber

Tamanho em bytes do tipo
“int” na linguagem C

Alocação dinâmica de memória

22

❑ **Realloc**

- A função `realloc()` serve para realocar memória
- Um ponteiro para o bloco é devolvido porque `realloc()` pode precisar mover o bloco para aumentar seu tamanho
 - Se isso ocorrer, o conteúdo do bloco antigo é copiado no novo bloco, e nenhuma informação é perdida
- Se não houver memória suficiente para a alocação, um ponteiro nulo é devolvido e o bloco original é deixado inalterado

Alocação dinâmica de memória

23

❏ Realloc (exemplo)

```
int *p = NULL, n = 30;  
p = (int *) malloc(sizeof(int));  
p = (int *) realloc(p, n*sizeof(int));
```

Tipo de dados que o
ponteiro irá receber

Número de bytes que
queremos realocar

Alocação dinâmica de memória

24

□ Free

- A função free recebe como entrada o ponteiro para a memória alocada
- Toda memória alocada deve ser LIBERADA
 - É uma boa prática de programação

```
// alocando memória  
int *p = NULL;  
p = (int *) malloc(sizeof(int));  
  
//liberando memória  
free(p);
```


Atividade 1

25

- ❑ Escreva um trecho de código que seja capaz de ler uma string do teclado e em seguida escrever a string. O seu código deve perguntar ao usuário o tamanho da string que ele deseja digitar. Faça usando alocação dinâmica.

Atividade 2

26

- ❑ **Considere um cadastro de produtos de um estoque, com as seguintes informações para cada produto:**
 - Código de identificação do produto: representado por um valor inteiro
 - Nome do produto: com até 50 caracteres
 - Quantidade disponível no estoque: representado por um número inteiro
 - Preço de venda: representado por um valor real
- a) Defina uma estrutura em C, denominada produto, que tenha os campos apropriados para guardar as informações de um produto, conforme descrito acima
- b) Escreva um programa que leia um inteiro a, uma string b de 50 caracteres, um inteiro c, e um float d e atribua esses valores lidos aos componentes de uma variável p que é do tipo struct produto
- c) Imprima os valores de p

Atividade 2

27

- ❑ **Considere o mesmo produto do exercício anterior**
- a) Declare um vetor de estruturas produto com x produtos alocados dinamicamente
- b) Escreva um programa principal que, para i de 0 a 4, leia os campos da estrutura nas variáveis a, b, c e d (como no exercício anterior), chame uma função para armazenar essas variáveis na posição i do vetor de estrutura
- c) A função recebe os dados de um produto (código, nome, quantidade e preço) e armazena-os em um endereço de um struct produto recebido como parâmetro. Essa função pode ter o seguinte protótipo: `void gravaProd (int cod, char* nome, int quant, float preco, struct produto *p);`
- d) Imprima o vetor de produtos, sendo um produto por linha, e no fim o total que o dono da loja receberia se vendesse todos os produtos

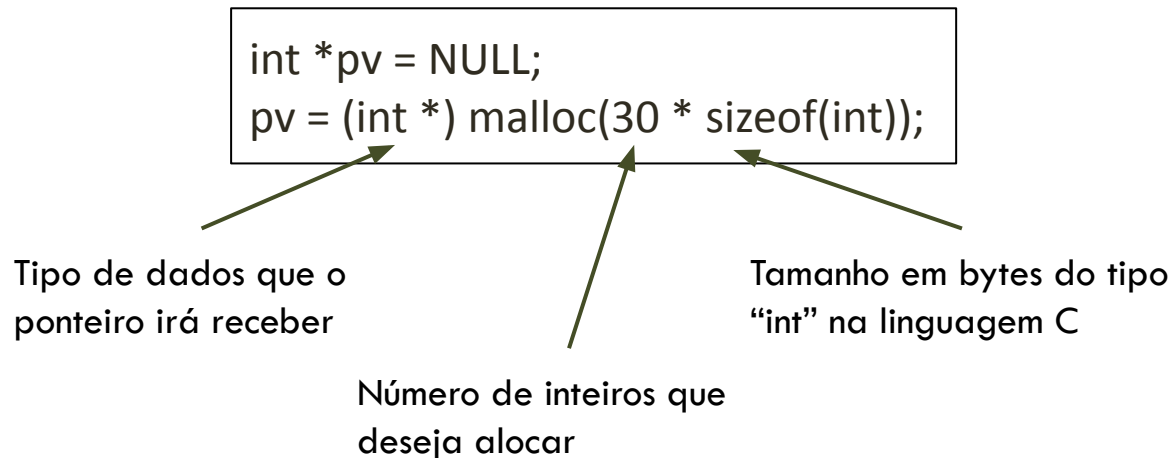
28

Alocação dinâmica de vetores e matrizes

Alocação dinâmica de vetores

29

- ❑ A alocação dinâmica de vetores utiliza os conceitos aprendidos na aula sobre ponteiros e as funções de alocação dinâmica apresentadas
- ❑ Alocar dinamicamente um espaço de memória para 30 inteiros (vetor)



Alocação dinâmica de vetores

30

Exemplo

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

float *alocar_vetor_real (int n) {
    float *v;    /* ponteiro para o vetor */
    /* verifica parametros recebidos */
    if (n < 1) {
        printf ("** Erro: parametro invalido **\n");
        return (NULL);
    }
    /* aloca o vetor */
    v = (float *) calloc(n, sizeof(float));
    if (v == NULL) {
        printf ("** Erro: memoria insuficiente **");
        return (NULL);
    }
    /* retorna o ponteiro para o vetor */
    return (v);
}
```

Alocação dinâmica de vetores

31

❏ Exemplo

```
void insere_vetor(float *p, int n){
    int i;
    for(i=0; i < n; i++){
        *(p+i) = i*10; // p[i] = i*10;
    }
}

void imprime_vetor(float *p, int n){
    int i;
    for(i=0; i < n; i++){
        printf("%.2f\n", *(p+i));
    }
}
```

Alocação dinâmica de vetores

32

❏ Exemplo

```
float *liberar_vetor_real (float *v) {  
    if (v == NULL)  
        return (NULL);  
    free(v); /* libera o vetor */  
    return (NULL); /* retorna nulo para um o ponteiro */  
}  
  
int main () {  
    float *p;  
    int a = 10;  
    p = alocar_vetor_real (a);  
    insere_vetor(p, a);  
    imprime_vetor(p, a);  
    p = liberar_vetor_real (p);  
    return(0);  
}
```


Alocação dinâmica de vetores

33

Exemplo

```
float *liberar_vetor_real (float *v) {  
    if (v == NULL)  
        return (NULL);  
    free(v); /* libera o vetor */  
    return (NULL); /* retorna nulo para um o ponteiro */  
}  
  
int main () {  
    float *p;  
    int a = 10;  
    p = alocar_vetor_real (a);  
    insere_vetor(p, a);  
    imprime_vetor(p, a);  
    p = liberar_vetor_real (p);  
    return(0);  
}
```

Atividade: refaça o aloca vetores e use o realloc

Alocação dinâmica de matrizes

34

- ❑ A alocação dinâmica de memória para matrizes é realizada da mesma forma que para vetores, com a diferença que teremos um ponteiro apontando para outro ponteiro que aponta para o valor final, ou seja, é um ponteiro para ponteiro, o que é denominado indireção múltipla
- ❑ A indireção múltipla pode ser levada a qualquer dimensão desejada, mas raramente é necessário mais de um ponteiro para um ponteiro

Alocação dinâmica de matrizes

35

- ❑ Um exemplo de implementação para matriz real bidimensional é fornecido a seguir
- ❑ A estrutura de dados utilizada neste exemplo é composta por um vetor de ponteiros (correspondendo ao primeiro índice da matriz), sendo que cada ponteiro aponta para o início de uma linha da matriz
- ❑ Em cada linha existe um vetor alocado dinamicamente, como descrito anteriormente (compondo o segundo índice da matriz)

Alocação dinâmica de matrizes

36

Exemplo

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

float **alocar_matriz_real (int m, int n) {
    float **v; /* ponteiro para a matriz */
    int i;
    /* verifica parametros recebidos */
    if (m < 1 || n < 1) {
        printf ("** Erro: parametro invalido **\n");
        return (NULL);
    }
    /* aloca as linhas da matriz */
    v = (float **) calloc(m, sizeof(float *)); // Vetor de m ponteiros para float (linhas)
    if (v == NULL) {
        printf ("** Erro: memoria insuficiente **");
        return (NULL);
    }
    /* aloca as colunas da matriz */
    for ( i = 0; i < m; i++ ) {
        v[i] = (float *)calloc (n, sizeof(float)); // m vetores de n floats (colunas)
        if (v[i] == NULL) {
            printf ("** Erro: memoria insuficiente **");
            return (NULL);
        }
    }
    return (v);
}
```

Alocação dinâmica de matrizes

37

❏ Exemplo

```
float **liberar_matriz_real (int m, int n, float **v) {  
    int i;  
    if (v == NULL)  
        return (NULL);  
    /* verifica parametros recebidos */  
    if (m < 1 || n < 1) {  
        printf ("** Erro: Parametro invalido **\n");  
        return (v);  
    }  
    for (i=0; i<m; i++)  
        free (v[i]); /* libera as linhas da matriz (vetor de float) */  
    free (v); /* libera a matriz (vetor de ponteiros) */  
    return (NULL); /* retorna nulo para um ponteiro */  
}
```

Alocação dinâmica de matrizes

38

❏ Exemplo

```
int main () {  
    float **mat; /* matriz a ser alocada */  
    int l=4, c=4; /* número de linhas e colunas da matriz */  
    int i, j;  
    mat = alocar_matriz_real (l, c);  
    for (i = 0; i < l; i++)  
        for (j = 0; j < c; j++)  
            mat[i][j] = i+j;  
    mat = liberar_matriz_real (l, c, mat);  
    return(0);  
}
```

Vetores locais a funções

39

```
float* prod_vetorial(float* u, float* v){
    float p[3];
    p[0] = u[1]*v[2] - v[1]*u[2];
    p[1] = u[2]*v[0] - v[2]*u[0];
    p[2] = u[0]*v[1] - v[0]*u[1];
    return p; /* ERRO: não podemos retornar endereço de área local (p é vetor) */
}
```

- ❑ Variável **p** declarada **localmente**:
 - Área de memória que a variável **p** ocupa deixa de ser válida quando a função **prod_vetorial** termina
 - Função que chama **prod_vetorial** não pode acessar a área apontada pelo valor retornado

Vetores locais a funções

40

```
float* prod_vetorial(float* u, float* v){  
    float *p = (float*) malloc(3*sizeof(float));  
    p[0] = u[1]*v[2] - v[1]*u[2];  
    p[1] = u[2]*v[0] - v[2]*u[0];  
    p[2] = u[0]*v[1] - v[0]*u[1];  
    return p;  
}
```

- ❑ Variável **p** alocada **dinamicamente**:
 - Área de memória que a variável **p** ocupa permanece válida mesmo após o término da função **prod_vetorial**
- ❑ Função que chama **prod_vetorial** pode acessar o ponteiro retornado
- ❑ Problema - alocação dinâmica para cada chamada da função
 - Ineficiente do ponto de vista computacional
 - Requer que a função que chama seja responsável pela liberação do espaço

Vetores locais a funções

41

```
void prod_vetorial(float* u, float* v, float* p){  
    p[0] = u[1]*v[2] - v[1]*u[2];  
    p[1] = u[2]*v[0] - v[2]*u[0];  
    p[2] = u[0]*v[1] - v[0]*u[1];  
}
```

- ❑ Função **prod_vetorial** recebe três vetores:
 - Dois vetores com dados de entrada
 - Um vetor para armazenar o resultado
- ❑ Solução mais adequada pois não envolve alocação dinâmica

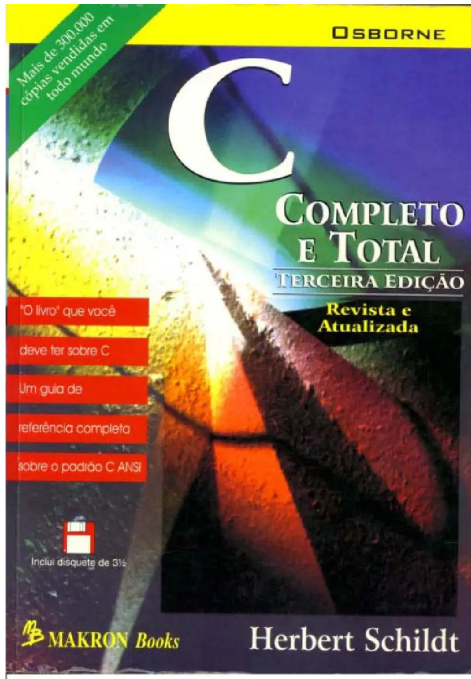
Atividade

42

- ❑ Escreva um programa em C que solicita ao usuário a quantidade de notas de uma turma e aloca um vetor de notas (números reais). Depois de ler as notas, imprime a média aritmética. Obs: não deve ocorrer desperdício de memória, e após ser utilizada a memória deve ser devolvida
- ❑ A partir dos conceitos vistos nesta aula, aloque uma estrutura equivalente para uma matriz tridimensional
 - `float ***mat`

Referências

43



SCHILD, Herbert. **C completo e total**. Makron, 3ª edição revista e atualizada, 1997.



SZWARCHFITER, J. **Estruturas de Dados e seus algoritmos**. 3 ed. Rio de Janeiro: LTC, 2010.