Comandos de Entrada e Saída

1 SAÍDA DE DADOS

- Função printf

Função para impressão de dados formatados na saída padrão (*stdout* - tela) definida no arquivo cabeçalho *stdio.h*.

Sintaxe:

printf ("<texto para impressão>", <expressão1>, <expressão2>, ...);

onde:

<texto para impressão> é o texto que será impresso na tela. Este texto pode ser composto de caracteres que serão exibidos na tela e de caracteres especiais cuja função é indicar: a) os locais e tipos de expressões que serão mostradas ou b) ações que serão executadas como pular uma linha ou emitir um bip no alto-falante.

<expressão> este argumento é opcional e indica um valor que será impresso dentro do <texto para impressão>. Pode ser uma variável, constante, expressão aritmética ou lógica ou ainda uma chamada a outra função.

Exemplos:

```
printf ("Este texto será mostrado na tela.");
printf ("O valor da variável idade é %d.", idade);
printf ("Depois deste texto serão puladas duas linhas.\n\n");
```

Para imprimir o conteúdo de uma variável dentro do texto é necessário inserir neste mesmo texto no local de impressão um % seguido do tipo a ser mostrado. As opções disponíveis para impressão de variáveis são:

Código	Tipo da Impressão
%c	caracter
%o	número na base octadecimal
%x	número na base hexadecimal
%d	número na base decimal
%e	número em notação científica
%f	número float
%l	número long int
%lf	número double
%u	número sem sinal
%s	string
%p	endereço
%%	sinal de %

Estes código de impressão podem ser usados também para alterar o tamanho do campo e a forma de impressão do resultado:

Tamanho do campo:

Para especificar o tamanho mínimo deve-se colocar a quantidade de casas do campo entre o % e o símbolo do tipo correspondente.

Exemplos (supondo que as variáveis *num1* e *num2* tenham o valor 8):

1.
```
printf ("Valor: %6d!", num1);
```

Imprime uma variável inteira com no mínimo 6 casas, alinhadas a direita. Se a variável *num* possuir menos de 6 dígitos o número será completado com espaços em branco a esquerda.

Saída do comando:
Valor: 8!

2.
```
printf("Valor: %-6d!", num1);
```

Imprime uma variável inteira com no mínimo 6 casas, alinhadas a esquerda. Se a variável *num* possuir menos de 6 dígitos o número será

completado com espaços em branco a direita.

Saída do comando:
Valor: 8 !

3. `printf("Valor: %4.2f!", num2);`

Imprime uma variável *float* com no mínimo 4 dígitos sendo no máximo 2 decimais. Se não for especificado o número de casas decimais serão impressas 6 casas decimais.

Saída do comando:
Valor: 8.00!

Impressão de zeros a esquerda:

Na seção anterior foi mostrado que um número é preenchido com espaços em branco a esquerda se não tiver o tamanho mínimo especificado no comando **printf**. É possível mudar o caracter de preenchimento para o zero, bastando colocar o número zero entre o % e o número de casas indicado.

Exemplo:
```
printf ("Valor: %06d!", num1);
```

Este comando imprimirá o valor da variável *num1* alinhado a direita com zeros a esquerda conforme a saída indicada abaixo:

Valor: 000008!

Além de valores de variáveis, podemos especificar, dentro do texto para impressão, caracteres especiais que representam determinadas ações, conforme a tabela abaixo:

Código	**Ação**
\n	Pula uma linha
\t	Tabulação
\b	Backspace
\a	Beep no alto-falante
\"	Imprime "
\\	Imprime \

Estes caracteres especiais assim como os códigos de impressão podem ser usados em qualquer quantidade e em qualquer local dentro do <texto para impressão>.

Outros exemplos:
```
printf ("Teste %% %%")                                  -> "Teste % %"
printf ("%f",40.345)                                    -> "40.345"
printf ("Um caractere %c e um inteiro %d",'D',120)      -> "Um caractere D e um inteiro 120"
printf ("%s e um exemplo","Este")                       -> "Este e um exemplo"
printf ("%s%d%%","Juros de ",10)                        -> "Juros de 10%"
```

- Função putchar

Função para impressão de caracteres isolados na saída padrão definida no arquivo cabeçalho *stdio.h*.

Sintaxe:
```
putchar( <caracter ou código ASCII do caracter para impressão> );
```

Exemplos:
```
char letra = 'A';
putchar(letra);
putchar('A');
putchar(65);
```

Todos os exemplos acima apresentam na tela a letra 'A'. No último deles o código 65 representa a letra 'A' na tabela ASCII.

2 ENTRADA DE DADOS

- Função scanf

Função para leitura de dados da entrada padrão (*stdin* - teclado) definida no arquivo cabeçalho *stdio.h*.

Sintaxe:

scanf("<string de leitura>", &<variável 1>, &<variável 2>, ...);

onde:

<string para leitura> é composta pelos tipos das variáveis para leitura, por caracteres que devem ser digitados junto com as variáveis e com formatação especial (entre colchetes);

&<variável> endereço da variável que receberá o valor que for digitado no teclado. Os valores digitados devem ser separados por um espaço em branco, uma tabulação ou pelo *enter*.

Exemplos:

1. `scanf("%d", &numero);`

   Este comando aguarda a digitação de um valor numérico do teclado. Este valor digitado é colocado na variável *numero*.

2. `scanf("%c", &letra);`

   Este comando aguarda a digitação de um caracter do teclado. Este caracter digitado é colocado na variável *letra*.

3. `scanf("%f", &preco);`

   Este comando aguarda a digitação de um valor numérico do teclado. Este valor digitado é colocado na variável *preco*. As casas decimais devem ser digitadas após um ponto e não uma vírgula. Não é possível estabelecer o tamanho máximo de uma variável com o *scanf*.

4. `scanf("%d%d", &num1, &num2);`

   Este comando aguarda a digitação de dois valores numéricos do teclado. O primeiro valor digitado é colocado em *num1* e o segundo em *num2*. Estes valores podem ser separados por espaço em branco, tabulação ou *enter* e finalizados por um *enter*.

5. `scanf("%d,%d", &num1, &num2);`

   Este comando aguarda a digitação de dois valores numéricos do teclado. O primeiro valor digitado é colocado em *num1* e o segundo em *num2*. A diferença com o anterior é a vírgula colocada entre os dois *%d*. Na execução do programa esta vírgula deve ser digitada entre os dois números. Qualquer caracter colocado dentro da <string de leitura> que não seja um tipo (% mais a letra) e que não esteja entre colchetes deve ser digitado no momento da execução do programa.

Os códigos de impressão que podem ser utilizados no *scanf* são semelhantes aos do *printf* como indica a tabela abaixo:

Código	**Tipo da variável lida**
%c	caracter
%s	string
%d	inteiro na base decimal
%x	inteiro na base hexadecimal
%o	inteiro na base octadecimal
%u	inteiro sem sinal
%l	inteiro longo
%f	número float
%lf	número double

%e número em notação científica

- Função getchar

Função para leitura de caracteres isolados da entrada padrão (*stdin* - teclado) definida no arquivo cabeçalho *stdio.h*.

Sintaxe:

<varchar> = **getchar**();

onde:

<varchar> é a variável que receberá o caracter digitado e recuperado pelo **getchar**. O usuário deve teclar *enter* após o caracter.

- Funções getch e getche

Funções para leitura de caracteres isolados da entrada padrão (*stdin* - teclado) definidas no arquivo cabeçalho *conio.h*. A diferença destas funções para o **getchar** é que os caracteres são recuperados no momento da digitação, sem a necessidade de se pressionar a teclar *enter*. A diferença entre o **getch** e o **getche** é que esta mostra o caracter digitado enquanto aquela, não.

Sintaxe:

<varchar> = **getch**();
<varchar> = **getche**();

onde:

<varchar> é a variável que receberá o caracter digitado e recuperado por **getch** ou **getche**.

2 - OPERADORES

2.1 - ARITMÉTICOS E DE ATRIBUIÇÃO

Os operadores aritméticos são usados para desenvolver operações matemáticas. A seguir apresentamos a lista dos operadores aritméticos do C:

Operador	**Ação**
+	Soma (inteira e ponto flutuante)
-	Subtração ou Troca de sinal (inteira e ponto flutuante)
*	Multiplicação (inteira e ponto flutuante)
/	Divisão (inteira e ponto flutuante)
%	Resto de divisão (de inteiros)
++	Incremento (inteiro e ponto flutuante)
--	Decremento (inteiro e ponto flutuante)

O C possui operadores unários e binários. Os unários agem sobre uma variável apenas, modificando ou não o seu valor, e retornam o valor final da variável. Os binários usam duas variáveis e retornam um terceiro valor, sem alterar as variáveis originais. A soma é um operador binário pois pega duas variáveis, soma seus valores, sem alterar as variáveis, e retorna esta soma. Outros operadores binários são os operadores - (subtração), *****, / e **%**. O operador - como troca de sinal é um operador unário que não altera a variável sobre a qual é aplicado, pois ele retorna o valor da variável multiplicado por -1.

O operador / (divisão) quando aplicado a variáveis inteiras, nos fornece o resultado da divisão inteira; quando aplicado a variáveis em ponto flutuante nos fornece o resultado da divisão "real". O operador % fornece o resto da divisão de dois inteiros. Assim seja o seguinte trecho de código:

```
int a = 17, b = 3;
int x, y;
float z = 17. , z1, z2;
x = a / b;
y = a % b;
z1 = z / b;
z2 = a/b;
```

ao final da execução destas linhas, os valores calculados seriam x = 5, y = 2, z1 = 5.666666 e z2 = 5.0 . Note que, na linha correspondente a z2, primeiramente é feita uma divisão inteira (pois os dois operandos são inteiros). Somente após efetuada a divisão é que o resultado é atribuído a uma variável float.

Os operadores de incremento e decremento são unários que alteram a variável sobre a qual estão aplicados. O que eles fazem é incrementar ou decrementar, a variável sobre a qual estão aplicados, de 1. Então

```
x++;
x--;
```

são equivalentes a

```
x=x+1;
x=x-1;
```

Estes operadores podem ser pré-fixados ou pós- fixados. A diferença é que quando são pré-fixados eles incrementam e retornam o valor da variável já incrementada. Quando são pós-fixados eles retornam o valor da variável sem o incremento e depois incrementam a variável. Então, em

```
x=23;
y=x++;
```

teremos, no final, **y=23** e **x=24**. Em

```
x=23;
y=++x;
```

teremos, no final, **y=24** e **x=24**. Uma curiosidade: a linguagem de programação C++ tem este nome pois ela seria um "incremento" da linguagem C padrão. A linguagem C++ é igual a linguagem C só que com extensões que permitem a programação orientada a objeto, o que é um recurso extra.

O operador de atribuição do C é o =. O que ele faz é pegar o valor à direita e atribuir à variável da esquerda. Além disto ele retorna o valor que ele atribuiu. Isto faz com que as seguintes expressões sejam válidas:

```
x=y=z=1.5;              /* Expressao 1 */

if (k=w) ...            /* Expressao 2 */
```

A expressão 1 é válida, pois quando fazemos **z=1.5** ela retorna 1.5, que é passado adiante, fazendo y = 1.5 e posteriormente x = 1.5. A expressão 2 será verdadeira se **w** for diferente de zero, pois este será o valor retornado por **k=w**. Pense bem antes de usar a expressão dois, pois ela pode gerar erros de interpretação. Você *não* está comparando **k** e **w**. Você está atribuindo o valor de **w** a **k** e usando este valor para tomar a decisão.

2.2 - OPERADORES RELACIONAIS E LÓGICOS

Os operadores relacionais do C realizam ***comparações*** entre variáveis. São eles:

Operador	Ação
>	Maior do que
>=	Maior ou igual a
<	Menor do que
<=	Menor ou igual a
==	Igual a
!=	Diferente de

Os operadores relacionais retornam verdadeiro (1) ou falso (0). Para verificar o funcionamento dos operadores relacionais, execute o programa abaixo:

```
/* Este programa ilustra o funcionamento dos operadores relacionais. */
#include <stdio.h>
int main()
{
    int i, j;
    printf("\nEntre com dois numeros inteiros: ");
    scanf("%d%d", &i, &j);
    printf("\n%d == %d é %d\n", i, j, i==j);
    printf("\n%d != %d é %d\n", i, j, i!=j);
    printf("\n%d <= %d é %d\n", i, j, i<=j);
    printf("\n%d >= %d é %d\n", i, j, i>=j);
    printf("\n%d < %d é %d\n", i, j, i<j);
    printf("\n%d > %d é %d\n", i, j, i>j);
    return(0);
}
```

Você pode notar que o resultado dos operadores relacionais é sempre igual a 0 (falso) ou 1 (verdadeiro).
Para fazer ***operações com valores lógicos*** (verdadeiro e falso) temos ***os operadores lógicos***:

Operador	Ação
&&	AND (E)
||	OR (OU)
!	NOT (NÃO)

Usando os operadores relacionais e lógicos podemos realizar uma grande gama de testes. A tabela-verdade destes operadores é dada a seguir:

p	q	p AND q	p OR q
falso	falso	falso	falso
falso	verdadeiro	falso	verdadeiro
verdadeiro	falso	falso	verdadeiro
verdadeiro	verdadeiro	verdadeiro	verdadeiro

O programa a seguir ilustra o funcionamento dos operadores lógicos. Compile-o e faça testes com vários valores para i e j:

```
#include <stdio.h>
int main()
{
    int i, j;
    printf("informe dois números(cada um sendo 0 ou 1): ");
    scanf("%d%d", &i, &j);
    printf("%d AND %d é %d\n", i, j, i && j);
    printf("%d OR %d é %d\n", i, j, i || j);
    printf("NOT %d é %d\n", i, !i);
}
```

Exemplo: No trecho de programa abaixo a operação j++ será executada, pois o resultado da expressão lógica é verdadeiro:

```
int i = 5, j =7;
if ( (i > 3) && ( j <= 7) && ( i != j) ) j++;
```

- V and V and V = V

Mais um exemplo. O programa abaixo, imprime na tela somente os números pares entre 1 e 100, apesar da variação de i ocorrer de 1 em 1:

```
/* Imprime os números pares entre 1 e 100. */
#include <stdio.h>
int main()
{
    int i;
    for (i=1; i<=100; i++)
        if (!(i%2))
            printf ("%d ",i); /* o operador de resto dará falso (zero) */
}  /* quando usada c/ número par. Esse resultado é invertido pelo ! */
```

2.3 - OPERADORES LÓGICOS BIT A BIT

O C permite que se faça ***operações lógicas "bit-a- bit"*** em números. Ou seja, neste caso, o número é representado por sua forma binária e as operações são feitas em cada bit dele. Imagine um número inteiro de 16 bits, a variável i, armazenando o valor 2. A representação binária de i, será: 0000000000000010 (quinze zeros e um único 1 na segunda posição da direita para a esquerda). Poderemos fazer operações em cada um dos bits deste número. Por exemplo, se fizermos a negação do número (operação binária NOT, ou operador binário ~ em C), isto é, ~i, o número se transformará em 1111111111111101. As operações binárias ajudam programadores que queiram trabalhar com o computador em "baixo nível". As operações lógicas bit a bit só podem ser usadas nos tipos **char**, **int** e **long int**. Os operadores são:

Operador	Ação
&	AND
|	OR
^	XOR (OR exclusivo)
~	NOT
>>	Deslocamento de bits a direita
<<	Deslocamento de bits a esquerda

Os operadores **&**, |, ^ e **~** são as operações lógicas bit a bit. A forma geral dos operadores de deslocamento é:

valor>>número_de_deslocamentos
valor<<número_de_deslocamentos

O número_de_deslocamentos indica o quanto cada bit irá ser deslocado. Por exemplo, para a variável i anterior, armazenando o número 2:

i << 3;

fará com que i agora tenha a representação binária: 0000000000010000, isto é, o valor armazenado em i passa a ser igual a 16.

2.4 - OPERADORES CONJUGADOS ARITMÉTICOS

São conhecidos também como operadores de atribuição ou operadores de atribuição compostos. Substituem expressões aritméticas onde a variável que recebe o resultado também faz parte da expressão, como nos exemplos:

cont = cont + 5;
*x = x * (y-1) ;*

Estes exemplos podem ser convertidos para:

cont += 5;
*x *= y-1;*

usando o operador conjugado +=.

Os operadores conjugados aritméticos são: +=, -=, *=, /=, %=. Existem também os operadores conjugados para operações com bits que serão vistos em outro capítulo. Atenção! Ao contrário do que afirmam certos livros, estes operadores NÃO geram um código de máquina mais eficiente! Eles servem apenas para tornar o código mais compacto.

2.5 - OPERADORES DE MOLDE - (TYPE CAST)

Em certas expressões com tipos de dados misturados é possível realizar a conversão automática de tipos. Por exemplo, quando uma expressão contém variáveis *int* e *float* o tipo do resultado será *float*. Ou seja, as únicas conversões automáticas possíveis são aquelas que convertem um operador mais "estreito" para um mais "largo". Entretanto existem situações onde não é possível a conversão automática ou o próprio programador precisa controlar o tipo de uma expressão. Nestes casos deve ser usado um *type cast*:

Sintaxe:

(tipo) expressão;

Onde *tipo* pode ser qualquer tipo da linguagem C usado em declarações e *expressão* é qualquer expressão válida.

Exemplo:

int valor;
valor = (int) 3.141592;

Converte o valor de *pi* para um inteiro temporariamente, ou seja, após a execução do comando, a variável *valor* terá como conteúdo o número 3. Sem *type cast* não seria possível realizar esta atribuição de maneira direta. Para garantir um funcionamento preciso do programa e compatibilidade entre plataformas sempre deve-se usar *type cast*.

Exercícios

1. Usando apenas um comando **printf** fazer um programa que imprima os valores de variáveis representando a população de um país, o peso de uma pessoa e o símbolo do oxigênio.
2. Fazer um programa que imprima, com apenas um comando **printf**, as frases abaixo exatamente como estão. Todos os valores numéricos devem estar armazenados dentro de variáveis.

   a) "Vendemos 50% a mais que no ano passado."

   b) Em 31/12/2000 nosso capital era de R$ 50.000.000,00.

   c) A barra utilizada para pastas seria '/' ou '\'?

3. Fazer um programa que leia um caracter do teclado com a função **scanf** e mostre o seu código ASCII correspondente. Refaça o programa usando para a leitura do teclado as funções **getch**, **getche** e **getchar**.

4. Diga o resultado das variáveis x, y e z depois da seguinte seqüência de operações:

```
int x,y,z;
x=y=10;
z=++x;
x=-x;
y++;
x=x+y-(z--);
```

5. Fazer um programa que leia dois valores do tipo *float* do teclado e mostre a sua soma, divisão, multiplicação e média aritmética real e inteira.

6. Fazer um programa que leia do teclado um valor inteiro representando uma temperatura em graus celsius e converta-a para uma temperatura em farenheit. (fórmula: c= 5/9 (f-32)) .

7. Diga se as seguintes expressões serão verdadeiras ou falsas:
   - ((10>5)||(5>10))
   - (!(5==6)&&(5!=6)&&((2>1)||(5<=4)))

8. A linguagem C tem este nome porque foi a sucessora da linguagem B.
   - ❑ Verdadeiro
   - ❑ Falso

9. Em C, variáveis com nomes abc e Abc representam a mesma variável .
   - ❑ Verdadeiro
   - ❑ Falso

10. O programa

```
#include <stdio.h>
main()
{
int x;
scanf("%d",&x);
printf("%d",x);
}
```

    Lê uma variável pelo teclado e a imprime na tela
    - ❑ Verdadeiro
    - ❑ Falso

11. A instrução #include <stdio.h> no programa anterior é colocada para que possamos utilizar as funções **scanf e printf**
    - ❑ Verdadeiro
    - ❑ Falso

12. Os comentários na linguagem C só podem ter uma linha de comprimento
    - ❑ Verdadeiro
    - ❑ Falso

13. Sendo i uma variável inteira, a seguinte chamada a scanf é válida: scanf("%d", i);
- ❑ Verdadeiro
- ❑ Falso

14. O que faz o seguinte programa em C?

```
#include <stdio.h>
main()
{
int i =2;
printf ("\n O valor de i = %d ", i);
}
```

a) Nada
b) Imprime: O valor de i = 2
c) Imprime: \n O valor de i = %d
d) Pula para a próxima linha e imprime: O valor de i = 2

15. O comando printf ("%s%d%%","Juros de ",10); imprime:
a) Juros de 10%
b) %s%d%% Juros de 10
c) % Juros de 10
d) 10 Juros de
e) Nenhuma das anteriores

16. Para **a=1**, **b=2**, **c=3**, **d=4**, calcule o valor das expressões a seguir:

`b += a + c;` **Resposta**:

`b *= d = c + 20;` **Resposta**:

`a += b += c += 5;` **Resposta**:

`a == b` **Resposta**:

17. Qual a saída do programa ?

```
int a = 6, b = 5;
a = b++;
if ( a = b)
      b += a + 1;
printf ("%d",a - b);
```

Resposta: