Programação

Folha de Consulta

Diogo Silva

2024-06-13

Variáveis

```c
int a; // declarar variável inteira chamada a
float b=3.14; // declarar variável real chamada b e inicializar com 3.14
char c1,c2='a'; // declarar variável char c1 e c2, e inicializar c2 com 'a'
```

Operadores

Tabela 1: Operadores aritméticos, relacionais, lógicos e de atribuição

Operador	Descrição	Exemplo	Operador	Descrição	Exemplo
+	Soma	a + b	==	Igual	a == b
-	Subtração	a - b	!=	Diferente	a != b
*	Multiplicação	a * b	>	Maior	a > b
/	Divisão	a / b	<	Menor	a < b
%	Resto da divisão	a % b	>=	Maior ou igual	a >= b
++	Incremento	a++	&&	E	a && b
–	Decremento	a–	||	OU	a || b
=	Atribuição	a = b	!	NÃO	!a
+=	Atribuição com soma	a += b			
-=	Atribuição com subtração	a -= b			
*=	Atribuição com multiplicação	a *= b			
/=	Atribuição com divisão	a /= b			

I/O Formatado

```c
scanf("%descritor", &variavel); // lê um dado do tipo tipo_dado e armazena em variavel
printf("%descritor", variavel); // escreve um dado do tipo tipo_dado e armazena em variavel
```

Tabela 2: Descritores de tipos de dados

Descritor	Tipo de dado
%d	int
%f	float
%c	char
%s	string

Descritor	Tipo de dado
%lf	double
%u	unsigned int
%ld	long int
%lu	unsigned long int

I/O Não Formatado na consola

```c
getchar(); // lê um char
/* lê uma string da consola (stdin),
com tamanaho máximo de TAMANHO
e escreve no array frase */
fgets(frase, TAMANHO, stdin);
```

```c
putchar(variavel); // escreve um char
puts(variavel); // escreve uma string
```

Estruturas de controlo

if

```c
if (condição) {
    // bloco de código
} else if (condição) {
    // bloco de código
} else {
    // bloco de código
}
```

switch

```c
switch (variavel) {
    case valor1:
        // bloco de código
        break;
    case valor2:
        // bloco de código
        break;
    default:
        // bloco de código
}
```

Ciclos

for

```c
for (inicialização; condição; pós-instrução) {
    // instruções
}
/*
inicialização é executada 1x
condição é testada antes
  de cada iteração
pós-instrução é executada
  no final de cada iteração
*/
```

while e do while

```c
/* enquanto condição for verdadeira */
while (condição) {
    // instruções
}
```

```c
/* executa o bloco de código pelo menos uma vez */
do {
    // intruções
} /* enquanto condição for verdadeira */
while (condição);
```

Funções

- Declaração de função

```
tipo_retorno nome_função (tipo_parametro1 nome_parametro1, tipo_parametro2 nome_parametro2) {
    // bloco de código
}
```

- Chamada de função

```
int a;
a = funcao(parametro1, parametro2); // se funcao devolvesse um int
```

Vetores

- Declaração de vetor

```
int vetor[10]; // declara vetor de inteiros com 10 posições
```

- Acesso a posição do vetor

```
vetor[0] = 1; // acessa a primeira posição do vetor e atribui 1
```

- Vetores e funções

```
int vetor[10];
funcao(vetor); // passa o vetor como parâmetro
```

```
void funcao(int vetor[]) { // recebe o vetor como parâmetro
    // função não sabe o tamanho do vetor
    // alterações ao vetor são persistentes
}
```

Vetores multidimensionais

```
int matriz[10][10]; // declara matriz de inteiros com 10 linhas e 10 colunas
matriz[0][0] = 1; // acessa a primeira linha e primeira coluna da matriz e atribui 1
int b = matriz[0][0]; // acessa a primeira linha e primeira coluna da matriz e atribui a b
```

```
int matriz[10][10][10]; //vetor tridimensional
funcao(matriz); // passa a matriz como parâmetro
```

```
void funcao(int matriz[][10][10]) { // recebe a matriz como parâmetro
    // obrigatório indicar o tamanho de todas as dimensões menos a primeira
    // alterações à matriz são persistentes
}

// VLAs (Variable Length Arrays)
void funcao(int x, int y, int z, int matriz[x][y][z]) {
    // alterações à matriz são persistentes
```

```
}

// exemplo a percorrer uma matriz
void print_vec(int size, int array[size]){
    pritnf("Conteúdo do vector: ");
    for(int i = 0; i < size; i++){
        printf("%d ", array[i]);
    }
    printf("\n");
}
```

Strings

```
char string[10]; // declara string de 10 caracteres
char string[10] = "ola"; // declara e inicializa string
char string[10] = {'o', 'l', 'a', '\0'}; // declara e inicializa string
```

Tabela 3: Funções úteis da string.h

Função	Descrição
strlen(string)	retorna o tamanho da string
strcpy(string1, string2)	copia string2 para string1
strcat(string1, string2)	concatena string2 a string1
strcmp(string1, string2)	compara string1 com string2, devolve 0 se forem iguais, < 0 se string1 $<$ string2 e > 0 se string1 $>$ string2

Apontadores

```
int *p; // declara ponteiro para inteiro

int a = 10;
p = &a; // ponteiro recebe o endereço de memória da variável a
```

```
int a = 10;
int *p;
p = &a;
printf("%d", *p); // imprime o valor apontado por ponteiro
```

Estruturas

- Declaração de estrutura

```
struct nome_estrutura {
    tipo1 nome1;
    tipo2 nome2;
};
```

- Acesso a campo da estrutura

```
struct nome_estrutura variavel;
variavel.nome1 = 1; // acessa o campo nome1 da variável variavel e atribui 1
```

- Estruturas e apontadores

```
struct nome_estrutura *ponteiro;
ponteiro = &variavel; // ponteiro recebe o endereço de memória da variável variavel
ponteiro->nome1 = 1; // acessa o campo nome1 da variável apontada por ponteiro e atribui 1
```

- `typedef`

```
typedef struct nome_estrutura {
    tipo1 nome1;
    tipo2 nome2;
} NomeEstrutura; // NomeEstrutura passa a ser um tipo

NomeEstrutura variavel; // declara variável do tipo NomeEstrutura

void funcao(NomeEstrutura x) {
    // bloco de código
}

NomeEstrutura funcao2() {
    temp = NomeEstrutura;
    return temp;
}
```

Ficheiros

- Abertura e fecho de ficheiro

```
FILE *ficheiro;
ficheiro = fopen("nome_ficheiro", "modo_abertura");
// abre o ficheiro nome_ficheiro no modo modo_abertura
// se devolver NULL, não foi possível abrir o ficheiro

fclose(ficheiro); // fecha o ficheiro
fcloseall(); // fecha todos os ficheiros abertos
```

- Modos de abertura de ficheiro
  - Para ficheiros binários, acrescentar b ao modo de abertura, e.g. “rb”, “wb”, “ab”, “rb+”, “wb+”, “ab+”.

Tabela 4: Modos de abertura de ficheiro

Modo	Leitura	Escrita	Criação	Posicionamento
r	Sim	Não	Não	Início
w	Não	Sim	Sim	Início
a	Não	Sim	Sim	Fim
r+	Sim	Sim	Não	Início
w+	Sim	Sim	Sim	Início
a+	Sim	Sim	Sim	Fim

- Leitura e escrita formatada

```c
int a = 10;
float b = 3.14;
char c = 'a';
// escreve no ficheiro os valores de a, b e c
fprintf(ficheiro, "%d %f %c", a, b, c);
// lê do ficheiro os valores de a, b e c
// devolve EOF se apanhar o fim do ficheiro
fscanf(ficheiro, "%d %f %c", &a, &b, &c);


char linha[100];
// lê uma linha do ficheiro (ou um máximo de 100 char) e guarda em linha
// devolve NULL se apanhar o fim do ficheiro
fgets(linha, 100, ficheiro);

fgetc(ficheiro); // lê um char do ficheiro
fputc('a', ficheiro); // escreve um char no ficheiro
fputs("ola", ficheiro); // escreve uma string no ficheiro
```

- Leitura e escrita ficheiros binários

```c
int a = 10;
int n_lidos, n_escritos;
FILE *ficheiro = fopen("nome_ficheiro", "wb");
n_escritos=fwrite(&a, sizeof(int), 1, ficheiro); // escreve no ficheiro o valor de a
fclose(ficheiro);

ficheiro = fopen("nome_ficheiro", "rb");
n_lidos=fread(&a, sizeof(int), 1, ficheiro); // lê do ficheiro o valor de a
fclose(ficheiro);
```

- Posicionamento no ficheiro

```c
// posiciona o cursor no ficheiro na posição 10
fseek(ficheiro, 10, SEEK_SET);
// posiciona o cursor no ficheiro 10 posições à frente da posição atual
fseek(ficheiro, 10, SEEK_CUR);
// posiciona o cursor no ficheiro 10 posições antes do fim do ficheiro
fseek(ficheiro, -10, SEEK_END);
ftell(ficheiro); // devolve a posição atual do cursor no ficheiro
rewind(ficheiro); // posiciona o cursor no início do ficheiro
```

Alocação dinâmica de memória

```c
// usar malloc
int *p;
p = malloc(2 * sizeof(int)); // aloca memória para 2 int
free(p); // libera a memória alocada para ponteiro

// usar calloc
int *p;
```

```c
p = calloc(3, sizeof(int)); // aloca memória para 3 int e inicializa a zero
free(p);

// alocar dinâmicamente um vector bidimencional (10 linhas por 20 colunas)
int **matriz;
matriz = (int **) malloc(10 * sizeof(int *));
for (int i = 0; i < 10; i++) {
    matriz[i] = (int *) malloc(20 * sizeof(int));
}

// libertar a memória alocada
for (int i = 0; i < 10; i++) {
    free(matriz[i]);
}
free(matriz);
```

Estruturas de Dados Dinâmicas

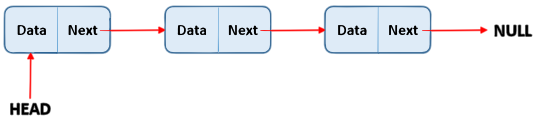

Figura 1: Lista ligada

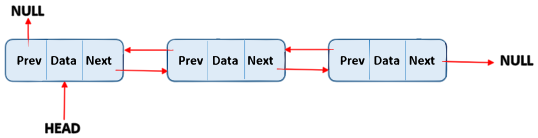

Figura 2: Lista duplamente ligada

Declaração de lista ligada

```c
typedef struct node {
    int data;
    struct node *next;
} Node;

Node *head = NULL;
```

Declaração de lista duplamente ligada

```c
typedef struct node {
    int data;
    struct node *next, *prev;
} Node;

Node *head = NULL;
```

Adicionar novo nó no final a **lista ligada**

```c
Node *new_node = malloc(sizeof(Node));
new_node->data = data;
new_node->next = NULL;

if (head == NULL) { // lista vazia
    head = new_node;
} else { // lista não vazia
    Node *current = head, *prev = NULL;
    while (current->next != NULL) {
        prev = current;
        current = current->next;
    }
    prev->next = new_node;
}
```

Libertar **lista ligada**

```c
Node *current = head, *next;
while (current != NULL) {
    next = current->next;
    free(current);
    current = next;
}
head = NULL;
```

Fila

Funciona como FIFO (First In First Out) e pode ser implementada como uma lista ligada, em que um novo item é adicionado na cauda da lista, e qualquer item é sempre removido da cabeça da lista.

Pilha

Funciona como LIFO (Last In First Out) e pode ser implementada como uma lista ligada, em que um novo item é adicionado na cabeça da lista, e qualquer item é sempre removido da cabeça da lista.