Algoritmos e Estruturas de Dados

The C Programming Language, Pt 1

Prática 1

Tipo de Dado	Exemplo
int	5
float	6.2
double	6.2
char	'a'

- Ao declararmos uma variável, função ou estrutura, estamos informando ao compilador sobre sua existência e sobre como armazená-la:

```
int numero;
float preco;
```

- Tendo declarado, estamos livres para definir a variável dentro daquele escopo sem precisar repetir seu tipo:

```
numero = 3;
preco = 6/5;
```

- Funções possuem um modelo de declaração especial:

```
tipo_do_retorno nomeDaFuncao(tipo_do_arg1 arg1, tipo_do_arg2 arg2, ...);
```

```
int somar_numeros(int num1, int num2);

double gerarNumero();
```

Typecast

```
int i = 3;
int j = 8;
return (double) i/j;

// Retorna 0.375
```

Prática 2

- Vamos ver como acessar um elemento qualquer de um array:

```
double exemplo[] = {6.5, 1.1, 9.5, 2.0};
```

Índices: 0 1 2 3

```
double valor1 = exemplo[2];
double valor2 = exemplo[0];
```

ARITMÉTICOS

(relacionam 2 operandos)

Operador	Uso	Significado
+	A + B	A mais B.
-	A - B	A menos B.
*	A * B	A vezes B.
/	A / B	A dividido por B.
%	A % B	Resto quando A é divido por B.
++	A++	Lê o valor de A e depois* faz A+1.
	++A	Faz A+1 e só então lê o valor de A.
--	A--	Lê o valor de A e depois* faz A-1.
	--A	Faz A-1 e só então lê o valor de A.

COMPARATIVOS

(retornam 1 [*true*] ou 0 [*false*])

Operador	Uso	Significado
==	A == B	A é igual a B?
!=	A != B	A é diferente de B?
>	A > B	A é maior que B?
<	A < B	A é menor que B?
>=	A >= B	A é maior ou igual a B?
<=	A <= B	A é menor ou igual a B?

LÓGICOS

(retornam 1 [*true*] ou 0 [*false*])

Operador	Uso	Significado
!	!(x<y)	Retorna o oposto do resultado da expressão. *False* vira *true* e *true* vira *false*.
&&	(x<y) && (a>b)	Retorna *true* apenas quando ambas as expressões forem *true (AND lógico)*.
||	(x<y) || (a>b)	Retorna *true* desde que pelo menos uma expressão seja *true (OR lógico)*.

IF – ELSE

```
if (condicao){
    // Executa aqui se condicao
    // avaliar para TRUE (1)
} else {
    // Executa aqui se condicao
    // avaliar para FALSE (0)
}
```

LOOP WHILE

```
while (condicao){
    // Repete enquanto
    // condicao for TRUE (1)
}
// Quando condicao for
// avaliada em FALSE (0),
// o loop quebra e executa aqui
```

```
int contagem = 0, max = 50;

while (contagem != max){
    contagem++;  // contagem = contagem + 1
}
// Quando o loop quebrar,
// contagem vai ser igual a 50
```

LOOP FOR

```
for (setup inicial; condicao; iteracao){
    // Enquanto condicao for TRUE (1),
    // repete este código aqui
}
// Quando condicao for avaliada em
// FALSE (0), o loop quebra e executa aqui
```

```
int arr[] = {5, 4, 2, 8}
for (i=0; i<4; i++){
    arr[i] += 10;
}
// Quando o loop quebrar,
// arr vai ser {15, 14, 12, 18}
```