3- ENTRADA E SAÍDA EM C

- Os principais meios para executar operações de entrada e saída (E/S) são:
  - Entrada e saída pelo console (ou padrão): teclado e tela.
  - Arquivos (leitura e escrita)

As funções de biblioteca são responsáveis pela E/S. As bibliotecas padrão mais utilizadas são stdio.h e conio.h.

3.1 - Funções para manipular dados de entrada e saída padrão

3.1.1 - Leitura e escrita de caracteres

- getchar( ): lê o próximo caracter em um fluxo de texto.

  Ex.: c = getchar( ) /* c recebe o caracter */

  - getchar( ) é uma MACRO definida em <stdio.h> e é derivada da função getc().

    #define getchar( ) getc(STDIN)
    obs.: STDIN é a entrada padrão

  - getchar( ) retorna EOF se ocorre algum erro. EOF também é uma MACRO, está definida em stdio.h e é igual a -1 <ctrl z>

- putchar( ): Imprime o conteúdo da variável c sob a forma de um caracter.

  Ex.: putchar(c);

  - putchar( ) também é uma MACRO definida em <stdio.h> e é derivada da função putc().

    #define putchar(c) put((c), STDOUT)
    obs.: STDOUT é a saída padrão

Exemplo:

```
#include <stdio.h>
main( )
{
        char ch;
          printf (“Primeiro programa \n”);
          ch = getchar( );    /*  espera um caracter  */
        putchar (ch);
}
```

Exemplo 2: Programa que lê um caracter e inverte a caixa deles, isto é, escreve maiúsculas como minúsculas e vice-versa.

```
#include <stdio.h>.
#include <ctype.h>
main( )
{
  char ch;
  printf( “Entre com um texto!  Ponto para sair. \n”);
  do
  {
        ch = getchar( );
        if  ( islower(ch) )
            ch = toupper(ch);
        else
                ch = tolower(ch);
        putchar(ch);
  }
  while (ch !=  ‘.’);
}
```

- gets( ) e puts( ) são funções para ler e escrever strings, as mesmas serão vistas no capítulo sobre *strings*.

3.1.2 - Saída formatada

- A função printf ( ) é responsável pela saída formatada.
- printf( ) especifica o que imprimir (variáveis, resultados de expressões, etc.) e como imprimir, isto é, a saída pode ser FORMATADA.

  Sintaxe:

  printf (<formato dos elementos>, < lista de elementos >);

- A primeira parte dos argumentos de printf( ) está entre "aspas" e corresponde aos comandos de formato.

  Ex.: printf (**"%d % 10.3f %d \n "**, i, a*b, 4);

- A segunda parte dos argumentos especifica a lista de elementos a serem impressos.

  Ex.: printf ("%d % 10.3f %d \n ", **i, a*b, 4**);

- O símbolo **%** converte o número ou caracter para a forma desejada.

  Exemplos:

```
printf ("%d ", s );        /*  %d indica que vai imprimir um inteiro */

printf ( "%10d ", s);    /*  imprime valor de s em 10 espaços */

printf ( "%10.5 f ", s);   /*  imprime um real em 10 espaços,
                              5 dos quais  depois do ponto : 4 algarismos, ponto, 5
                              algarismos. */

printf(" Olá ! Tudo bem ? \n "); /* imprime apenas a  frase */
```

- Algumas formatações:

  %c → caracter

%d, %i → inteiro decimal

%u → inteiro sem sinal

%f → ponto flutuante

%lf → ponto flutuante precisão dupla

%e, %E → real em notação científica

%s → cadeia de caracteres (*string*)

%p → endereço de uma variável

%o → octal

%x → hexadecimal

\n → nova linha

\t → tab

\\ → \ (barra)

\" → aspas

%% → % (porcentagem)

Exemplo:

Supor n = 18,
printf ("% 5d ", n); produz a saída **_ _ _ 1 8** .
printf ("% - 5d ", n); produz a saída **1 8 _ _ _** .

- É possível imprimir vários valores, a formatação dos mesmos deve ser adequada.

printf ("%d % 10.3f \n ", i, m);
inteiro real pula linha var inteira var real

- printf( ) devolve o número de caracteres impressos ou um número negativo no caso de ocorrer algum erro.

3.1.3 - Entrada formatada

- A função *scanf*( ) é responsável pela entrada formatada.

- *scanf*( ) é muito parecida com printf( ): lê os valores e interpreta-os de acordo com o formato especificado.

  scanf (<formato dos elementos>, < lista de elementos >);

- *scanf*( ) devolve o número de dados que foram lidos, se ocorrer algum erro, a mesma devolve EOF.

- Os símbolos utilizados para formatação são os mesmos usados na função printf( ).

- Na segunda parte dos argumentos, deve-se colocar o endereço & dos elementos a serem lidos. (passagem por referência)

Supor a leitura de três variáveis inteiras:

```
int dia, mês, ano;
scanf ( "%d  %d  %d", &dia, &mes, &ano);
```

obs.: é passado o endereço das variáveis onde os valores serão armazenados.

É equivalente a usar três comandos.

Entrar os dados usando brancos, enter, ou tabulações

- scanf( ) ignora espaços em branco e tabulações (\t).

scanf ("%d%d%d", &dia, &mes, &ano) também funciona.

se \n for colocado será necessário digitar <enter> e "EOF"

Exemplos:

```
scanf("% 2d", &x)
/*  Lê um inteiro de 2 dígitos e atribui seu valor a x */

scanf ("%f   %d", &num, &x);
/* lê um real e um inteiro, por exemplo   10.5   e   135 */

char mês[18];
scanf ("%s", mês)
/* não precisa o símbolo  & pois mês é nome de vetor -
    isto será explicado posteriormente */
```

para ler strings: funções gets() e puts (), serão vistas mais para frente

Exercícios:

1) Faça um programa para ler 2 números reais (double), um float e um inteiro, fornecidos em uma linha e imprimi-los em uma única linha também.

2) Faça um programa para ler e imprimir 3 inteiros em linhas separadas.

Para entregrar:

> tar cvf lista.tar *

> /home/html/inf/silvia/entrega/bin/entrega   oficinac 1 lista.tar

3.1.4 - Algumas considerações sobre a entrada de dados

- Em C toda entrada e saída é feita com fluxos de entrada (*streams*) de caracteres organizados em linhas. Cada linha consiste de zero ou mais caracteres e termina com o caracter de final de linha.

- Em ambiente Unix, digitando ^D (segure a tecla de Ctrl e pressione D) você diz ao programa que terminou a entrada de dados → EOF. (end of file, final de arquivo). Em ambiente MS-Windows, você faz isto digitando ^Z (segure a tecla de Ctrl e pressione Z).

- Pode haver até 254 caracteres em uma linha (incluindo o caracter de final de linha). Quando um programa inicia, o sistema operacional automaticamente define quem é a entrada padrão (geralmente o teclado) e quem é a saída padrão (geralmente a tela).

- A função getchar() retorna um valor, o caractere lido (mais precisamente, o código inteiro ASCII correspondente ao caractere).

- O getchar começa a ler depois que a tecla return ou enter é digitada no final de uma sequência de caracteres (dizemos que a entrada para a função getchar() está no *fluxo de entrada (stream)*).

Considere o seguinte programa:

```
#include <stdio.h>
main(){
   int ch;
   ch = getchar();
}
```

getchar() obtém sua entrada do teclado. Portanto, quando o programa acima é executado, o programa espera que o usuário digite alguma coisa. Cada caractere digitado é mostrado no monitor.

- O usuário pode digitar diversos caracteres na mesma linha, inclusive *backspace* para corrigir caracteres já digitados.

- No momento que return é digitado, o primeiro caractere da sequência digitada é o resultado da função getchar(). Portanto, na instrução do programa acima o caractere (ou melhor, o seu código ASCII) é atribuído a variável ch.

- Note que o usuário pode ter digitado diversos caracteres antes de teclar return, mas a função getchar() só começará a ler o que foi digitado depois que for teclado return. Além disso, com uma chamada da função getchar() só o primeiro caractere da sequência digitada é lido.

- Você deve saber que o caractere de nova linha, `\n`, que tem o código ASCII 10, é automaticamente adicionado na sequência de caracteres de entrada quando o return é teclado. Isso não tem importância quando a função getchar() é chamada uma única vez, mas isto pode causar problemas quando ele é usado dentro de um laço.

Exemplo 1: Considere o seguinte programa:

```
#include <stdio.h>
int main()
{
   int ch;
   printf( "Entre com uma letra: " );
   ch = getchar();
   if (ch < 'A' || ch > 'z' )
     printf( "Voce nao teclou uma letra!" );
   else
     printf( "Voce teclou %c, e seu codigo ASCII eh %d.\n", ch, ch );
}
```

Um exemplo da execução do programa:

```
Entre com uma letra: A
Voce teclou A, e seu codigo ASCII eh 65.
```

No exemplo de execução acima o usuário teclou A e depois return.

Outro exemplo de execução do programa:

```
Entre com uma letra: AbcD
Voce teclou A, e seu codigo  ASCII eh 65.
```

Neste caso o usuário digitou quatro caracteres e depois teclou return. Embora quatro caracteres tenham sido digitados, somente uma chamada a função getchar() foi feita pelo programa, portanto só um caractere foi lido. O valor atribuído ao argumento da função é o código ASCII do primeiro caractere lido.

Exemplo 2: O programa abaixo conta o número de caracteres digitados (incluindo o caractere de ``próxima linha"):

```
#include <stdio.h>

int main()
{
   int total = 0, ch;
   /* Le o proximo ch e para quando encontrar final do arquivo */
   while  ( (ch = getchar()) != EOF ) {
      total++;
   }
    printf( "\n%d caracteres digitados\n", total );
 }
```

Depois de entrar com o comando ^D você deve teclar return → quando você usa getchar(), você deve teclar return no final.

- Entretanto o caracter de nova linha continua no fluxo de entrada e se você executar um getchar() depois de um scanf() ou de um getchar() você lerá o caractere de nova linha deixado no fluxo de entrada.

- Da mesma forma, quando você usa scanf() para ler informações, ele somente lê o que é necessário. Se voce usar scanf() para ler um número inteiro e digitar 42 (seguido de return), o scanf() lê 42, mas deixa os 2 espaços em branco (e o caractere de nova linha do return no fluxo de entrada.

- Quando o scanf() é usado em um laço. Se você digitar um valor do tipo errado, o scanf() lerá o valor errado e a execução do laço continuará na sentença após o scanf(). Na próxima iteração do laço o scanf() vai tentar ler novamente, mas o ``lixo" deixado da iteração anterior ainda estará lá, e portanto a chamada corrente do scanf() também não dará certo.

- Há uma maneira simples de resolver este problema; toda vez que você usar getchar() (para ler um caracter só) ou scanf(), você deve ler todo o ``lixo" restante até o caractere de nova linha. Colocando as seguintes linhas após chamadas a getchar() ou scanf(), o problema é eliminado:

```
/* Pula o restante da linha */
while( getchar() != '\n' );
```

Note que isso não é necessário após todas as chamadas a getchar() ou scanf(). Só depois daquelas chamadas que precedem getchar() (ou scanf()), especialmente em um laço.

- A função scanf() retorna um inteiro que é o número de itens (valores) lidos com sucesso. Você pode verificar se o scanf() funcionou testando se o valor retornado é igual ao número de especificadores de formato no primeiro argumento da função.

```
int main()
{  int total = 0, num;

  while( total < 20 )
  {   printf( "Total = %d\n", total );
      printf( "Entre com um numero: " );
      if( scanf("%d", &num) < 1 )
         /* Ignora o resto da linha */
          while( getchar() != '\n' );
      else
          total += num;
  }
  printf( "Final total = %d\n", total );
}
```