4. Variáveis & E/S

DIM0321

2015.1

Outline

Regras definição

Definição

- Uma variável possui um tipo de informação que ela guarda
- Declaração: `<tipo> <nome>` ou `<tipo> <nome> = <expressão>`
- Nomes possíveis sequência de letras, números e sublinhado, começando com letra ou sublinhado.
  `[a-zA-z_][a-zA-Z_0-9]*`
- Letras maiúsculas e minúsculas são diferentes (C é **case-sensitive**)

Costume

1. Nomes de variáveis usam minúsculas
2. Constantes simbólicas usam maiúsculas

Declaração

Regra

- Variáveis devem ser inicializadas no início de um bloco { }

Primeira opção

```
1  #define NUM 10
2  int lower, upper, step;
3  char c, line[1000], v[NUM];
```

Alternativa

```
1  #define NUM 10
2  int lower;
3  int upper;
4  int  step;
5  char c;
6  char line[1000];
7  char v[NUM];
```

- Conveniente para adicionar comentários para cada declaração.

Declaração com inicialização

- É possível inicializar variáveis no momento da declaração

```
char newline = '\n';
int i = 0;
int limit = MAXLINE + 1;
float e = 1e-2;
char *names[4] = {"foo", "bar", "baz", "bill" };
const double pi = 3.14159;
const char msg[] = "alert";
```

Atribuição

Operador de atribuição

- = é o operador de atribuição de C
- = $\neq$ == ! Programas erram silenciosamente com conversões explícitas.

Valor retornado

```c
#include <stdio.h>
int main(void)
{
     int c;
     printf("%d\n", c = 2);
     return 0;
}
```

- o ; é necessário para converter a expressão em comando

2

Memória

Sequência de células

- Memória $\approx$ um conjunto sequencial de célula
- Tamanho da célula = 1 **byte** (=8 bits)
- Cada célula possuí um endereço para acessar os dados dela

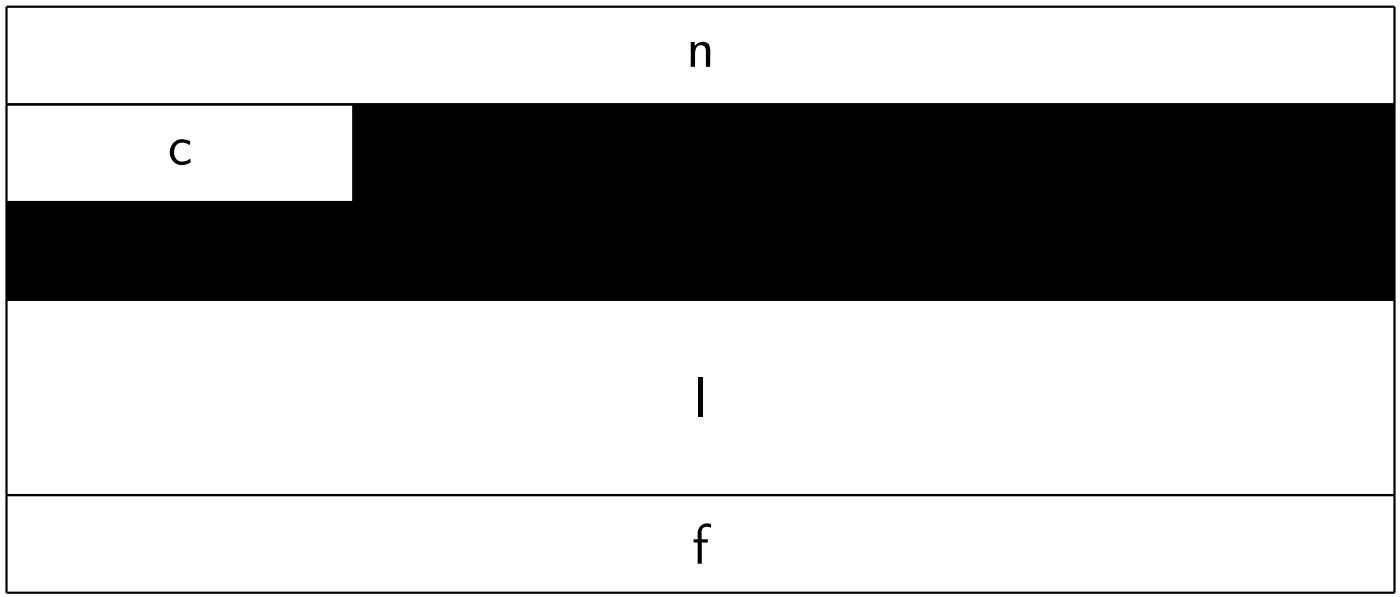
Variáveis

- Na memória, uma variável pode ocupar uma ou mais células.
- Por exemplo, um elemento de 64 bits (i.e. do tipo `double` ou `long long`) ocupa um espaço de 8 bytes.

Exemplo alinhamento

```
1   #include <stdio.h>
2   int main(void)
3   {
4         int n = 0;
5         char c = 245;
6         long long l = 2561;
7         float f = 3.14;
8
9
10        printf("&n = %p\n", &n);
11        printf("&c = %p\n", &c);
12        printf("&l = %p\n", &l);
13        printf("&f = %p\n", &f);
14        return 0;
15  }
```

```
&n = 0x7fff2dd30d4c
&c = 0x7fff2dd30d4b
&l = 0x7fff2dd30d40
&f = 0x7fff2dd30d3c
```



Operadores de endereço

Sintaxe

C tem dois operadores relacionados ao uso de endereços: `&` e `*`.

`&` / **referenciamento** permite acessar o endereço na memória de uma variável.

`*` / **dereferenciamento** permite acessar o valor armazenado dentro de um endereço.

Identidade

Para variáveis inicializadas corretamente: `*&x == x`

Inteiros

Tipos

Tipo	Tipo sem sinal	# bits	#bytes
char	unsigned char	8	1
short	unsigned short	16	2
int	unsigned char	32	4
long	unsigned long	32	4
long long	unsigned long long	64	8

- Os tamanhos em bits/bytes são indicativos.
- O padrão não manda exatamente os tamanhos.
- A representação do sinal usa **um bit**

Exemplo

```c
#include <stdio.h>

int main(void)
{
     int n_int = 1;
     long n_long = 0xdeadbeefl; // 1L
     unsigned int n_uint = 031u; // 1U
     unsigned long n_luint = 0x0ful;
     printf("ni = %x\nnl = %ld\nnu = %u\nnlu = %lu\n",
    n_int, n_long, n_uint, n_luint);
     return 0;
}
```

Resultados

ni	=	1
nl	=	3735928559
nu	=	25
nlu	=	15

Pontos flutuantes

Tipo	# bits	#bytes
float	32	4
double	64	8
long double	128	16

- Os tamanhos são fixados pelo padrão IEEE 754

Operadores aritméticos

Operador	Significado	Tipos
-	subtração, menos unário	todos
+	adição, + unário	todos
/	divisão	todos
*	multiplicação	todos
%	resto	todos salvo double ou float

Exemplo

```c
#include <stdio.h>

int main (void) {
  int a = 55;
  int b = 3;
  printf("+: %i\n-: %i\n/: %i\n*: %i\n%: %i\n",
 a + b, a - b, a / (float) b, a * b, a % b);
  return(0);
}
```

```
+:         58
-:         52
/:        165
*:          1
%:  620619280
```

Operadores de incrementação / decrementação

Primeira abordagem

- `n++` $\equiv$ `n = n + 1`
- `n--` $\equiv$ `n = n - 1`

Semântica

```c
#include <stdio.h>

int main(void)
{
     int x, y, n;

     n = 5;
     x = n++;
     y = ++n;
     printf("%d, %d, %d\n", n, x, y);
     return 0;
}
```

Regras de conversão implícitas

Básicas

- Se um operando for `long double`, converta o outro para `long double`
- Senão, se um for `double`, converta o outro para `double`
- Senão, se um for `float`, converta o outro para `float`
- Senão, converta, `char` e `short` para `int`
- Depois se um operando for `long`, converta o outro para `long`

Avançadas

- As regras de promoção dos elementos sem sinais são mais complexos.
- Um elemento `unsigned t` é convertido para o menor que pode conter todos os valores dele.

Atribuição

Quando `x = y`, `y` é convertido para o tipo de `x`.

Conversão explícita

A construção (<tipo>) <nome> permite converter explicitamente um valor para um outro.

```c
#include <stdio.h>

int main(void)
{
     float f = 3.1045e2;
     int i = 3;

     printf("%d\n%e\n",
    3 + (int) f, (float) i + f);
     return 0;
}
```

313
3.134500 (+02)

Caractere

Notação

- ‘a’ denota uma constante do tipo caractere
- A tabela ASCII (`man ascii`) mostra equivalência entre caractere e valor inteiro do tipo `char`
  - ‘0’ $\neq$ 0
  - ‘0’ = 48
- Caracteres especiais:

  nova linha ‘`\n`’
  tabulação ‘`\t`’
  backslash ‘`\\`’
- Notações numéricas
  - valor em octal ‘`\0ooo`’
  - valor hexadecimal ‘`\xddd`’

Observação

- C não tem um tipo dedicado para caractere.
- Tipo Um caractere = `char`

Strings

Definição

- Strings são sequências de caracteres entre ""
- Constantes desse tipo podem ser concatenadas no tempo da compilação
- O tipo é `char <nome>[]` ou `char * <nome>`
- Use a palavra-chave `const` para indicar uma constante.

Exemplo

```
#include <stdio.h>

int main(void)
{
     const char hw[] = "hello" "world";
     char* msg = "escreva \"hello\tworld\", por \'favor\'";
     const char* msg2 = "0 ou \x30 : mesma coisa\012";
     printf("%s = hello world ?\n", hw);
     printf("%s\n", msg);
     printf("%s", msg2);
     return 0;
}
```

Booleanos

- Cabeçalho `stdbool.h`
- 2 valores
  1. `true`
  2. `false`

Relação com inteiros

- Antes da introdução de `stdbool.h`: uso de inteiros
- Equivalência
  - `false` ≈ 0
  - `true` $\approx$ qualquer valor $\neq 0$
- Os nomes representam as constantes **inteiras** 0 e 1
- Não é considerado como um tipo diferente do que o tipo inteiro

Operadores relacionais

Operadores

Símbolo	Sentido
==	igualdade
!=	diferente
>	maior
<	menor
>=	maior ou igual
<=	menor ou igual

- O valor retornado por esses operadores é:
  - 0/false se a comparação for falsa
  - 1/true se a comparação for verdadeira

Exemplo

```c
#include <stdio.h>
int main(void)
{
  int n, m;
  int values_read;
  printf("Entre com dois inteiros\n");

  values_read = scanf("%d %d", &n, &m);
  if (values_read != 2) return -1; // problema de leitura
  printf("n > m: %d", n > m);
  printf("n < m: %d", n < m);
  printf("n == m: %d", n == m);
}
```

Operadores lógicos

Símbolo	Significado
!	não
&&	e
||	ou
^	xou

Exemplo

```c
int main(void)
{
     int n, m, o;
     int values_read;

     printf("Entre com 3 inteiros\n");
     values_read = scanf("%d %d %d", &n, &m, &o);
     if (values_read != 3) return -1;
     if (n > 0 && m > 0 && o > 0) printf("As 3 variáveis são positivas!\n");
     if (n > 0 && m > 0 && !(o >= 0))
  printf("A variável o não é positiva!\n");
     if (n > 0 || m > 0 || o > 0)
  printf("Uma variável (ou mais) é positiva\n");
     return 0;
}
```

Precedências (resumo)

- Em ordem descrescente de precedências

Operadores
++, –, !, +, - (unários)
*, /, %
+, - (binários)
<, >, <=, >=
==, !=
&&
||

getchar / putchar

Resumo

- ler/escrever **1** caractere da entrada/na saída padrão
- cabeçalho `stdio.h`
- declaração

```
1  int getchar(void);
2  int putchar(int c);
```

- `man getchar`
- `man putchar`

Exemplo

```
1   #include <stdio.h>
2   int main(void)
3   {
4        int c;
5
6        while ((c = getchar()) != EOF) {
7     putchar(tolower(c));
8        }
9        return 0;
10  }
```

printf

```c
#include <stdio.h>

int printf(const char *format, ...);
```

- man 3 printf

Especificadores de conversão

Tipo	Formato
int	d, i
unsigned int	u
char	c
float	e, f, g
double	e, f, g
strings	s

Comando de leitura `scanf`

```c
#include <stdio.h>

int scanf(const char *format, ...);
```

- `scanf` é o companheiro de entrada do `printf`.
- Ela usa o mesmo padrão de formato para *ler* informações de entrada.
- `man 3 scanf`

Exemplo

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
int main(int argc, char *argv[])
{
  float a, b, c;
  int values_read;

  if (scanf("%f %f %f", &a, &b, &c) != 3) {
       assert(argc >= 4);
       a = atof(argv[1]);
       b = atof(argv[2]);
       c = atof(argv[3]);
  }

  printf("Volume: %-5.3f\n", a * b * c);
  return 0;
}
```

```
Volume: 26.866
```

Comentários na linguagem C

As partes de *comentários* podem ser indicados de dois jeitos:

1. Para comentar uma linha só, pode usar `//`
2. Para comentários multilinhas, se usa a construção `/* <comentário> */`

Estilo: no caso 2, é frequente começar cada linha com um `*` alinhada com a `*` do inicio do comentários `/*`

Referências

Precisões

[KR88] Seções 2.1 – 2.8, 2.12

[Bac13] 2.1 – 2.3, 2.5, 3.1 – 3.4

André Backes, *Linguagem C completa e descomplicada*, Elsevier, 2013.

Brian W. Kernighan and Dennis M. Ritchie, *The (ANSI) C Programming Language*, 2nd ed., Prentice Hall Professional Technical Reference, 1988.

Perguntas ?

http://dimap.ufrn.br/~richard/dim0321