Técnicas de Programação

TP0801

Prof. Giovane Barcelos
giovane_barcelos@uniritter.edu.br

Plano de Ensino
Conteúdo programático

1. Introdução à programação em C 2. Desenvolvimento estruturado de programas em C 3. Controle de programa 4. Funções 5. Arrays 6. Ponteiros	**N1**
7. Caracteres e strings 8. Entrada/Saída formatada 9. Estruturas, uniões, manipulações de bits e enumerações 10. Processamento de arquivos 11. Estruturas de dados 12. O pré-processador 13. Outros tópicos sobre C	**N2**

Objetivos

- A usar as funções da biblioteca de tratamento de caracteres (<ctype.h>).
- A usar as funções de conversão de strings da biblioteca de utilitários gerais (<stdlib.h>).
- A usar as funções de entrada/saída de string e caracteres da biblioteca-padrão de entrada/saída (<stdio.h>).
- A usar as funções de processamento de string da biblioteca de tratamento de strings (<string.h>).
- O poder das bibliotecas de funções que levam à reutilização de software.

Objetivos

- A usar streams de entrada e saída.
- A usar todas as capacidades de formatação da impressão.
- A usar todas as capacidades de formatação da entrada.
- A imprimir com larguras de campo e precisões.
- A usar flags de formatação na string de controle de formato printf.
- A enviar literais e sequências de escape.
- A formatar a entrada usando scanf.

Introdução

- Uma parte importante da solução de qualquer problema é a apresentação dos resultados.
- Neste capítulo, discutiremos em detalhes os recursos de formatação de scanf e printf.
- Essas funções recebem dados do stream de entrada-padrão e enviam dados para o stream de saída-padrão.
- Quatro outras funções que usam entrada-padrão e saída-padrão — gets, puts, getchar e putchar — foram discutidas no Capítulo.
- Inclua o cabeçalho <stdio.h> nos programas que chamam essas funções.

Streams

- Todas as entradas e saídas são realizadas a partir de streams (ou fluxos), que são sequências de bytes.
- Nas operações de entrada, os bytes fluem de um dispositivo (por exemplo, um teclado, uma unidade de disco, uma conexão de rede) para a memória principal.
- Nas operações de saída, os bytes fluem da memória principal para um dispositivo (por exemplo, uma tela de vídeo, uma impressora, uma unidade de disco, uma conexão de rede e assim por diante).
- Quando a execução do programa começa, três streams são conectados ao programa automaticamente.

Streams

- Em geral, o stream de entrada-padrão é conectado ao teclado, e o stream de saída-padrão é conectado à tela.
- Os sistemas operacionais normalmente permitem que esses streams sejam redirecionados para outros dispositivos.
- Um terceiro stream, o stream de erro-padrão, é conectado à tela.
- As mensagens de erro são enviadas para o stream de erro-padrão.

Formatação da saída com printf

- A formatação precisa da saída é obtida a partir de printf.
- Todas as chamadas a printf contêm uma string de controle de formato, que descreve o formato de saída.
- A string de controle de formato consiste em especificadores de conversão, flags, larguras de campo, precisões e caracteres literais.
- Com o sinal de porcentagem (%), eles formam as especificações de conversão.

Formatação da saída com printf

- **A função printf pode realizar as capacidades de formatação a seguir (elas serão discutidas neste capítulo):**
  - **Arredondamento de valores de ponto flutuante para um número indicado de casas decimais.**
  - **Alinhamento de uma coluna de números com pontos decimais que aparecem uns sobre os outros.**
  - **Alinhamento à direita e alinhamento à esquerda das saídas.**
  - **Inserção de caracteres literais em locais exatos em uma linha de saída.**
  - **Representação de números em ponto flutuante em formato exponencial.**
  - **Representação de inteiros não sinalizados em formato octal e hexadecimal. Saiba mais sobre valores octais e hexadecimais no Apêndice C.**
  - **Exibição de todos os tipos de dados com larguras de campo de tamanho fixo e precisões.**

Formatação da saída com printf

- **A função printf tem a forma**
  - **Printf (*string-de-controle-de-formato, outros-argumentos*);**

***string-de-controle-de-forma*to descreve o formato da saída, e outros-argumentos (que são opcionais) correspondem a cada uma das especificações de conversão na *string-de-controle-de-formato*.**

- **Cada especificação de conversão começa com um sinal de porcentagem e termina com um especificador de conversão.**
- **Pode haver muitas especificações de conversão em uma string de controle de formato.**

Formatação da saída com printf

Erro comum de programação 9.1
Esquecer de usar aspas para deliminar uma string-de-controle-de-formato consiste em um erro de sintaxe.

Boa prática de programação 9.1
Formate as saídas de maneira nítida para a apresentação, tornando as saídas do programa mais legíveis para, assim, reduzir o número de erros cometidos pelos usuários.

Impressão de inteiros

Especificador de conversão	Descrição
d	Exibido como um inteiro decimal com sinal.
i	Exibido como um inteiro decimal com sinal. [*Nota:* os especificadores i e d são diferentes quando usados juntamente com scanf.]
o	Exibido como um inteiro octal sem sinal.
u	Exibido como um inteiro decimal sem sinal.
x ou X	Exibido como um inteiro hexadecimal sem sinal. X faz com que os dígitos de 0 a 9 e as letras de A a F sejam exibidas, e x faz com que os dígitos de 0 a 9 e de a a f sejam exibidos.
h ou l (letra l)	Colocados antes de qualquer especificador de conversão de inteiro para indicar que um inteiro short ou long será exibido, respectivamente. As letras h e l são chamadas de **modificadores de tamanho**.

Figura 9.1 ■ Especificadores de conversão de inteiros.

Impressão de inteiros

- Um inteiro é um número que não possui casas decimais, como 776, 0 ou –52.
- Os valores inteiros são exibidos em um de vários formatos.
- A Figura 9.1 descreve os especificadores de conversão de inteiros.

Impressão de inteiros

```
 1  /* Fig 9.2: fig09_02.c */
 2  /* Usando especificadores de conversão de inteiros */
 3  #include <stdio.h>
 4
 5  int main( void )
 6  {
 7     printf( "%d\n", 455 );
 8     printf( "%i\n", 455 ); /* i é o mesmo que d em printf */
 9     printf( "%d\n", +455 );
10     printf( "%d\n", -455 );
11     printf( "%hd\n", 32000 );
```

Figura 9.2 ■ Uso de especificadores de conversão de inteiros. (Parte 1 de 2.)

Impressão de inteiros

```
12      printf( "%ld\n", 2000000000L ); /* sufixo L torna a literal um long */
13      printf( "%o\n", 455 );
14      printf( "%u\n", 455 );
15      printf( "%u\n", -455 );
16      printf( "%x\n", 455 );
17      printf( "%X\n", 455 );
18      return 0; /* indica conclusão bem-sucedida */
19   } /* fim do main */
```

```
455
455
455
-455
32000
2000000000
707
455
4294966841
1c7
1C7
```

Figura 9.2 ■ Uso de especificadores de conversão de inteiros. (Parte 2 de 2.)

Impressão de inteiros

```
1  /* Fig 9.2: fig09_02.c */
2  /* Usando especificadores de conversão de inteiros */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     printf( "%d\n", 455 );
8     printf( "%i\n", 455 ); /* i é o mesmo que d em printf */
9     printf( "%d\n", +455 );
10    printf( "%d\n", -455 );
11    printf( "%hd\n", 32000 );
```

Figura 9.2 ■ Uso de especificadores de conversão de inteiros. (Parte 1 de 2.)

Impressão de inteiros

```
12      printf( "%ld\n", 2000000000L ); /* sufixo L torna a literal um long */
13      printf( "%o\n", 455 );
14      printf( "%u\n", 455 );
15      printf( "%u\n", -455 );
16      printf( "%x\n", 455 );
17      printf( "%X\n", 455 );
18      return 0; /* indica conclusão bem-sucedida */
19   } /* fim do main */
```

```
455
455
455
-455
32000
2000000000
707
455
4294966841
1c7
1C7
```

Figura 9.2 ■ Uso de especificadores de conversão de inteiros. (Parte 2 de 2.)

Impressão de inteiros

- A Figura 9.2 imprime um inteiro usando cada um dos especificadores de conversão de inteiros.
- Apenas o sinal de subtração é impresso; sinais de adição são suprimidos.
- Além disso, o valor -455, quando lido por %u (linha 15), é convertido para o valor sem sinal 4294966841.

Impressão de inteiros

Erro comum de programação 9.2
Imprimir um valor negativo usando um especificador de conversão que espera por um valor unsigned.

Impressão de números em ponto flutuante

Especificador de conversão	Descrição
e ou E	Exibe um valor de ponto flutuante em notação exponencial.
F	Exibe valores de ponto flutuante em notação de ponto fixo. [*Nota:* em C99, você também pode usar F.]
g ou G	Exibe um valor de ponto flutuante no formato de ponto flutuante `f` ou no formato exponencial `e` (ou E), com base na magnitude do valor.
L	Usado antes de qualquer especificador de conversão de ponto flutuante para indicar que um valor de ponto flutuante `long double` será exibido.

Figura 9.3 ■ Especificadores de conversão de ponto flutuante.

Impressão de números em ponto flutuante

- Um valor de ponto flutuante contém um ponto decimal, como em 33.5, 0.0 ou -657.983.
- Os valores de ponto flutuante são exibidos em um de vários formatos.
- A Figura 9.3 descreve os especificadores de conversão de ponto flutuante.
- Os especificadores de conversão e e E em valores de ponto flutuante em notação exponencial — em computação, o equivalente à notação científica usada na matemática.

Impressão de números em ponto flutuante

- Por exemplo, em notação científica, o valor 150,4582 é representado como
  - $1.504582 \acute{} 10^2$
- e é representado em notação exponencial como
  - 1.504582E+02
- pelo computador.
- Essa notação indica que 1.504582 é multiplicado por 10 elevado à segunda potência (E+02).
- O E significa 'expoente'.

Impressão de números em ponto flutuante

- Os valores exibidos com os especificadores de conversão e, E e f mostram seis dígitos de precisão à direita do ponto decimal como padrão (por exemplo, 1.04592); outras precisões podem ser especificadas explicitamente.
- O especificador de conversão f sempre imprime pelo menos um dígito à esquerda do ponto decimal.
- Os especificadores de conversão e e E imprimem e minúsculo e E maiúsculo, respectivamente, precedendo o expoente, e imprimem exatamente um dígito à esquerda do ponto decimal.
- O especificador de conversão g (ou G) imprime tanto no formato e (E) quanto no formato f sem zeros no final (1.234000 é impresso como 1.234).

Impressão de números em ponto flutuante

- Os valores são impressos com e (E) se, após a conversão para a notação exponencial, o expoente do valor for menor que -4, ou o expoente for maior ou igual à precisão especificada (seis dígitos significativos como padrão para g e G).
- Caso contrário, o especificador de conversão f é usado para imprimir o valor.
- Zeros finais não são impressos na parte fracionária de uma saída de valor com g ou G.
- Pelo menos um dígito decimal é necessário para que o ponto decimal seja apresentado.

Impressão de números em ponto flutuante

- Com o especificador de conversão g, os valores 0.0000875, 8750000.0, 8.75, 87.50 e 875 são impressos como 8.75e-05, 8.75e+06, 8.75, 87.5 e 875.
- O valor 0.0000875 usa a notação e porque, ao ser convertido para a notação exponencial, seu expoente (-5) é menor que -4.
- O valor 8750000.0 usa a notação e porque seu expoente (6) é igual à precisão-padrão.

Impressão de números em ponto flutuante

- A precisão dos especificadores de conversão g e G indica o número máximo de dígitos significativos a serem impressos, incluindo o dígito à esquerda do ponto decimal.
- O valor 1234567.0 é impresso como 1.23457e+06, ao usarmos o especificador de conversão %g (lembre-se de que todos os especificadores de conversão de ponto flutuante possuem uma precisão padrão de 6).
- Existem 6 dígitos significativos no resultado.
- A diferença entre g e G idêntica à diferença entre e e E, quando o valor é impresso em notação exponencial — um g minúsculo faz com que um e minúsculo seja exibido, e um G maiúsculo faz com que um E maiúsculo seja exibido.

Impressão de números em ponto flutuante

Dica de prevenção de erro 9.1

Ao exibir dados, cuide para que o usuário esteja ciente de situações em que eles poderão ser imprecisos devido à
formatação (por exemplo, erros de arredondamento da especificação de precisões).

Impressão de números em ponto flutuante

```
1  /* Fig 9.4: fig09_04.c */
2  /* Imprimindo números em ponto flutuante com
3  especificadores de conversão de ponto flutuante */
4
5  #include <stdio.h>
6
7  int main( void )
8  {
9     printf( "%e\n", 1234567.89 );
10    printf( "%e\n", +1234567.89 );
11    printf( "%e\n", -1234567.89 );
12    printf( "%E\n", 1234567.89 );
13    printf( "%f\n", 1234567.89 );
14    printf( "%g\n", 1234567.89 );
15    printf( "%G\n", 1234567.89 );
16    return 0; /* indica conclusão bem-sucedida */
17 } /* fim do main */
```

```
1.234568e+006
1.234568e+006
-1.234568e+006
1.234568E+006
1234567.890000
1.23457e+006
1.23457E+006
```

Figura 9.4 ■ Uso de especificadores de conversão de ponto flutuante.

Impressão de números em ponto flutuante

- A Figura 9.4 demonstra cada um dos especificadores de conversão de ponto flutuante.
- Os especificadores de conversão %E, %e e %g fazem com que o valor seja arredondado na saída, mas o mesmo não ocorre com %f.
- Com alguns compiladores, o expoente nas saídas será mostrado com dois dígitos à direita do sinal +.

Impressão de strings e caracteres

```c
/* Fig 9.5: fig09_05c */
/* Imprimindo strings e caracteres */
#include <stdio.h>

int main( void )
{
   char character = 'A'; /* initialize char */
   char string[] = "Isso é uma string"; /* inicializa array de char */
   const char *stringPtr = "Isso também é uma string"; /* ponteiro de char */

   printf( "%c\n", character );
   printf( "%s\n", "Isso é uma string" );
   printf( "%s\n", string );
   printf( "%s\n", stringPtr );
   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
```

```
A
Isso é uma string
Isso é uma string
Isso também é uma string
```

Figura 9.5 ■ Uso de especificadores de conversão de caractere e string.

Impressão de strings e caracteres

- Os especificadores de conversão c e s são usados para imprimir caracteres individuais e strings, respectivamente.
- O especificador de conversão c requer um argumento char.
- O especificador de conversão s requer um ponteiro para char como argumento.
- O especificador de conversão s faz com que os caracteres sejam impressos até que um caractere nulo de finalização ('\0') seja encontrado.
- O programa mostrado na Figura 9.5 apresenta caracteres e strings com especificadores de conversão c e s.

Impressão de strings e caracteres

Erro comum de programação 9.3
Usar %c para imprimir uma string consiste em um erro. O especificador de conversão %c espera por um argumento char. Uma string é um ponteiro para char (ou seja, char *).

Erro comum de programação 9.4
Normalmente, usar %s para imprimir um argumento char causa um erro fatal, chamado violação de acesso, no tempo de execução. O especificador de conversão %s espera por um argumento do tipo ponteiro para char.

Impressão de strings e caracteres

Erro comum de programação 9.5

Usar aspas simples em torno de strings de caracteres consiste em um erro de sintaxe. Strings de caracteres devem ser delimitados por aspas duplas.

Erro comum de programação 9.6

Usar aspas duplas em torno de uma constante de caractere cria um ponteiro para uma string que consiste em dois
caracteres, sendo o segundo um caractere nulo de finalização.

Outros especificadores de conversão

Especificador de conversão	Descrição
p	Exibe um valor de ponteiro de uma maneira definida pela implementação.
n	Armazena o número de caracteres já enviados na instrução `printf` atual. Um ponteiro para um inteiro é fornecido como argumento correspondente. Nada é exibido.
%	Exibe o caractere de porcentagem.

Figura 9.6 ■ Outros operadores de conversão.

Outros especificadores de conversão

- The three remaining conversion specifiers are p, n and % (Fig. 9.6).
- The conversion specifier %n stores the number of characters output so far in the current printf—the corresponding argument is a pointer to an integer variable in which the value is stored—nothing is printed by a %n.
- The conversion specifier % causes a percent sign to be output.

Outros especificadores de conversão

```
1  /* Fig 9.7: fig09_07.c */
2  /* Usando os especificadores de conversão p, n e % */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     int *ptr; /* declara ponteiro para int */
8     int x = 12345; /* inicializa int x */
9     int y; /* declara int y */
10
11    ptr = &x; /* atribui endereço de x a ptr */
12    printf( "O valor de ptr é %p\n", ptr );
13    printf( "O endereço de x é %p\n\n", &x );
14
15    printf( "Total de caracteres impressos nessa linha:%n", &y );
16    printf( " %d\n\n", y );
17
18    y = printf( "Essa linha tem 29 caracteres\n" );
19    printf( "%d caracteres foram impressos\n\n", y );
20
21    printf( "Imprimindo um %% em uma string de controle de formato\n" );
22    return 0; /* indica conclusão bem-sucedida */
23 } /* fim do main */
```

Outros especificadores de conversão

```
O valor de ptr é 0012FF78
O endereço de x é 0012FF78

Total de caracteres impressos nessa linha: 38

Essa linha tem 29 caracteres
29 caracteres foram impressos

Imprimindo um % em uma string de controle de formato
```

Figura 9.7 ■ Uso de especificadores de conversão p, n e %.

Outros especificadores de conversão

- O %p da Figura 9.7 imprime o valor de ptr o endereço de x; esses valores são idênticos porque ptr recebe o endereço de x.
- Em seguida, %n armazena o número de caracteres enviados pela terceira instrução printf (linha 15) na variável inteira y, o valor de y é impresso.
- A última instrução printf (linha 21) usa %% para imprimir o caractere % em uma string de caracteres.

Outros especificadores de conversão

- **Todas as chamadas a printf retornam um valor — seja o número de caracteres enviados ou um valor negativo, se houve um erro na saída.**
- **[*Nota*: esse exemplo não funcionará no Microsoft Visual C++, pois %n foi desativado pela Microsoft 'por motivos de segurança'. Para executar o restante do programa, remova as linhas 15 e 16.]**

Outros especificadores de conversão

Dica de portabilidade 9.1

O especificador de conversão p exibe um endereço de uma maneira definida pela implementação (em muitos sistemas, a notação hexadecimal é usada no lugar da notação decimal).

Erro comum de programação 9.7

Tentar imprimir um caractere de porcentagem literal usando % em vez de %% na string de controle de formato.

Quando % aparece em uma string de controle de formato, ele precisa ser seguido por um especificador de conversão.

Impressão com larguras de campo e precisão

```c
/* Fig 9.8: fig09_08.c */
/* Imprimindo inteiros alinhados à direita */
#include <stdio.h>

int main( void )
{
   printf( "%4d\n", 1 );
   printf( "%4d\n", 12 );
   printf( "%4d\n", 123 );
   printf( "%4d\n", 1234 );
   printf( "%4d\n\n", 12345 );

   printf( "%4d\n", -1 );
   printf( "%4d\n", -12 );
   printf( "%4d\n", -123 );
   printf( "%4d\n", -1234 );
   printf( "%4d\n", -12345 );
   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
```

```
   1
  12
 123
1234
12345

  -1
 -12
-123
-1234
-12345
```

Figura 9.8 ■ Alinhamento de inteiros à direita em um campo.

Impressão com larguras de campo e precisão

- O tamanho exato de um campo em que os dados são impressos é especificado por uma **largura de campo**.
- Se a largura do campo for maior que os dados a serem impressos, eles normalmente serão alinhados à direita dentro desse campo.
- Um inteiro que representa a largura do campo é inserido entre o sinal de porcentagem (%) e o especificador de conversão (por exemplo, %4d).
- A Figura 9.8 imprime dois grupos de cinco números cada um, alinhando à direita os números que contêm menos dígitos do que a largura do campo.
- A largura do campo é aumentada para que se possa imprimir valores maiores que o campo, e o sinal de subtração para um valor negativo usa uma posição de caractere na largura do campo.
- As larguras de campo podem ser usadas com todos os especificadores de conversão.

Impressão com larguras de campo e precisão

Erro comum de programação 9.8
Não oferecer um campo com largura suficiente para acomodar um valor a ser impresso pode deslocar outros dados
a serem impressos, e isso pode produzir saídas confusas. Conheça seus dados!

Impressão com larguras de campo e precisão

- A função printf também permite que você especifique a precisão com que os dados são impressos.
- A precisão tem diferentes significados para diferentes tipos de dados.
- Quando usada com especificadores de conversão de inteiros, a precisão indica o número mínimo de dígitos a serem impressos.
- Se o valor impresso tiver menos dígitos que a precisão especificada, e o valor da precisão tiver um zero inicial ou ponto decimal, zeros serão prefixados ao valor impresso até que o número total de dígitos seja equivalente à precisão.
- Se não houver um zero ou um ponto decimal presente no valor da precisão, espaços serão inseridos em seu lugar.

Impressão com larguras de campo e precisão

- A precisão-padrão para inteiros é 1.
- Quando usada com os especificadores de conversão de ponto flutuante e, E e f, a precisão será o número de dígitos que aparece após o ponto decimal.
- Quando usada juntamente com os especificadores de conversão g e G, a precisão será o número máximo de dígitos significativos a serem impressos.
- Quando usada com o especificador de conversão s, a precisão será o número máximo de caracteres a serem escritos a partir da string.
- Para usar a precisão, coloque um ponto decimal (.), seguido por um inteiro representando a precisão entre o sinal de porcentagem e o especificador de conversão.

Impressão com larguras de campo e precisão

```
1  /* Fig 9.9: fig09_09.c */
2  /* Usando precisão ao imprimir números inteiros,
3     números em ponto flutuante e strings */
4  #include <stdio.h>
5
6  int main( void )
7  {
8     int i = 873; /* inicializa int i */
9     double f = 123.94536; /* inicializa double f */
10    char s[] = "Feliz Aniversário"; /* inicializa array s de char */
11
12    printf( "Usando precisão para inteiros\n" );
13    printf( "\t%.4d\n\t%.9d\n\n", i, i );
14
15    printf( "Usando precisão para números em ponto flutuante\n" );
16    printf( "\t%.3f\n\t%.3e\n\t%.3g\n\n", f, f, f );
17
18    printf( "Usando precisão para strings\n" );
19    printf( "\t%.11s\n", s );
20    return 0; /* indica conclusão bem-sucedida */
21 } /* fim do main */
```

```
Usando precisão para inteiros
         0873
         000000873

Usando precisão para números em ponto flutuante
         123.945
         1.239e+002
         124

Usando precisão para strings
         Feliz Anive
```

Figura 9.9 ■ Uso de precisões na exibição de informações de vários tipos.

Impressão com larguras de campo e precisão

- A Figura 9.9 demonstra o uso da precisão nas strings de controle de formato.
- Quando um valor de ponto flutuante é impresso com uma precisão menor que o número original de casas decimais no valor, o valor é arredondado.

Impressão com larguras de campo e precisão

- A largura do campo e a precisão podem ser combinadas ao se colocar a largura do campo seguida por um ponto decimal, este seguido por uma precisão entre o sinal de porcentagem e o especificador de conversão, como na instrução
  - printf("%9.3f", 123.456789);

  que apresenta 123.457 com três dígitos à direita do ponto decimal, alinhado à direita em um campo de nove dígitos.
- É possível especificar a largura do campo e a precisão usando expressões inteiras da lista de argumentos que seguem a string de controle de formato.

Impressão com larguras de campo e precisão

- Para usar esse recurso, insira um asterisco (*) no lugar da largura do campo ou precisão (ou ambos).
- O argumento int correspondente na lista de argumentos é avaliado e usado no lugar do asterisco.
- O valor da largura de um campo pode ser positivo ou negativo (o que faz com que a saída seja alinhada à esquerda no campo conforme descrito na próxima seção).
- A instrução
  - printf("%*.*f", 7, 2, 98.736);

  usa 7 para a largura do campo, 2 para a precisão e envia o valor 98.74 alinhado à direita.

Uso de flags na string de controle de formato de printf

Flag	Descrição
– (sinal de subtração)	Alinha a saída à esquerda dentro do campo especificado.
+ (sinal de adição)	Exibe um sinal de adição antes dos valores positivos e um sinal de subtração antes de valores negativos.
Espaço	Imprime um espaço antes de um valor positivo não impresso com a flag +.
#	Prefixa 0 ao valor de saída quando usado com o especificador de conversão octal o. Prefixa 0x ou 0X ao valor de saída quando usado com os especificadores de conversão hexadecimais x ou X. Força um ponto decimal para um número em ponto flutuante impresso com e, E, f, g ou G que não contém uma parte fracionária. (Normalmente, o ponto decimal só será impresso se houver um dígito depois dele.) Para os especificadores g e G, os zeros iniciais não são eliminados.
0 (zero)	Preenche um campo com zeros iniciais.

Figura 9.10 ■ Flags de string de controle de formato.

Uso de flags na string de controle de formato de printf

- A função printf também oferece flags para suplementar suas capacidades de formatação de saída.
- Cinco flags estão disponíveis para uso nas strings de controle de formato (Figura 9.10).
- Para usar uma flag em uma string de controle de formato, coloque-a imediatamente à direita do sinal de porcentagem.
- Várias flags podem ser combinadas em um especificador de conversão.

Uso de flags na string de controle de formato de printf

```
 1  /* Fig 9.11: fig09_11.c */
 2  /* Alinhando valores à direita e à esquerda */
 3  #include <stdio.h>
 4
 5  int main( void )
 6  {
 7     printf( "%10s%10d%10c%10f\n\n", "olá", 7, 'a', 1.23 );
 8     printf( "%-10s%-10d%-10c%-10f\n", "olá", 7, 'a', 1.23 );
 9     return 0; /* indica conclusão bem-sucedida */
10  } /* fim do main */
```

```
      olá         7         a      1.230000

olá       7         a         1.230000
```

Figura 9.11 ■ Alinhamento de strings à esquerda em um campo.

Uso de flags na string de controle de formato de printf

- **A Figura 9.11 mostra os alinhamentos à direita e à esquerda de uma string, de um inteiro, de um caractere e de um número em ponto flutuante.**

Uso de flags na string de controle de formato de printf

```
 1  /* Fig 9.12: fig09_12.c */
 2  /* Imprimindo números com e sem a flag + */
 3  #include <stdio.h>
 4
 5  int main( void )
 6  {
 7     printf( "%d\n%d\n", 786, -786 );
 8     printf( "%+d\n%+d\n", 786, -786 );
 9     return 0; /* indica conclusão bem-sucedida */
10  } /* fim do main */
```

```
786
-786
+786
-786
```

Figura 9.12 ■ Impressão de números positivos e negativos, com e sem a flag +.

Uso de flags na string de controle de formato de printf

- A Figura 9.12 imprime um número positivo e um número negativo, ambos com e sem a flag +.
- O sinal de subtração é exibido em ambos os casos, mas o sinal de adição é exibido apenas quando a flag + é usada.

Uso de flags na string de controle de formato de printf

```
 1  /* Fig 9.13: fig09_13.c */
 2  /* Imprimindo um espaço antes dos valores com sinal
 3     não precedidos por + ou - */
 4  #include <stdio.h>
 5
 6  int main( void )
 7  {
 8     printf( "% d\n% d\n", 547, -547 );
 9     return 0; /* indica conclusão bem-sucedida */
10  } /* fim do main */
```

```
 547
-547
```

Figura 9.13 ■ Uso da flag de espaço.

Uso de flags na string de controle de formato de printf

- A Figura 9.13 insere um espaço no início do número positivo com a flag de espaço.
- Isso é útil para alinhar números positivos e negativos com o mesmo número de dígitos.
- O valor -547 não é precedido por um espaço na saída devido ao seu sinal de subtração.

Uso de flags na string de controle de formato de printf

```
1  /* Fig 9.14: fig09_14.c */
2  /* Usando a flag # com especificadores de conversão
3     o, x, X e qualquer especificador de ponto flutuante */
4  #include <stdio.h>
5
6  int main( void )
7  {
8     int c = 1427; /* inicializa c */
9     double p = 1427.0; /* inicializa p */
10
```

Figura 9.14 ■ Uso da flag #. (Parte 1 de 2.)

Uso de flags na string de controle de formato de printf

```
11      printf( "%#o\n", c );
12      printf( "%#x\n", c );
13      printf( "%#X\n", c );
14      printf( "\n%g\n", p );
15      printf( "%#g\n", p );
16      return 0; /* indica conclusão bem-sucedida */
17   } /* fim do main */
```

```
02623
0x593
0X593

1427
1427.00
```

Figura 9.14 ■ Uso da flag #. (Parte 2 de 2.)

Uso de flags na string de controle de formato de printf

- A Figura 9.14 usa a flag # para prefixar 0 ao valor octal, e 0x e 0X aos valores hexadecimais, forçando o ponto decimal em um valor impresso com g.

Uso de flags na string de controle de formato de printf

```
1  /* Fig 9.15: fig09_15.c */
2  /* Imprimir com a flag 0( zero ) preenche com zeros iniciais */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     printf( "%+09d\n", 452 );
8     printf( "%09d\n", 452 );
9     return 0; /* indica conclusão bem-sucedida */
10 } /* fim do main */
```

```
+00000452
000000452
```

Figura 9.15 ■ Uso da flag 0 (zero).

Uso de flags na string de controle de formato de printf

- A Figura 9.15 combina a flag + e a flag 0 (zero) para imprimir 452 um campo de 9 espaços com um sinal de + e zeros iniciais, e depois imprime 452 novamente, usando apenas a flag 0 e um campo de 9 espaços.

Impressão de literais e de sequências de escape

Sequência de escape	Descrição
\' (aspas simples)	Exibe o caractere de aspas simples (‘).
\" (aspas duplas)	Exibe o caractere de aspas duplas (“).
\? (ponto de interrogação)	Exibe o caractere de ponto de interrogação (?).
\\ (barra invertida)	Exibe o caractere de barra invertida (\).
\a (alerta ou campainha)	Causa alerta audível (campainha) ou visual.
\b (backspace)	Move o cursor uma posição para trás na linha atual.
\f (nova página ou form feed)	Move o cursor para o início da página seguinte.
\n (newline)	Move o cursor para o início da linha seguinte.
\r (carriage return)	Move o cursor para o início da linha atual.
\t (tabulação horizontal)	Move o cursor para a posição de tabulação horizontal seguinte.
\v (tabulação vertical)	Move o cursor para a posição de tabulação vertical seguinte.

Figura 9.16 ■ Sequências de escape.

Impressão de literais e de sequências de escape

- A maior parte dos caracteres literais a serem impressos em uma instrução printf pode simplesmente ser incluída na string de controle de formato.
- Porém, existem vários caracteres ‘problemáticos’, como as aspas duplas (") , que delimitam a própria string de controle de formato.
- Diversos caracteres de controle, como newline e tabulação, precisam ser representados por sequências de escape.
- Uma sequência de escape é representada por uma barra invertida (\) seguida por um caractere de escape em particular.
- A Figura 9.16 lista as sequências de escape e as ações que elas provocam.

Impressão de literais e de sequências de escape

Erro comum de programação 9.9
Tentar imprimir aspas simples, aspas duplas ou o caractere de barra invertida como dados literais em uma instrução
printf sem iniciá-los com uma barra invertida para formar uma sequência de escape apropriada consiste
em um erro.

Leitura da entrada formatada com scanf

- A formatação precisa de entrada pode ser obtida a partir de scanf.
- Cada instrução scanf contém uma string de controle de formato que descreve o formato dos dados a serem informados.
- A string de controle de formato consiste em especificadores de conversão e em caracteres literais.
- A função scanf tem as seguintes capacidades de formatação de entrada:
  - Entrada de todos os tipos de dados.
  - Entrada de caracteres específicos de um stream de entrada.
  - Salto de caracteres específicos no stream de entrada.

Leitura da entrada formatada com scanf

- **A função scanf é escrita no seguinte formato:**
  - **scanf (*string-de-controle-de-formato, outros-argumentos*);**

***string-de-controle-de-formato* descreve os formatos da entrada, e *outros-argumentos* são ponteiros para as variáveis em que a entrada será armazenada.**

Leitura da entrada formatada com scanf

Boa prática de programação 9.2
Ao inserir dados, peça ao usuário um item de dado ou alguns poucos itens de cada vez. Evite pedir ao usuário que digite muitos itens de dados em resposta a um único pedido.

Boa prática de programação 9.3
Sempre considere o que o usuário e o seu programa farão quando (e não se) dados incorretos forem inseridos — por exemplo, um valor para um inteiro que não faz sentido no contexto de um programa, ou uma string em que faltam pontuação ou espaços.

Leitura da entrada formatada com scanf

Especificador de conversão	Descrição
Inteiros	
`d`	Leem um inteiro decimal com sinal opcional. O argumento correspondente é um ponteiro para um `int`.
`i`	Lê um inteiro decimal, octal ou hexadecimal com sinal opcional. O argumento correspondente é um ponteiro para um `int`.
`o`	Lê um inteiro octal. O argumento correspondente é um ponteiro para um `int` sem sinal.
`u`	Lê um inteiro decimal sem sinal. O argumento correspondente é um ponteiro para um `int` sem sinal.
`x` ou `X`	Lê um inteiro hexadecimal sem sinal. O argumento correspondente é um ponteiro para um `int` sem sinal.
`h` ou `l`	Colocados antes de qualquer um dos especificadores de conversão de inteiros para indicar que um inteiro `short` ou um inteiro `long` deve ser inserido.
Números em ponto flutuante	
`e`, `E`, `f`, `g` ou `G`	Leem um valor de ponto flutuante. O argumento correspondente é um ponteiro para uma variável de ponto flutuante.
`l` ou `L`	Colocados antes de qualquer um dos especificadores de conversão de ponto flutuante para indicar que um valor `double` ou um `long double` deve ser inserido. O argumento correspondente é um ponteiro para uma variável `double` ou uma variável `long double`.
Caracteres e strings	
`c`	Lê um caractere. O argumento correspondente é um ponteiro para um `char`; nenhum nulo (`'\0'`) é acrescentado.
`s`	Lê uma string. O argumento correspondente é um ponteiro para um array do tipo `char` que é grande o suficiente para manter a string e um caractere nulo (`'\0'`) de término — que é automaticamente acrescentado.

Leitura da entrada formatada com scanf

Conjunto de varredura [*caracteres de varredura*]	
	Varre uma string em busca de um conjunto de caracteres que estão armazenados em um array.
Diversos	
p	Lê um endereço de mesmo formato produzido quando um endereço é enviado com %p em uma instrução `printf`.
n	Armazena o número de caracteres inseridos até esse ponto da chamada a `scanf`. O argumento correspondente é um ponteiro para um `int`.
%	Salta o sinal de porcentagem (%) na entrada.

Figura 9.17 ■ Especificadores de conversão para scanf.

Leitura da entrada formatada com scanf

- A Figura 9.17 resume os especificadores de conversão usados na inserção de todos os tipos de dados.
- O restante desta seção oferecerá programas que demonstram a leitura de dados utilizando os diversos especificadores de conversão de scanf.

Leitura da entrada formatada com scanf

```
1  /* Fig 9.18: fig09_18.c */
2  /* Lendo inteiros */
3  #include <stdio.h>
4
5  int main( void )
6  {
```

Figura 9.18 ■ Leitura da entrada com especificadores de conversão de inteiros. (Parte 1 de 2.)

Leitura da entrada formatada com scanf

```
 7      int a;
 8      int b;
 9      int c;
10      int d;
11      int e;
12      int f;
13      int g;
14
15      printf( "Digite sete inteiros: " );
16      scanf( "%d%i%i%i%o%u%x", &a, &b, &c, &d, &e, &f, &g );
17
18      printf( "A entrada exibida como inteiros decimais é:\n" );
19      printf( "%d %d %d %d %d %d %d\n", a, b, c, d, e, f, g );
20      return 0; /* indica conclusão bem-sucedida */
21  } /* fim do main */
```

```
Digite sete inteiros: -70 -70 070 0x70 70 70 70
A entrada exibida como inteiros decimais é:
-70 -70 56 112 56 70 112
```

Figura 9.18 ■ Leitura da entrada com especificadores de conversão de inteiros. (Parte 2 de 2.)

Leitura da entrada formatada com scanf

- A Figura 9.18 lê inteiros com os diversos especificadores de conversão de inteiros, e exibe esses inteiros como números decimais.
- O especificador de conversão %i é capaz de inserir inteiros decimais, octais e hexadecimais.

Leitura da entrada formatada com scanf

```
1  /* Fig 9.19: fig09_19.c */
2  /* Lendo números em ponto flutuante */
3  #include <stdio.h>
4
5  /* função main inicia a execução do programa */
6  int main( void )
7  {
8     double a;
9     double b;
10    double c;
11
12    printf( "Digite três números em ponto flutuante: \n" );
13    scanf( "%le%lf%lg", &a, &b, &c );
14
15    printf( "Estes são os números digitados em\n" );
16    printf( "notação de ponto flutuante simples:\n" );
17    printf( "%f\n%f\n%f\n", a, b, c );
18    return 0; /* indica conclusão bem-sucedida */
19 } /* fim do main */
```

```
Digite três números em ponto flutuante:
1.27987 1.27987e+03 3.38476e-06
Estes são os números digitados em
notação de ponto flutuante simples:
1.279870
1279.870000
0.000003
```

Figura 9.19 ■ Leitura da entrada com especificadores de conversão de ponto flutuante.

Leitura da entrada formatada com scanf

- Ao inserir números em ponto flutuante, qualquer um dos especificadores de conversão de ponto flutuante e, E, f, g ou G podem ser usados.
- A Figura 9.19 lê três números em ponto flutuante, um com cada um dos três tipos de especificadores de conversão flutuantes, e apresenta os três números com o especificador de conversão f.
- A saída do programa confirma o fato de que os valores de ponto flutuante são imprecisos — isso é destacado pelo terceiro valor impresso.

Leitura da entrada formatada com scanf

```c
/* Fig 9.20: fig09_20.c */
/* Lendo caracteres e strings */
#include <stdio.h>

int main( void )
{
   char x;
   char y[ 9 ];

   printf( "Digite uma string: " );
   scanf( "%c%s", &x, y );

   printf( "A entrada foi:\n" );
   printf( "o caractere \"%c\" ", x );
   printf( "e a string \"%s\"\n", y );
   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
```

```
Digite uma string: Domingo
A entrada foi:
o caractere "D" e a string "omingo"
```

Figura 9.20 ■ Entrada com caracteres e strings.

Leitura da entrada formatada com scanf

- Caracteres e strings são lidos utilizando-se os especificadores de conversão c e s, respectivamente.
- A Figura 9.20 pede ao usuário que digite uma string.
- O programa insere o primeiro caractere da string com %c e o armazena na variável de caractere x, e depois insere o restante da string com %s e o armazena no array de caracteres y.

Leitura da entrada formatada com scanf

- Uma sequência de caracteres pode ser lida a partir de um conjunto de varredura.
- Um conjunto de varredura é um conjunto de caracteres delimitados por colchetes, [], e precedidos por um sinal de porcentagem na string de controle de formato.
- Um conjunto de varredura procura os caracteres no stream de entrada, buscando apenas por caracteres que combinam com os caracteres contidos no conjunto de varredura.
- Toda vez que um caractere é combinado, ele é armazenado no argumento correspondente do conjunto de varredura — um ponteiro para um array de caracteres.
- O conjunto de varredura termina a inserção de caracteres quando um caractere que não está contido no conjunto de varredura é encontrado.

Leitura da entrada formatada com scanf

```
1  /* Fig 9.21: fig09_21.c */
2  /* Usando um conjunto de varredura */
3  #include <stdio.h>
4
5  /* função main inicia a execução do programa */
6  int main( void )
7  {
8     char z[ 9 ]; /* declara array z */
9
10    printf( "Digite a string: " );
11    scanf( "%[aeiou]", z ); /* procura por um conjunto de caracteres */
12
13    printf( "A entrada foi \"%s\"\n", z );
14    return 0; /* indica conclusão bem-sucedida */
15 } /* fim do main */
```

```
Digite a string: ooeeooahah
A entrada foi "ooeeooa"
```

Figura 9.21 ■ Uso de um conjunto de varredura.

Leitura da entrada formatada com scanf

- **Se o primeiro caractere no *stream* de entrada não combinar com um caractere no conjunto de varredura, somente o caractere nulo será armazenado no array.**
- **A Figura 9.21 usa o conjunto de varredura [aeiou] para varrer o stream de entrada em busca de vogais.**
- **Observe que as sete primeiras letras da entrada são lidas.**
- **A oitava letra (h) não está no conjunto de varredura e, portanto, a varredura é terminada.**

Leitura da entrada formatada com scanf

```
1  /* Fig 9.22: fig09_22.c */
2  /* Usando um conjunto de varredura invertido */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     char z[ 9 ];
8
9     printf( "Digite uma string: " );
10    scanf( "%[^aeiou]", z ); /* conjunto de varredura invertido */
11
12    printf( "A entrada foi \"%s\"\n", z );
13    return 0; /* indica conclusão bem-sucedida */
14 } /* fim do main */
```

```
Digite uma string: String
A entrada foi "Str"
```

Figura 9.22 ■ Uso de um conjunto de varredura invertido.

Leitura da entrada formatada com scanf

- O conjunto de varredura também pode ser usado para varrer os caracteres não contidos no conjunto de varredura; para isso é necessário usar um conjunto de varredura invertido.
- Para criar um conjunto de varredura invertido, coloque um circunflexo (^) dentro dos colchetes, antes dos caracteres de varredura.
- Isso faz com que os caracteres que não aparecem no conjunto de varredura sejam armazenados.
- Quando um caractere contido no conjunto de varredura invertido é encontrado, a entrada termina.
- A Figura 9.22 usa o conjunto de varredura invertido[^aeiou] para procurar por consoantes — ou melhor, para procurar por 'não vogais'.

Leitura da entrada formatada com scanf

```c
/* Fig 9.23: fig09_23.c */
/* Inserindo dados com uma largura de campo */
#include <stdio.h>

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

   printf( "Digite um inteiro com seis dígitos: " );
   scanf( "%2d%d", &x, &y );

   printf( "Os inteiros digitados foram %d e %d\n", x, y );
   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
```

```
Digite um inteiro com seis dígitos: 123456
Os inteiros digitados foram 12 e 3456
```

Figura 9.23 ■ Inserção de dados com uma largura de campo.

Leitura da entrada formatada com scanf

- **Uma largura de campo pode ser usada em um especificador de conversão scanf na leitura de um número específico de caracteres do stream de entrada.**
- **A Figura 9.23 entra com uma série de dígitos consecutivos como um inteiro de dois dígitos, e um inteiro que consiste nos dígitos restantes do stream de entrada.**

Leitura da entrada formatada com scanf

- Normalmente, é necessário pular certos caracteres no stream de entrada.
- Por exemplo, uma data poderia ser informada como
  - 11-10-1999
- Cada número da data precisa ser armazenado, mas os traços que separam os números podem ser descartados.
- Para eliminar caracteres desnecessários, inclua-os na string de controle de formato de scanf (caracteres de espaço em branco — por exemplo, espaço, newline e tabulação — pulam todo o espaço em branco inicial).

Leitura da entrada formatada com scanf

- Por exemplo, para pular os traços na entrada, use a instrução
  - scanf("%d-%d-%d", &mes, &dia, &ano);
- Embora esse scanf elimine os traços da data que apresentamos, é possível que a data seja informada como
  - 10/11/1999
- Nesse caso, o scanf que mostramos não eliminaria os caracteres desnecessários.
- Por esse motivo, scanf oferece o caractere de supressão de atribuição *.

Leitura da entrada formatada com scanf

```c
/* Fig 9.24: fig09_24.c */
/* Lendo e descartando caracteres do stream de entrada */
#include <stdio.h>

int main( void )
{
   int month1;
   int day1;
   int year1;
   int month2;
   int day2;
   int year2;

   printf( "Digite uma data no formato mm-dd-aaaa: " );
   scanf( "%d%*c%d%*c%d", &month1, &day1, &year1 );

   printf( "mês = %d dia = %d ano = %d\n\n", month1, day1, year1 );

   printf( "Digite uma data no formato mm/dd/aaaa: " );
   scanf( "%d%*c%d%*c%d", &month2, &day2, &year2 );

   printf( "mês = %d  dia = %d  ano = %d\n", month2, day2, year2 );
   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
```

```
Digite uma data no formato mm-dd-aaaa: 11-18-2003
mês = 11  dia = 18  ano = 2003

Digite uma data no formato mm/dd/aaaa: 11/18/2003
mês = 11  dia = 18  ano = 2003
```

Figura 9.24 ■ Leitura e descarte de caracteres do stream de entrada.

Leitura da entrada formatada com scanf

- O caractere de supressão de atribuição permite que scanf leia qualquer tipo de dado da entrada e o descarte sem atribuí-lo a uma variável.
- A Figura 9.24 usa o caractere de supressão de atribuição no especificador de conversão %c para indicar que um caractere que aparece no stream de entrada deve ser lido e descartado.
- Somente o mês, dia e ano são armazenados.
- Os valores das variáveis são impressos para demonstrar que eles realmente serão inseridos corretamente.
- As listas de argumentos para cada chamada a scanf não contêm variáveis para os especificadores de conversão que usam o caractere de supressão de atribuição.
- Os caracteres correspondentes são simplesmente descartados.

Lembre-se

“ Corrija a causa, não o sintoma. ” Steve Maguire