Leitura 2- Tipos de Dados Básicos

Paulo Vieira

Agenda

Sintaxe Básica

Tipos de Dados

Variáveis

Constantes

Atribuição

Operações com inteiros

Formatação, multiplas entradas|saídas

Formatos entrada|saída

Sintaxe básica

Identificadores:

- Um identificador em C é um nome que pode ser usado para identificar uma variável, uma função, ou qualquer outro item definido pelo utilizador
- Um identificador começa com uma letra (a-z,A-Z) ou underscore (_), e pode seguir-se (repetindo um numero finito de vezes) uma letra, um underscore ou um digito (0-9)

Keywords - Palavras Chaves (Reservadas). Não podem ser usadas como constantes ou variáveis ou qualquer outro nome de identificador

auto	else	long	switch
break	enum	register	typedef
case	extern	return	union
char	float	short	unsigned
const	for	signed	void
continue	goto	sizeof	volatile
default	if	static	while
do	int	struct	_Packed
double			

Espaços em branco:

- Uma linha contendo apenas espaços em branco, possivelmente com um comentário, é conhecida como linha em branco e um compilador C ignora-a totalmente.
- Os espaços em branco servem para separar declarações

Exemplo:

```
int age;
fruit = apples + oranges;   // get the total fruit
```

Tipos de dados

- Os tipos de dados em C referem-se a um sistema extenso usado para declarar variáveis ou funções de diferentes tipos.

- O tipo de uma variável determina quanto espaço ela ocupa no armazenamento e como o padrão de bits armazenado é interpretado

Os tipos em C podem ser classificados como:

Sr.No.	Types & Description
1	**Basic Types** They are arithmetic types and are further classified into: (a) integer types and (b) floating-point types.
2	**Enumerated types** They are again arithmetic types and they are used to define variables that can only assign certain discrete integer values throughout the program.
3	**The type void** The type specifier *void* indicates that no value is available.
4	**Derived types** They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types.

Tipos de dados

- Os array type e os structure type são chamados de tipos agregados.

- O tipo de uma função especifica o tipo do valor de retorno da função.
- Há tipos e outros tipos que não o são

- O operador **sizeof** retorna o tamanho de um objecto ou tipo em bytes operator. (definido no header limit.h)

- Tipos floats estão definidos no header float.h

Tipos inteiros, integer types

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes or (4bytes for 32 bit OS)	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

Tipos floats, standard floating-point types

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

Tipos de dados

O tipo void especifica que nenhum valores é disponibilizado.

É usado nas 3 situações referidas →

Sr.No.	Types & Description
1	**Function returns as void** There are various functions in C which do not return any value or you can say they return void. A function with no return value has the return type as void. For example, **void exit (int status);**
2	**Function arguments as void** There are various functions in C which do not accept any parameter. A function with no parameter can accept a void. For example, **int rand(void);**
3	**Pointers to void** A pointer of type void * represents the address of an object, but not its type. For example, a memory allocation function **void *malloc(size_t size);** returns a pointer to void which can be casted to any data type.

Tipos de dados

Exemplos de código

```c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <float.h>

int main(int argc, char** argv) {

    printf("CHAR_BIT    :   %d\n", CHAR_BIT);
    printf("CHAR_MAX    :   %d\n", CHAR_MAX);
    printf("CHAR_MIN    :   %d\n", CHAR_MIN);
    printf("INT_MAX     :   %d\n", INT_MAX);
    printf("INT_MIN     :   %d\n", INT_MIN);
    printf("LONG_MAX    :   %ld\n", (long) LONG_MAX);
    printf("LONG_MIN    :   %ld\n", (long) LONG_MIN);
    printf("SCHAR_MAX   :   %d\n", SCHAR_MAX);
    printf("SCHAR_MIN   :   %d\n", SCHAR_MIN);
    printf("SHRT_MAX    :   %d\n", SHRT_MAX);
    printf("SHRT_MIN    :   %d\n", SHRT_MIN);
    printf("UCHAR_MAX   :   %d\n", UCHAR_MAX);
    printf("UINT_MAX    :   %u\n", (unsigned int) UINT_MAX);
    printf("ULONG_MAX   :   %lu\n", (unsigned long) ULONG_MAX);
    printf("USHRT_MAX   :   %d\n", (unsigned short) USHRT_MAX);

    return 0;
}
```

```
$gcc -o main *.c -lm

$main
CHAR_BIT    :   8
CHAR_MAX    :   127
CHAR_MIN    :   -128
INT_MAX     :   2147483647
INT_MIN     :   -2147483648
LONG_MAX    :   9223372036854775807
LONG_MIN    :   -9223372036854775808
SCHAR_MAX   :   127
SCHAR_MIN   :   -128
SHRT_MAX    :   32767
SHRT_MIN    :   -32768
UCHAR_MAX   :   255
UINT_MAX    :   4294967295
ULONG_MAX   :   18446744073709551615
USHRT_MAX   :   65535
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <float.h>

int main(int argc, char** argv) {

    printf("Storage size for float : %d \n", sizeof(float));
    printf("FLT_MAX     :   %g\n", (float) FLT_MAX);
    printf("FLT_MIN     :   %g\n", (float) FLT_MIN);
    printf("-FLT_MAX    :   %g\n", (float) -FLT_MAX);
    printf("-FLT_MIN    :   %g\n", (float) -FLT_MIN);
    printf("DBL_MAX     :   %g\n", (double) DBL_MAX);
    printf("DBL_MIN     :   %g\n", (double) DBL_MIN);
    printf("-DBL_MAX     :  %g\n", (double) -DBL_MAX);
    printf("Precision value: %d\n", FLT_DIG );

    return 0;
}
```

```
$gcc -o main *.c -lm

$main
Storage size for float : 4
FLT_MAX     :   3.40282e+38
FLT_MIN     :   1.17549e-38
-FLT_MAX    :   -3.40282e+38
-FLT_MIN    :   -1.17549e-38
DBL_MAX     :   1.79769e+308
DBL_MIN     :   2.22507e-308
-DBL_MAX     :  -1.79769e+308
Precision value: 6
```

Tipos de dados simples e Variáveis

Sr.No.	Type & Description
1	**char** Typically a single octet(one byte). It is an integer type.
2	**int** The most natural size of integer for the machine.
3	**float** A single-precision floating point value.
4	**double** A double-precision floating point value.
5	**void** Represents the absence of type.

Uma variável nada mais é do que um nome dado a uma área de armazenamento que os programas podem manipular.

Cada variável em C têm:

<tipo> <identificador> = <valor>

- <tipo> um tipo específico, que determina o tamanho e o layout da memória da variável;
- <identificador> um nome;
- <valor> a faixa de valores que podem ser armazenados nessa memória;

Veem associados um conjunto de operações que podem ser aplicadas à variável.

Tipos de dados simples e Variáveis

```
type variable_list;
```

char, w_char, int, float, double, bool, or any user-defined object

inicializador

```
type variable_name = value;
```

```
int    i, j, k;
char   c, ch;
float  f, salary;
double d;
```

```
extern int d = 3, f = 5;    // declaration of d and f.
int d = 3, f = 5;           // definition and initializing d and f.
byte z = 22;                // definition and initializes z.
char x = 'x';               // the variable x has the value 'x'.
```

A palavra-chave **extern** serve para declarar uma variável em qualquer lugar. Embora se possa declarar uma variável várias vezes num programa C, ela pode ser definida apenas uma vez em um arquivo, função ou bloco de código.

Para definição sem um inicializador: variáveis de determinado tipo por valor NULL, outras de outro tipo têm por defeito um valor, e há tipos ainda cujo valor é indefinido .

Declaração e definição inicialização|uso de variáveis e funções

```c
#include <stdio.h>

// Variable declaration:
extern int a, b;
extern int c;
extern float f;

int main () {

   /* variable definition: */
   int a, b;
   int c;
   float f;

   /* actual initialization */
   a = 10;
   b = 20;

   c = a + b;
   printf("value of c : %d \n", c);

   f = 70.0/3.0;
   printf("value of f : %f \n", f);

   return 0;
}
```

extern

```
$gcc -o main *.c -lm
$main
value of c : 30
value of f : 23.333334
```

O exemplo para funções

```c
// function declaration
int func();

int main() {

   // function call
   int i = func();
}

// function definition
int func() {
   return 0;
}
```

Lvalues and Rvalues in C

Existem dois tipos de expressões em C

lvalue - são as expressões que se referem a um local da memória. Um lvalue pode aparecer como o lado esquerdo ou direito de uma atribuição.

rvalue - O termo rvalue refere-se a um valor de dados que é armazenado em algum endereço na memória. Um rvalue é uma expressão que não pode ter um valor atribuído a ela, o que significa que um rvalue pode aparecer no lado direito, mas não no lado esquerdo de uma atribuição.

Exemplo: as Variáveis são lvalues. Os literais numéricos são rvalues

```
int g = 20; // valid statement

10 = 20; // invalid statement; would generate compile-time error
```

Constantes

Constantes referem-se a valores fixos que o programa não pode alterar durante sua execução. Esses valores fixos também são chamados de literais.

Exemplo de literais

```
212         /* Legal */
215u        /* Legal */
0xFeeL      /* Legal */
078         /* Illegal: 8 is not an octal digit */
032UU       /* Illegal: cannot repeat a suffix */
```

```
85          /* decimal */
0213        /* octal */
0x4b        /* hexadecimal */
30          /* int */
30u         /* unsigned int */
30l         /* long */
30ul        /* unsigned long */
```

```
3.14159       /* Legal */
314159E-5L    /* Legal */
510E          /* Illegal: incomplete exponent */
210f          /* Illegal: no decimal or exponent */
.e55          /* Illegal: missing integer or fraction */
```

‘x’ '\u02C0'
\n
\t

```
"hello, dear"

"hello, \

dear"

"hello, " "d" "ear"
```

Constantes

As constantes podem ser de qualquer um dos tipos de dados básicos, constante:
inteira,flutuante, caractere ou um literal de string.
Também existem constantes de enumeração.

```c
#include <stdio.h>

#define LENGTH 10
#define WIDTH  5
#define NEWLINE '\n'

int main() {

   int area;

   area = LENGTH * WIDTH;
   printf("value of area : %d", area);
   printf("%c", NEWLINE);

   return 0;
}
```

```
$gcc -o main *.c -lm
$main
value of area : 50
```

Existem duas maneiras de definir constantes no C:

- Usando o preprocessador #define.
- Usando a keyword const.

```c
#include <stdio.h>

int main() {

   const int  LENGTH = 10;
   const int  WIDTH = 5;
   const char NEWLINE = '\n';
   int area;

   area = LENGTH * WIDTH;
   printf("value of area : %d", area);
   printf("%c", NEWLINE);

   return 0;
}
```

```
$gcc -o main *.c -lm
$main
value of area : 50
```

Atribuição

```
tipo var_1[var_2,...,var_n];  // declaração de variaveis



main() {
    // declaração de variáveis

    // instrução 1;
     ……
    // insdtrução n;
}
```

Atribuição

variável = expressão

int num; // num é declarado, valor lixo ou por defeito (depende do compilador)

num = -17; // num recebe o valor -17

int a=10, b, c=-123; // a, b, c tomam valores resp 10, lixo|defeito, -123

Int a=b=b=d=5; /* tb é possível declarar e atribuir desta forma se as variáveis têm todas o mesmo tipo e são inicializadas com o mesmo valor */

O valor de uma variável pode ser alterado ao longo do programa, o valor de uma constante não pode

Operações com inteiros

Operações básica: +, -, *, /, %

a/b = divisão inteira

21/4=5

100/25=4

11/3=3

a%b=resto da divisão de a por b

21 %4= 1

100%25=0

11%3=2

formato de entrada|saída de dados inteiros, dos inteiros scanf|printf, %d

```
#include <stdio.h>
int main(){
    int num;
    printf("Introduza um numero: ");
    scanf("%d",&num);
   printf("o numero introduzido foi %d\n", num);
   return 0;
}
```

formatação, múltiplas entradas|saídas dados

formato de entrada|saída de dados inteiros, dos inteiros scanf|printf, %d

```
#include <stdio.h>
int main(){
     int n1,n2;
    printf(“Introduza dois numeros: ”);
     scanf(“%d%d”,&n1,&n2);
   printf(“o resultado de %d+%d = %d\n”, n1,n2,n1+n2);
   return 0;
}
```

Formatação entrada|saída de dados

%c (character)

%d (inteiro)

%f (float)

%e (notação científica)

%E (notação científica)

Exemplo declaração e inicialização float|double

```
float PI=3.1415;
double erro=0.000001;
float total=0.0;
```

floats

123456.789

parte inteira

parte decimal

123.46 E 78

base

expoente

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes or (4bytes for 32 bit OS)	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

Formatos entrada|saída de dados

```
# include <stdio.h>
main(){
        float quilos= 1.0E3;
        double gramas= 1.0e6;
        printf("v_kg%E, v_gramas
        %e",quilos,gramas);
        return 0;
}
```

tipo	formato	observação
char	%c	um unico caracter
int	%d ou %i	um inteiro (base decimal)
int	%o	um inteiro (base octal)
int	%x ou %X	um inteiro base hexadecimal
short int	%hd	um short inteiro (base decimal)
long int	%ld	um long inteiro (base decimal)
unsigned short int	%hu	short interiro positivo
unsigned int	%u	inteiro positivo
unsigned long int	%lu	long inteiro positivo
float	%f ou %e ou %E	
double	%f ou %e ou %E	

Operações com reais: +, - , *, / (divisão real)

FIM