2. Constantes e Variáveis

Neste capitulo veremos como os dados constantes e variáveis são manipulados pela linguagem C. O que são constantes inteiras, reais, caracteres e strings. Quais são as regras de atribuição de nomes a variáveis e quais são os tipos de dados que O C pode manipular. Veremos também como são declaradas as variáveis e as constantes simbólicas usadas em um programa.

2.1 Constantes

O C possui quatro tipos básicos de constantes: **inteiras**, de **ponto flutuante**, **caracteres** e ***strings***. Constantes inteiras e de ponto flutuante representam números de um modo geral. Caracteres e strings representam letras e agrupamentos de letras (palavras).

2.1.1 Constantes inteiras

Uma constante inteira é um número de valor inteiro. De uma forma geral, constantes inteiras são seqüências de dígitos que representam números inteiros. Números inteiros podem ser escritos no formato **decimal** (base 10), **hexadecimal** (base 16) ou **octal** (base 8).

Uma constante inteira **decimal** é formada por uma seqüência de dígitos decimais: `0, 1, 2, 3, 4, 5, 6, 7, 8, 9`. Se a constante tiver dois ou mais dígitos, o primeiro **não** pode ser `0`. Na verdade, pode ser `0` mas o compilador considerará esta constante como **octal** e não decimal.

Exemplo: A seguir são mostradas algumas constantes inteiras decimais válidas.

```
0   3   -45   26338   -7575   1010
```

Exemplo: Algumas constantes inteiras decimais inválidas.

```
1.          (ponto)
1,2         (vírgula)
045         (primeiro dígito é 0: não é constante decimal)
212-22-33   (caracter ilegal: -)
```

Uma constante inteira **hexadecimal** é formada por uma seqüência de dígitos decimais: `0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F` (ou `a, b, c, d, e`). Uma constante

hexadecimal **deve** começar por `0x`. Neste caso, os dígitos hexadecimais podem ser minúsculos ou maiúsculos.

Exemplo: A seguir são mostrados algumas constantes inteiras hexadecimais **válidas**.

```
0x0   0x3   0x4f5a   0x2FFE   OxABCD   0xAaFf
```

Exemplo: Algumas constantes inteiras hexadecimais **inválidas**.

```
0x3.       (ponto)
0x1,e      (vírgula)
0x ff      (espaço)
FFEE       (não começa com 0x: não é constante hexadecimal)
0Xfg34     (caracter ilegal: g)
```

Uma constante inteira **octal** é formada por uma seqüência de dígitos octais: `0, 1, 2, 3, 4, 5, 6, 7`. A constante octal **deve** ter o primeiro digito `0` para que o compilador a identifique como tal

Exemplo: A seguir são mostrados algumas constantes octais **válidas**.

```
00   -03   045   02633   07575   -0101
```

Exemplo: Algumas constantes inteiras octais **inválidas**.

```
010.       (ponto)
01,2       (vírgula)
0 4        (espaço)
45         (primeiro digito não é 0: não é constante octal)
01784      (caracter ilegal: 8)
```

2.1.2 Constantes de ponto flutuante

Números **reais** (não inteiros) são representados em base 10, por números com um ponto decimal e (opcionalmente) um expoente. Um número ponto flutuante **deve** ter um ponto decimal que não pode ser substituído por uma vírgula. Um número de ponto flutuante pode ser escrito em notação científica. Neste caso o `x10` é substituído por `e` ou `E`. O número `1.23e4` representa `1.23 x` 10^4 ou `12300`.

Exemplo: Números de ponto flutuante válidos.

```
0.234   125.65   .93   1.23e-9   -1.e2   10.6e18   -.853E+67
```

A forma de representação de um número real em C é bastante flexível.

Exemplo: O número 314 pode ser representado por qualquer uma das seguintes formas:

```
314.  3.14e2     +3.14e+2   31.4e1     .314E+3    314e0
```

2.1.3 Constantes caracteres

Uma constante caracter é uma letra ou símbolo colocado entre **aspas simples**.

Exemplo: Abaixo estão representados algumas constantes caracteres.

```
'a'   'b'   'X'   '&'   '{'   ' '
```

Embora sejam **visualizados** como letras e símbolos as constantes caracteres são armazenadas internamente pelo computador como um número **inteiro** entre 0 e 255. O caracter `'A'` por exemplo, tem valor `65`. Os valores numéricos dos caracteres estão padronizados em uma tabela chamada de *American Standard Code for Information Interchange Table* ou simplesmente *tabela ASCII*. **Veja apêndice B.**

Certos codigos de controle da tabela ASCII (como o *line feed*) ou caracteres especiaiss (como `'`) possuem representação especial no C. Esta representacao chama-se **seqüência de escape** representada por uma *barra invertida* (\) e um *caracter*. Sequencias de escape são interpretadas como caracteres simples. Abaixo segue uma lista das principais sequencias de escape usadas no C.

Controle/Caracter	Sequencia de escape	Valor ASCII
nulo (null)	\0	00
campainha (bell)	\a	07
retrocesso (backspace)	\b	08
tabulacao horizontal	\t	09
nova linha (new line)	\n	10

tabulacao vertical	\v	11
alimentacao de folha (form feed)	\f	12
retorno de carro (carriage return)	\r	13
aspas (")	\"	34
apostrofo (')	\'	39
interrogacao (?)	\?	63
barra invertida (\)	\\	92

2.1.4 Constantes strings

Uma constante string consiste de um conjunto de caracteres colocados entre **aspas duplas**. Embora as instruções do C usem apenas os caracteres do conjunto padrão ASCII, as constantes caracter e string podem conter caracteres do conjunto estendido ASCII: é, ã, ç, ü, ...

Exemplo: Abaixo seguem algumas constantes strings válidas.

```
"Oba!"
"Caxias do Sul"
"A resposta é: "
"João Carlos da Silveira"
"a"
"isto é uma string"
```

2.2 Identificadores

Identificadores são os **nomes** das variáveis e funções usadas no programa. Por exemplo `raiz` e `MAX` são nomes de variáveis utilizadas no programa `e0101.cpp`.

2.2.1 Regras de sintaxe

Os identificadores devem seguir as seguintes regras de construção:

- Os identificadores **devem** começar por uma letra (`a - z` , `A - Z`) ou um *underscore* (`_`).
- O resto do identificador deve conter apenas letras, *underscores* ou dígitos (`0 - 9`). **Não pode** conter outros caracteres. Em C, os identificadores podem ter até 32 caracteres.
- Em C, letras maiúsculas são **diferentes** de letras minúsculas: Por exemplo: `MAX`, `max`, `Max` são nomes diferentes para o compilador. Esta propriedade é chamada de *case sensibility*.

Exemplo: os nomes abaixo são válidos:

```
abc, y24, VetorPontosMovimentoRobo, nota_1, TAM_MAX.
```

Exemplo: os nomes abaixo não são válidos:

```
3dia, vetor-1, pao&leite, iteração.
```

2.2.2 Palavras reservadas

Existem certos nomes que **não podem** ser usados como identificadores. São chamadas as *palavras reservadas* e são de uso restrito da linguagem C (comandos, estruturas, declarações, etc.). O conjunto de palavras reservadas usadas em C é o seguinte:

asm	auto	break	case	cdecl	char
class	const	continue	_cs	default	delete
do	double	_ds	else	enum	_es
extern	_export	far	_fastcall	float	for
friend	goto	huge	if	inline	int
interrupt	_loadds	long	near	new	operator
pascal	private	protected	public	register	return
_saveregs	_seg	short	signed	sizeof	_ss
static	struct	switch	template	this	typedef
union	unsigned	virtual	void	volatile	while

Exemplo: Não é possível declarar estes conjunto de variáveis:

```
do, re, mi, fa, sol, la, si
old, new
```

Dos conjuntos acima, `do` e `new` são palavras reservadas.

2.3 Tipos de dados

Em C, como na maioria das linguagens, os dados são divididos tipos: inteiro, real, caracter, etc. Esta divisão se deve basicamente ao número de *bytes* reservados para cada dado. Cada tipo de dado possui um intervalo de valores permitidos.

2.3.1 Tipos básicos

Abaixo segue uma lista dos tipos básicos de dados permitidos em C. Os tipos `char` e `int` são inteiros e os tipos `float` e `double` são de ponto flutuante.

Tipo	Tamanho	Intervalo	Uso
`char`	1 byte	-128 a 127	número muito pequeno e caracter ASCII
`int`	2 bytes	-32768 a 32767	contador, controle de laço
`float`	4 bytes	3.4e-38 a 3.4e38	real (precisão de 7 dígitos)
`double`	8 bytes	1.7e-308 a 1.7e308	científico (precisão de 15 dígitos)

2.3.2 Declaração de variáveis

Para que se possa usar uma variável em um programa, **é necessário** fazer uma *declaração de variável* antes. A declaração de variáveis simplesmente informa ao processador quais são os nomes utilizados para armazenar dados variáveis e quais são os tipos usados. Deste modo o processador pode *alocar* (reservar) o espaço necessário na memória para a manipulação destas variáveis. É possível declarar mais de uma variável ao mesmo tempo, basta separá-las por vírgulas (`,`).

Sintaxe: A sintaxe para declaração de variáveis é a seguinte:

`tipo variavel_1 [, variavel_2, ...] ;`

Onde *`tipo`* é o tipo de dado e *`variavel_1`* é o nome da variável a ser declarada. Se houver mais de uma variável, seus nomes são separados por vírgulas.

Exemplo: Declaração das variáveis:

```
int i;
int x,y,z;
char letra;
float nota_1,nota_2,media;
double num;
```

No exemplo acima, `i`, `x`, `y` e `z` foram declaradas variáveis inteiras. Assim elas podem armazenar valores inteiros de -32768 até 32767. Do mesmo modo `letra` foi declarada como variável caracter podendo receber valores de -128 até 127 ou caracteres do conjunto padrão ASCII. As variáveis `nota_1`,

`nota_2` e `media` foram declaradas como ponto flutuante tipo float e `num` como ponto flutuante tipo double.

A declaração de variáveis é feita, em geral, **dentro** de uma rotina. Por exemplo, a rotina principal `main()`. Deste modo se diz que está se fazendo uma declaração de variáveis *locais*. Variáveis locais podem ser referenciadas apenas dentro da rotina dentro da qual foi declarada, neste caso a rotina `main()`.

Exemplo: Observe o uso da declaração de variáveis no trecho de programa abaixo:

```
void main(){
  float raio, area;     // declaracao de variaveis
  raio = 2.5;
  área = 3.14 * raio * raio;
}
```

No exemplo acima, as variáveis `area` e `raio` foram declaradas como variáveis locais tipo `float`. Assim o processador faz a alocação de dois espaços (endereços) de 4 bytes cada para armazenar as informações, um para cada variável. Na terceira linha, o processador coloca no endereço alocado para `raio` o valor `2.5`. Depois, na quarta linha, o processador coloca o resultado da conta (`19.625`) no endereço de `área`.

É possível fazer a declaração de variáveis **fora** de uma rotina. Neste caso diz-se que se fez a declaração de variáveis *globais*. O uso de variáveis globais é explicado na sessão ?.?.

2.3.3 Tipos modificados

Além dos tipos de dados citados acima existem outros tipos de dados ditos **modificados**. Em C existem dois modificadores: o modificador `long` e o modificador `unsigned`. Tipicamente o modificador `long` aumenta o número de bytes usados para o registro do número. Por conseqüência o intervalo de validade do número fica aumentado significativamente. O modificador `unsigned,` usado somente em inteiros, permite que um **bit** usado para guardar o sinal do número seja usado para guardar o valor do número. Em conseqüência disto o intervalo do número fica dobrado, porém somente permite o uso de números positivos.

Tipo	Tamanho (bytes)	Intervalo
unsigned char	1	0 a 255
unsigned int	2	0 a 65 535
long int	4	-2 147 483 648 a 2 147 483 647
unsigned long int	4	0 a 4 294 967 295

```
long double            10              3.4e-4932 a 1.1e4932
```

2.3.4 Strings.

Uma string é um conjunto ordenado de caracteres que pode ser armazenado sob forma de um vetor um ponteiro. Estas estruturas de dados serão vistas em detalhe nos capitulo ? e ?. Por enquanto, nos basta saber como declarar e armazenar um conjunto caracter em uma variável.

Sintaxe: Para declararmos uma variável para receber um conjunto caracter devemos escrever:

```
char* var;
```

Exemplo: No exemplo seguinte a variável `nome` foi declarada como conjunto caracter e em seguida recebe uma constante string.

```
char* nome;
nome = "João Carlos de Oliveira Andrade";
```

2.3.5 Inicialização de variáveis.

Quando se faz a **declaração** de uma variável está se determinando que tipo de dado ela vai receber. É possível, em C, declarar uma variável e já armazenar nela um valor inicial. Chamamos este procedimento de **inicialização** de uma variável.

Sintaxe: A sintaxe para a inicialização de variáveis é:

```
tipo var_1 = valor_1 [, var_2 = valor_2, ...] ;
```

Onde *tipo* é o tipo de dado, *var_1* é o nome da variável a ser inicializada e *valor_1* é o valor inicial da variável.

Exemplo: Inicialização de variáveis:

```
int i = 0, j = 100;
float num = 13.5;
char* titulo = " Programa Teste ";
```

No exemplo acima, `i` e `j` foram declaradas variáveis tipo `int`. O valor inicial de `i` é `0` e o de `j` é `100`. Do mesmo modo `num` foi declarada como variável `float` com valor inicial de `13.5`. Tambem a variavel `titulo` foi declarada como um conjunto caracter e recebeu como conteudo inicial a string `" Programa Teste "`.

2.3.6 Conversão de tipo (*Casting*)

Algumas vezes queremos, momentaneamente, modificar o tipo de dado representado por uma variável, isto é, queremos que o dado seja apresentado em um tipo diferente do qual a variável foi inicialmente declarada. Por exemplo: declaramos uma variável como `int` e queremos, momentaneamente, que seu conteúdo seja apresentado como `float`. Este procedimento é chamado de **conversão de tipo** ou *casting* (moldagem, em inglês).

Sintaxe: A sintaxe da instrução de conversão de tipo é:
`(tipo) variável`

onde `tipo` é o nome do tipo ao qual queremos converter o dado armazenado em `variável`.

Exemplo: observe a conversão de tipo feita no exemplo abaixo:

```
int num;
float valor = 13.0;
num = (int)valor % 2;
```

No exemplo acima a variável `valor` foi declarada inicialmente como sendo do tipo `float` recebendo o valor inicial `13.0`. Logo em seguida o conteúdo de valor é convertido para o tipo `int` para realizar a operação módulo (%) com o inteiro 2. Aqui a conversão é necessária pois a operação módulo somente pode ser feita com inteiros. É importante salientar que a conversão de tipo é feita com o **dado** armazenado em uma variável mas a **variável** continua tendo o seu tipo original. No exemplo acima a variável `valor` e os dados nela armazenados continuam sendo do tipo `float` após a conversão.

Veremos na **seção 3.1** uma explicação mais detalhada do uso da conversão de tipos.

2.4 Constantes Simbólicas

Muitas vezes identificamos uma constante numérica por um símbolo: Pi = 3,14159 por exemplo. Podemos definir um nome simbólico para esta constante, isto é, podemos definir uma **constante simbólica** que represente valor.

2.4.1 Constantes definidas pelo programador

O programador pode definir constantes simbólicas em qualquer programa.

Sintaxe: A sintaxe da instrução de definição de uma constante simbólica é:

```
#define nome valor
```

Onde `#define` é uma diretiva de compilação que diz ao compilador para trocar as ocorrências do texto *nome* por *valor*. Observe que **não há ;** no final da instrução pois trata-se de um comando para o compilador e não para o processador. A instrução `#define` **deve** ser escrita **antes** da instrução de declaração da rotina principal.

Exemplo: a seguir definimos algumas constantes simbólicas.

```
#define PI  3.14159
#define ON  1
#define OFF 0
#define ENDERECO 0x378
void main(){
...
```

No exemplo acima, definimos `PI` como `3.14159`. Isto significa que todas as ocorrências do texto `PI` será trocado por `3.14159`. Assim se escrevemos uma instrução:

```
área = PI * raio * raio;
```

o compilador vai interpretar esta instrução como se fosse escrita assim:

```
área = 3.14159 * raio * raio;
```

Poderíamos escrever estas instruções assim:

```
float pi = 3.14159;
área = pi * área * área;
```

porém este tipo de instrução tem duas **desvantagens**: Primeiro, reserva 4 bytes de memória desnecessariamente. Segundo, esta instrução é executada mais lentamente pois o processador precisa acessar a memória para verificar qual é o valor de `pi`.

Observe também que no exemplo definimos os nomes simbólicos com letras maiúsculas. Isto **não é necessário**, podemos perfeitamente definir nomes simbólicos usando letras minúsculas, porém faz parte do jargão dos programadores C usar letras maiúsculas para definir constantes simbólicas.

O uso da diretiva `#define` não se restringe apenas ao apresentado acima, podemos usá-la para definir **macro instruções**. Não veremos o uso de macros neste texto, procure mais detalhes na bibliografia recomendada.

2.4.2 Constantes pré-definidas

Em alguns compiladores C, algumas constantes simbólicas já estão pré-definidas. Estas constantes em geral definam alguns valores matemáticos (π, $\pi/2$, e, etc.), limites de tipos etc. A seguir segue uma tabela contendo algumas (existem muitas outras) constantes simbólicas pré-definidas no compilador Turbo C++ da Borland.

Biblioteca	Constante	Valor	Significado
math.h	M_PI	3.14159...	π
math.h	M_PI_2	1.57079...	$\pi/2$
math.h	M_PI_4	0,78539...	$\pi/4$
math.h	M_1_PI	0,31830...	$1/\pi$
math.h	M_SQRT2	1,41421...	$\sqrt{2}$
conio.h	BLACK	0	valor da cor (preto)
conio.h	BLUE	1	valor da cor (azul)
conio.h	GREEN	2	valor da cor (verde)
conio.h	CYAN	3	valor da cor (cyan)
conio.h	RED	4	valor da cor (vermelho)
conio.h	MAGENTA	5	valor da cor (magenta)
limits.h	INT_MAX	32767	limite superior do tipo int
limits.h	INT_MIN	-32768	limite inferior do tipo int

Cada uma das constantes acima esta definida em uma biblioteca. Uma biblioteca, em C, é um arquivo pré-compilado chamado arquivo *header* (cabeçalho, em inglês). Em cada biblioteca estão agrupadas constantes e funções semelhantes (veja seção 3.7.2). Por exemplo, constantes e funções matemáticas estão guardadas na biblioteca `math.h` (*mathematical functions*), constantes e funções de manipulação teclado e monitor estão guardadas na biblioteca `conio.h` (*console input and output*). Para que se possa usar a constante simbólica em um programa é preciso **incluir** a biblioteca na compilação do programa.

Sintaxe: A sintaxe de inclusão de bibliotecas é a seguinte:

```
#include <nome_bib>
```

onde *nome_bib* é o nome da biblioteca que se deseja incluir. Esta instrução deve ser escrita antes do programa principal.

Exemplo: O programa abaixo usa a constante predefinida M_PI para calcular a área de um disco circular.

```
#include <math.h>
void main(){
  float area, raio = 5.0;
  área = M_PI * raio * raio;
}
```