RAFAEL VIEIRA COELHO

TIPOS DE DADOS E OPERADORES

PROGRAMAÇÃO 2

CAPÍTULO 3:TIPOS DE DADOS

https://books.goalkicker.com/CBook/

INTERPRETANDO DECLARAÇÕES

O compilador precisa interpretar as declarações de acordo com os operadores abaixo e sua respectiva precedência.

Operator	Relative Precedence	Associativity
[ ] (array subscription)	1	Left-to-right
( ) (function call)	1	Left-to-right
* (dereference)	2	Right-to-left

Expression	Interpretation
`thing[X]`	an array of size `X` of...
`thing(t1, t2, t3)`	a function taking `t1`, `t2`, `t3` and returning...
`*thing`	a pointer to...

EXEMPLOS DE VETORES

Caso seja usado parênteses para sobrecarregar a precedência, o * é aplicado primeiro. No exemplo abaixo, trata-se de um ponteiro para um array de caracteres (char) de tamanho 10.

```
char (*place)[10];
```

Sem o parênteses, names é um vetor de tamanho 20 de ponteiros para char.

```
char *names[20];
```

EXEMPLOS DE FUNÇÕES

fn é uma função que recebe como parâmetro de entrada um long e um short e retorna um dado do tipo int.

```
int fn(long, short);
```

fn é uma função que não recebe parâmetros (void) e retorna um ponteiro para int.

```
int *fn(void);
```

O parênteses inverte a precedência, então fp é um ponteiro para uma função que não recebe parâmetros e retorna um int.

```
int (*fp)(void);
```

Overriding the precedence of `()`: `fp` is a pointer to a function taking `void` and returning `int`.

VETORES BIDIMENSIONAIS

```
int arr[5][8];
```

arr é um vetor de duas dimensões (matriz) com 5 linhas e 8 colunas de elementos int.

```
int **ptr;
```

ptr é um ponteiro de um ponteiro para um int.

DECLARAÇÕES MÚLTIPLAS

Podemos fazer declarações de múltiplas variáveis desde que tenham o mesmo tipo, apenas separando-os com vírgula.

```c
int fn(void), *ptr, (*fp)(int), arr[10][20], num;
```

TIPO INTEIRO E CONSTANTES

Podemos representar um número inteiro em diversas bases, apenas alterando o prefixo ou sufixo.

```c
/* the following variables are initialized to the same value: */
int d = 42;   /* decimal constant (base10) */
int o = 052;  /* octal constant (base8) */
int x = 0xaf; /* hexadecimal constants (base16) */
int X = 0XAf; /* (letters 'a' through 'f' (case insensitive) represent 10 through 15) */
```

- Constantes decimais são sempre com sinal (signed).
- Constantes hexadecimais iniciam com 0x ou 0X (signed ou unsigned).
- Constantes cotais iniciam com 0 (signed ou unsigned).

```c
/* suffixes to describe width and signedness : */
long int i = 0x32; /* no suffix represent int, or long int */
unsigned int ui = 65535u; /* u or U represent unsigned int, or long int */
long int li = 655361; /* l or L represent long int */
```

TIPO REAL (EM PONTO FLUTUANTE)

```c
float f = 0.314f;          /* suffix f or F denotes type float */
double d = 0.314;          /* no suffix denotes double */
long double ld = 0.314l; /* suffix l or L denotes long double */

/* the different parts of a floating point definition are optional */
double x = 1.; /* valid, fractional part is optional */
double y = .1; /* valid, whole-number part is optional */

/* they can also defined in scientific notation */
double sd = 1.2e3; /* decimal fraction 1.2 is scaled by 10^3, that is 1200.0 */
```

TIPO LITERAL (STRING)

Uma string em C é uma sequencia de caracteres (char) terminados por zero.

```c
char* str = "hello, world"; /* string literal */

/* string literals can be used to initialize arrays */
char a1[] = "abc"; /* a1 is char[4] holding {'a','b','c','\0'} */
char a2[4] = "abc"; /* same as a1 */
char a3[3] = "abc"; /* a1 is char[3] holding {'a','b','c'}, missing the '\0' */
```

Elas não são modificáveis e tratadas como de apenas leitura pelo compilador (seção .rodata).

```c
char* s = "foobar";
s[0] = 'F'; /* undefined behaviour */

/* it's good practice to denote string literals as such, by using `const` */
char const* s1 = "foobar";
s1[0] = 'F'; /* compiler error! */
```

CAPÍTULO 4: OPERADORES

https://books.goalkicker.com/CBook/

OPERADORES

Um operador em programação é um símbolo que diz ao compilador que ele deve realizar uma operação:

(1) relacional,

(2) matemática

(3) ou lógica.

Ordem de Precedência de Operadores

Operador	Descrição
!	Negação
* / %	Multiplicação, divisão e resto da divisão
+ -	Adição, subtração
> < >= <=	Maior que, menor que, maior ou igual e menor ou igual
== !=	Igual e diferente
&&	Conectivo E
||	Conectivo OU
=	Operador de atribuição

OPERADORES RELACIONAIS

Operadores relacionais verificam se uma determinada relação é verdadeira ou falsa.

O resultado 1 é interpretado como verdadeiro e o 0 como falso.

São expressões comumente usadas em comandos de controle de fluxo (if, while e for), mas também podem ser armazenadas em variáveis.

```c
1 == 0;         /* evaluates to 0. */
1 == 1;         /* evaluates to 1. */

int x = 5;
int y = 5;
int *xptr = &x, *yptr = &y;
xptr == yptr;   /* evaluates to 0, the operands hold different location addresses. */
*xptr == *yptr; /* evaluates to 1, the operands point at locations that hold the same value. */
```

Obs: não confunda os operadores == e =

OPERADOR CONDICIONAL / TERNÁRIO

Avalia se uma expressão é diferente de zero (verdadeira).

```
a = b ? c : d;
```

É equivalente à:

```
if (b)
    a = c;
else
    a = d;
```

O que faz o exemplo abaixo?

```
int x = 5;
int y = 42;
printf("%i, %i\n", 1 ? x : y, 0 ? x : y);
```

% c especifica um char
% d especifica um int
% u especifica um unsigned int
% f especifica um double (ou float)
% e especifica um double (ou float) no formato científico
% g especifica um double (ou float) no formato mais apropriado (% f ou % e)
% s especifica uma cadeia de caracteres

OPERADORES RELACIONAIS

O operador ! nega a expressão a direita dele.

```
!someVal
```

E tem o mesmo efeito que:

```
someVal == 0
```

E também podemos negar a igualdade, o que resulta no operador diferente

```
1 != 0;          /* evaluates to 1. */
1 != 1;          /* evaluates to 0. */

int x = 5;
int y = 5;
int *xptr = &x, *yptr = &y;
xptr != yptr;    /* evaluates to 1, the operands hold different location addresses. */
*xptr != *yptr; /* evaluates to 0, the operands point at locations that hold the same value. */
```

OPERADORES RELACIONAIS

Podemos verificar se um valor é menor ou maior que outro.

```
5 > 4       /* evaluates to 1. */
4 > 5       /* evaluates to 0. */
4 > 4       /* evaluates to 0. */
```

```
5 < 4       /* evaluates to 0. */
4 < 5       /* evaluates to 1. */
4 < 4       /* evaluates to 0. */
```

OPERADORES RELACIONAIS

Usar o maior ou igual (>=) e menor ou igual (<=) para incluir o valor que está sendo testado:

```
5 >= 4      /* evaluates to 1. */
4 >= 5      /* evaluates to 0. */
4 >= 4      /* evaluates to 1. */
```

```
5 <= 4      /* evaluates to 0. */
4 <= 5      /* evaluates to 1. */
4 <= 4      /* evaluates to 1. */
```

COMPORTAMENTO CURTO CIRCUITO DE OPERADORES LÓGICOS

Caso a primeira expressão lógica resulte em falso em uma expressão composta &&, o restante da expressão não é avaliado.

```c
#include <stdio.h>

int main(void) {
  int a = 20;
  int b = -5;

  /* here 'b == -5' is not evaluated,
     since a 'a != 20' is false. */
  if (a != 20 && b == -5) {
    printf("I won't be printed!\n");
  }

  return 0;
}
```

OPERADOR VÍRGULA

Ele é usado para separar termos.

```c
int x = 42, y = 42;
printf("%i\n", (x *= 2, y)); /* Outputs "42". */
/*             ^         ^ this is a comma operator */
/*             this is a separator */
```

OPERADORES MATEMÁTICOS

O operador de adição (+) é usado para adicionar dois valores.

```c
#include <stdio.h>

int main(void)
{
    int a = 5;
    int b = 7;
    int c = a + b; /* c now holds the value 12 */

    printf("%d + %d = %d",a,b,c); /* will output "5 + 7 = 12" */

    return 0;
}
```

OPERADORES MATEMÁTICOS

O operador de subtração (-) é usado para diminuir o valor da direita do valor da esquerda.

```c
#include <stdio.h>

int main(void)
{
    int a = 10;
    int b = 7;

    int c = a - b; /* c now holds the value 3 */

    printf("%d - %d = %d",a,b,c); /* will output "10 - 7 = 3" */

    return 0;
}
```

OPERADORES MATEMÁTICOS

O operador de multiplicação (*) é usado para multiplicar dois valores.

```c
#include <stdio.h>

int main(void)
{
    int a = 5;
    int b = 7;

    int c = a * b; /* c now holds the value 35 */

    printf("%d * %d = %d",a,b,c); /* will output "5 * 7 = 35" */

    return 0;
}
```

Obs: não confunda com o operador usado para indicar que é um ponteiro.

O **operador de divisão (/)** é usado para fazer a divisão entre dois valores. Caso a divisão seja entre valores inteiros, a resposta conterá apenas a parte inteira. Caso estejam em ponto flutuante, o resultado será fracionado também.

```c
#include <stdio.h>

int main (void)
{
    int a = 19 / 2 ; /* a holds value 9   */
    int b = 18 / 2 ; /* b holds value 9   */
    int c = 255 / 2; /* c holds value 127 */
    int d = 44 / 4 ; /* d holds value 11  */
    double e = 19 / 2.0 ; /* e holds value 9.5   */
    double f = 18.0 / 2 ; /* f holds value 9.0   */
    double g = 255 / 2.0; /* g holds value 127.5 */
    double h = 45.0 / 4 ; /* h holds value 11.25 */

    printf("19 / 2 = %d\n", a);    /* Will output "19 / 2 = 9"    */
    printf("18 / 2 = %d\n", b);    /* Will output "18 / 2 = 9"    */
    printf("255 / 2 = %d\n", c);   /* Will output "255 / 2 = 127" */
    printf("44 / 4 = %d\n", d);    /* Will output "44 / 4 = 11"   */
    printf("19 / 2.0 = %g\n", e);  /* Will output "19 / 2.0 = 9.5"    */
    printf("18.0 / 2 = %g\n", f);  /* Will output "18.0 / 2 = 9"      */
    printf("255 / 2.0 = %g\n", g); /* Will output "255 / 2.0 = 127.5" */
    printf("45.0 / 4 = %g\n", h);  /* Will output "45.0 / 4 = 11.25"  */

    return 0;
}
```

O **operador de módulo (%)** é usado para obter o resto da divisão entre dois valores.

```c
#include <stdio.h>

int main (void) {
    int a = 25 % 2;    /* a holds value 1  */
    int b = 24 % 2;    /* b holds value 0  */
    int c = 155 % 5;   /* c holds value 0  */
    int d = 49 % 25;   /* d holds value 24 */

    printf("25 % 2 = %d\n", a);     /* Will output "25 % 2 = 1"    */
    printf("24 % 2 = %d\n", b);     /* Will output "24 % 2 = 0"    */
    printf("155 % 5 = %d\n", c);    /* Will output "155 % 5 = 0"   */
    printf("49 % 25 = %d\n", d);    /* Will output "49 % 25 = 24"  */

    return 0;
}
```

OPERADORES DE INCREMENTO / DECREMENTO

Os operadores de incremento e decremento podem ser usados na forma prefixa ou pósfixa.

```
int a = 1;
int b = 1;
int tmp = 0;

tmp = ++a;        /* increments a by one, and returns new value; a == 2, tmp == 2  */
tmp = a++;        /* increments a by one, but returns old value; a == 3, tmp == 2 */
tmp = --b;        /* decrements b by one, and returns new value; b == 0, tmp == 0 */
tmp = b--;        /* decrements b by one, but returns old value; b == -1, tmp == 0 */
```

OPERADOR DE CAST

Trata-se de uma conversão explícita entre tipos.

```c
int x = 3;
int y = 4;
printf("%f\n", (double)x / y); /* Outputs "0.750000". */
```

No exemplo, estamos convertendo x de int para double e o resultado é double também.

OPERADORES DE ATRIBUIÇÃO

Atribui um valor ou resultado de expressão (lado direito) para uma variável (lado esquerdo).

```c
int x = 5;       /* Variable x holds the value 5. Returns 5. */
char y = 'c';    /* Variable y holds the value 99. Returns 99
                  * (as the character 'c' is represented in the ASCII table with 99).
                  */
float z = 1.5;   /* variable z holds the value 1.5. Returns 1.5. */
char const* s = "foo"; /* Variable s holds the address of the first character of the string 'foo'.
*/
```

Vários operadores matemáticos podem ter uma atribuição composta:

```c
a += b  /* equal to: a = a + b */
a -= b  /* equal to: a = a - b */
a *= b  /* equal to: a = a * b */
a /= b  /* equal to: a = a / b */
a %= b  /* equal to: a = a % b */
```

OPERADORES LÓGICOS

O **operador E (&&)** é usado para testar se ambos os testes são verdadeiros.

```
0 && 0  /* Returns 0. */
0 && 1  /* Returns 0. */
2 && 0  /* Returns 0. */
2 && 3  /* Returns 1. */
```

O **operador OU (||)** é usado para testar algum dos testes é verdadeiro.

```
0 || 0  /* Returns 0. */
0 || 1  /* Returns 1.  */
2 || 0  /* Returns 1.  */
2 || 3  /* Returns 1.  */
```

CAPÍTULO 5: BOOLEAN

https://books.goalkicker.com/CBook/

BOOLEAN

A linguagem C não tem um tipo lógico nativo (**1 ou 0**). Para isto, precisamos importar a biblioteca **stdbool.h**

```c
#include <stdio.h>
#include <stdbool.h>

int main(void) {
    bool x = true;  /* equivalent to bool x = 1; */
    bool y = false; /* equivalent to bool y = 0; */
    if (x)  /* Functionally equivalent to if (x != 0) or if (x != false) */
    {
        puts("This will print!");
    }
    if (!y) /* Functionally equivalent to if (y == 0) or if (y == false) */
    {
        puts("This will also print!");
    }
}
```

USANDO #DEFINE

Podemos definir que 1 será interpretado como true e 0 como false através da macro #define.

```c
#include <stdio.h>

#define bool int
#define true 1
#define false 0

int main(void) {
    bool x = true;  /* Equivalent to int x = 1; */
    bool y = false; /* Equivalent to int y = 0; */
    if (x) /* Functionally equivalent to if (x != 0) or if (x != false) */
    {
        puts("This will print!");
    }
    if (!y) /* Functionally equivalent to if (y == 0) or if (y == false) */
    {
        puts("This will also print!");
    }
}
```

PODEMOS USAR O TIPO _BOOL

O tipo _Bool foi adicionado como padrão a partir da versão C99 e pode armazenar os valores 0 e 1.

```c
#include <stdio.h>

int main(void) {
    _Bool x = 1;
    _Bool y = 0;
    if(x) /* Equivalent to if (x == 1) */
    {
        puts("This will print!");
    }
    if (!y) /* Equivalent to if (y == 0) */
    {
        puts("This will also print!");
    }
}
```

PODEMOS DEFINIR NOVOS TIPOS ATRAVÉS DO

```c
#if __STDC_VERSION__ < 199900L
typedef enum { false, true } bool;
/* Modern C code might expect these to be macros. */
# ifndef bool
#  define bool bool
# endif
# ifndef true
#  define true true
# endif
# ifndef false
#  define false false
# endif
#else
# include <stdbool.h>
#endif

/* Somewhere later in the code ... */
bool b = true;
```

O código acima permite que compiladores com diferentes versões de C executem o mesmo código através do uso de macros.

OBRIGADO!

RAFAELVC2@GMAIL.COM

https://books.goalkicker.com