Programação Estruturada Aula 3 - Introdução

Prof. Luis Carlos Retondaro

Técnico em Telecomunicações
2^o Ano

CEFET/RJ - Centro Federal de Educação Tecnológica Celso Suckow da Fonseca

Campus Petrópolis

2017

Sumário

Constantes

- Constantes referem-se a valores fixos que o programa não pode alterar e podem ser de qualquer um dos tipos de dados básicos;
- **Exemplos:**
  - 'a', 'A';
  - 8, 10, 352;
  - "Linguagem C".

Constantes

- A maneira como cada constante é representada depende do seu tipo de dados:
  - **Constantes de caracter:** São envolvidas por aspas simples (’). Exemplo: ’a’ e ’%’;
  - **Constantes inteiras:** São especificados como números sem componentes fracionários. Exemplo: 10 e -100;
  - **Constantes de ponto flutuante:** Requerem o ponto decimal seguido pela parte fracionária do número. Exemplo: 11.123;
  - **Constante do tipo sting:** É um conjunto de caracteres entre aspas duplas. Exemplo: ”Hello world!”.

Constantes

- Os modificadores de tipo também podem ser utilizados, na declaração de constantes, para criar variações nos tipos básicos;
- Por padrão o C encaixa uma constante numérica no menor tipo de dado compatível que pode contê-lo;
- **Exemplo:** 10 é um int por padrão, mas 60.000 é um unsigned int e 100.000 é um long;
- Muito embora o valor 10 possa caber em um tipo caracter o compilador não atravessa os limites de tipo;
- O programador pode especificar precisamente o tipo da constante numérica que deseja através da utilização de um sufixo.

Constantes

Para tipos de ponto flutuante, usando o sufixo F a constante será tratada como um float, e usando L será um long double. Para tipos inteiros, o sufixo U representa unsigned e o L um long.

Tipo de dado	**Exemplo de constante**
int	1, 123, 21000, -234
long int	35000L, -34L
short int	10, -12, 90
unsigned int	10000U, 987U,
float	123.23F, 4.34F
double	123.23, 123123.33
long double	1001.2L

Constantes

Existem 2 maneiras mais comuns de criar uma constante em C:

1. **#define:** Forma padrão para criação de constantes com define:

```
#define identificador valor

//Exemplo
#define PI 3.14159265
#define ERRO "Erro!!!"
```

2. **const:** Também é possível criar constantes usando a palavra reservada const:

```
const tipo_da_variavel nome_da_variavel = valor;

//Exemplo
const int TAMANHO = 100;
```

Constantes

Exemplo de utilização de constantes

```c
// definindo constantes para o cálculo de uma circunferência

#include <stdio.h>

#define PI 3.14159
#define NEWLINE "\n"

int main (){
  float raio, circ;

  printf("Entre com o valor do raio \n");
  scanf("%f", &raio);    // raio
  circ = 2*PI*raio;        // utilizando a const. Pi.
  printf("circunferência = %f\n", circ);
  printf("%s", NEWLINE );
  return(0);
}
```

Constantes

Resultado da execução

```
Entre com o valor do raio
5.2
circunferência = 32.672535
```

Constantes Hexadecimais e Octais

- Para alguns problemas é mais fácil utilizar o sistema numérico na base 8 ou na base 16 em lugar do nosso sistema decimal padrão;
- O sistema numérico na base 8 é chamado de octal e utiliza números de 0 a 7;
- O sistema numérico na base 16 é o hexadecimal e utiliza números de 0 a 9 mais as letras de A a F que representam os números de 10 ao 15, respectivamente;
- Em virtude desses números serem frequentemente utilizados, o C permite especificar constantes inteiras em hexadecimal ou octal.

Constantes Hexadecimais e Octais

- Uma constante hexadecimal deve consistir em um 0x seguido por uma constante na forma hexadecimal;
- Uma constante octal começa com zero seguido de um número na forma octal;
- Exemplos:

```
int hex = 0x80;   // 128 em decimal
int oct = 012;    // 10 em decimal
```

Constantes Hexadecimais e Octais

- **Exemplo:**

```
#include <stdio.h>
void main(){
   int ex = 0x80;
   int oc = 012;
   printf("Exa: %d \t Octal: %d\n", ex, oc);
}
```

- **Resultado:**

```
Exa: 128        Octal: 10
```

Constantes strings

- O C suporta outro tipo de constante: a string;
- A string é um conjunto de caracteres colocado entre aspas duplas ("). Exemplo: "Isso é uma string";
- Embora o C permita que o programador defina constantes string, o C não possui formalmente um tipo de dado string;
- Não se deve confundir strings com caracteres. Uma constante de um único caracter é colocado entre aspas simples ('a'), porém "a" é uma string contendo apenas uma letra.

Barra invertida

- Colocar entre aspas simples as constantes de tipo caracter funciona para a maioria dos caracteres imprimíveis;
- Alguns caracteres são impossíveis de inserir pelo teclado, por isso o C criou constantes especiais de caracter de barra invertida.

Códigos de barra invertida

Código	Signnificado
\b	BackSpace
\f	Quebra de página
\n	Nova linha
\r	Retorno de carro
\t	Tabulação horizontal

Barra invertida

Códigos de barra invertida

Código	Signnificado
\"	Aspas duplas
\'	Aspas simples
\0	Nulo
\\	Barra invertida
\v	Tabulação vertical
\a	Emite sinal sonoro
\N	Constante octal (onde N é a constante)
\xN	Constante hexadecimal (onde N é a constante)

Funções de entrada e saída em C

- As funções de saída são utilizadas para interface com o usuário, ou seja, é utilizada para a comunicação entre a máquina e o usuário de alguma maneira, não importando que seja em uma folha impressa, que seja na tela, que seja colorido, que seja um desenho, etc;

- O programador deve se preocupar com toda comunicação entre a máquina e o usuário, de forma a induzir o usuário a praticar determinada ação ou mostrar claramente o resultado de um processamento.

Funções de entrada e saída em C

- A entrada de dados é algo muito importante para qualquer programa, que deve ser usado em conjunto com uma função de saída;
- A função de saída deve induzir o usuário a fazer uma ação, e a função de entrada é que vai armazenar esta ação do usuário;
- As funções de E/S em C são encontradas na biblioteca padrão de entrada e saída, stdio.h.

A função printf()

- A função printf() tem a seguinte forma geral:

```
printf("string_de_controle", lista_de_argumentos);
```

1. A string de controle é uma descrição de tudo que a função vai imprimir na tela;
2. Para a impressão dos argumentos é preciso especificar códigos de controle (especificadores de formato).

Exemplo - função printf()

```c
//exemplo de utilização da função printf()

#include <stdio.h> // biblioteca padrão I/O de C

int main (){
  printf ("Esta e uma mensagem sendo mostrada na tela");
  return 0;
}
```

A função printf()

Especificadores de formato

ESPECIFICADOR	VALOR
%d	inteiro
%o	inteiro em formato octal
%x	inteiro em formato hexadecimal
%X	
%u	unsigned int
%ld	long int
%f	float
%c	char
%e	float em formato exponencial
%E	
%g	float. C escolhe melhor maneira de exibição entre normal e exponencial
%G	
%s	string
%p	endereço de um ponteiro
%n	quantos caracteres a função printf exibiu

Constantes

Exemplo de impressão de constantes

```c
#include <stdio.h>

int main(){
   printf("O programa A imprime o número 2\n");
   printf("O programa %c imprime o número 2\n",'A');
   printf("O programa A imprime o número %d\n",2);
   printf("O programa %c imprime o número %d\n",'A',2);
   printf("O programa %c %s %d\n",'A',"imprime o n umero",2);
   return(0);
}
```

Constantes

Resultado da execução

```
O programa A imprime o número 2
O programa A imprime o número 2
O programa A imprime o número 2
O programa A imprime o número 2
O programa A imprime o número 2
```

A função printf()

Formatando valores inteiros

SINTAXE	EFEITO
printf (" %5d ",valor);	exibe valor com um mínimo de 5 caracteres
printf(" %05d ",valor);	exibe valor com um mínimo de 5 caracteres precedendo-o com zeros
#%o	exibe um valor octal precedido de 0 (zero)
#%x	exibe um valor hexadecimal precedido de 0x
#%X	

Formatando valores float

```
printf("O valor é: %5.3f ", valor);
```

Exibe valor com um mínimo de 5 caracteres e com 3 casas decimais.

A função printf()

Exemplo

```c
//Programa que formata inteiros na impressao
#include <stdio.h>

int main()
  {
    int valor = 1;
    float num = 3.1589;

    printf("\nJustificado a direita  => %5d\n",valor);
    printf("Justificado a esquerda => %-5d\n",valor);
    printf("Num com 2 casas decimais: %.2f\n",num);
    printf("Num com 6 casas decimais: %.6f\n",num);
    printf("Num com 10 caracteres e 3 casas decimais: %10.3f\n",num);
    return(0);
  }
```

A função printf()

Resultado de execução

```
Justificado a direita  =>     1
Justificado a esquerda => 1
Num com 2 casas decimais: 3.16
Num com 6 casas decimais: 3.158900
Num com 10 caracteres e 3 casas decimais:      3.159
```

A função scanf()

- A função scanf() é utilizada para realizar a entrada de dados para o programa através do teclado;
- O formato geral da função scanf() é:

```
scanf(string_de_controle, lista_de_argumentos);
```

- O scanf() deve ter como primeiro argumento uma string, denominado string de formato, que descreve como deve ser a seqüência de caracteres da entrada.

A função scanf()

Exemplo de uso da função scanf()

```c
 // Exemplo da função scanf()
#include <stdio.h>

int main(){

        int num;

        printf("Digite um número: ");

        scanf("%d", &num);
        printf("Numero digitado: %d\n", num);
        return(0);
}
```

A função scanf()

Exemplo de uso da função scanf() com várias entradas na mesma linha

```
 // Exemplo da função scanf() com múltiplas entradas na mesma linha
#include <stdio.h>
int main (void){
   int a, b, r;
   printf ("Digite dois numeros para serem somados:\n");
   scanf ("%d %d", &a, &b);
   r=a+b;
   printf ("\nA soma de %d mais %d e igual a %d", a, b, r);
   return 0;

}
```

As funções gets() e getchar()

- As funções gets() e getchar(), assim com a scanf() lêem da entrada padrão, porém elas não suportam formatação;
- A função gets(), vem de get string, e lê uma string até o final da linha ou até que não se tenha mais dados para ler. Ao final adiciona o terminador de string "\0";
- A função getchar() lê apenas um caracter;
- Não se usa o & quando ao ler uma string dado que ela é um vetor de caracteres e o seu nome é o endereço de memória do primeiro elemento do vetor.

```
//Exemplos

char nome[20], c;
gets(nome);
c = getchar();
```

As funções gets() e getchar()

- A função scanf() lê tudo até aparecer um espaço em branco e pára antes dele, já a função gets() lê tudo até aparecer um new line, inclusive um nada.

```
#include <stdio.h>

int main()
{
  char buffer[10], sexo;
  int idade;

  printf("Entre com o seu nome: \n");
  gets(buffer);
  printf("Entre com o sexo (F ou M): \n");
  sexo = getchar();
  printf("Entre com a idade:\n");
  scanf("%d",&idade);
  printf("\n**** Dados ****\n");
  printf("Nome: %s\n", buffer);
  printf("Sexo: %c\n",sexo);
  printf("Idade: %d\n",idade);
  printf("****************\n\n");
  return 0;
}
```

As funções gets() e getchar()

Resultado de execução

```
Entre com o seu nome:
Laura Assis
Entre com o sexo (F ou M):
F
Entre com a idade:
31

**** Dados ****
Nome: Laura Assis
Sexo: F
Idade: 31
****************
```

As funções puts() e putchar()

- São as funções mais simples do cabeçalho stdio.h. Ambas enviam (ou "imprimem") à saída padrão os caracteres fornecidos a elas;
- A função puts(), vem de "put string" e é utilizada para colocar uma string na saída de dados;
- A função putchar() significa "put char" e é utilizada para colocar um caracter na saída de dados.

```
//Exemplos

 char nome[20];
 gets(nome);
 puts(nome);
 puts(" Aula de C");
 putchar('A');
 putchar('\n');
```

As funções puts() e putchar()

- Note que junto com a função puts() devemos usar literais de string (com aspas duplas), e com putchar() devemos usar literais de caractere (com aspas simples);

- Se o programador tentar compilar algo como putchar("T"), o compilador enviaria uma mensagem de erro, pois "T" é diferente de 'T'.

As funções puts() e putchar()

- Pode-se usar caracteres especiais como tabulação e quebra de linha;

```
//Exemplos

 puts ("Primeira linha \n Segunda linha \t e um grande espaço (tab)");
 putchar ('\n'); // apenas envia uma quebra de linha
 puts("Fim");
```

- A função puts() sempre coloca uma quebra de linha após imprimir a string, o que não ocorre com o putschar() e printf().

Include

- A instrução *#include < stdio.h >* que precede o *main*() é necessária pois:
  - Para definir uma função é necessário dizer o tipo de retorno, identificador da função e lista de parâmetros (protótipo da função);
  - Para que a função *main*() ative uma outra função, definida pelo usuário ou contida na biblioteca, seu protótipo deve ser definido antes de sua chamada;
  - Os protótipos de funções do sistema estão reunidos (de acordo com objetivos semelhantes) em arquivos cabeçalhos (*header files*).

Include

- A instrução *#include < stdio.h >* que precede o *main*() é necessária pois:
  - A instrução *#include < stdio.h >*, anexa ao *main*() os protótipos das funções de biblioteca que executam ações padrões de E/S;
  - stdio vem de *standard input output*, entrada e saída padrão e *h* é a extensão padrão dos arquivos de cabeçalhos.

Referências

1. C Completo e Total, Herbert Schidt; Pearson Makron Books; 3a. Ed., 1997.

2. Linguagem C. DAMAS, Luis. 10a. Edição. LTC, 2014.