Programação

Cheatsheet

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

Table 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
```

Table 2: Descritores de tipos de dados

Descritor	Tipo de dado
%d	int
%f	float
%c	char
%s	string
%lf	double
%u	unsigned int

Descritor	Tipo de dado
%ld	long int
%lu	unsigned long int

I/O Não Formatado

```
getchar(); // lê um char
fgets(variavel, tamanho, stdin); // lê uma string

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

Estruturas de controlo

- if

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

- switch

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

- Ciclos

```
while (condição) { // enquanto condição for verdadeira
    // bloco de código
}

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

for (inicialização; condição; incremento) {
    // bloco de código
}
```

```
// inicialização é executada uma vez
// condição é testada antes de cada iteração
// pós-instrução é executada no final de cada iteração
```

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
}
```

Strings

- Declaração de string

```
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
```

- string.h

Table 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
```

Alocação dinâmica de memória

```
int *p;
p = (int *) malloc(1 * sizeof(int)); // aloca memória para um inteiro

free(ponteiro); // libera a memória alocada para 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+”.

Table 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
```