Programação II
AULA 01 – Revisão de Programação I

1 Introdução à Linguagem C

O C é uma linguagem com Case Sensitive", isto é, maiúsculas e minúsculas fazem diferença. Se se declarar uma variável com o nome soma ela será diferente de Soma, SOMA, SoMa ou sOmA.

1.1 FUNÇÃO printf()

Sintaxe: *printf("expressão de controle", argumentos);*

É uma função de E/S, que permite escrever no dispositivo padrão (tela). Usa o caractere % seguido de uma letra para identificar o formato de impressão. A expressão de controle pode conter caracteres que serão exibidos na tela e os códigos de formatação que indicam o formato em que os argumentos devem ser impressos. Cada argumento deve ser separado por vírgula.

Código	Significado
%d	Inteiro
%f	Float
%c	Caractere
%s	String
%ld	Decimal longo
%lf	Ponto flutuante longo (Double)

Pode-se indicar o número de casas decimais de um número de ponto flutuante. Por exemplo, a notação %10.4f indica um ponto flutuante de comprimento total dez e com 4 casas decimais. Entretanto, esta mesma notação, quando aplicada a tipos como inteiros e *strings* indica o número mínimo e máximo de casas. Então %5.8d é um inteiro com comprimento mínimo de cinco e máximo de oito. O sinal menos (-) precedendo a especificação do tamanho do campo justifica os campos à esquerda.

Exemplos:

Código	Imprime
`printf ("%5.2f",456.671);`	`|  456.67|`
`printf ("%5.2f",2.671);`	`|    2.67|`
`printf ("%-10s","Ola");`	`|Ola       |`

1.2 CONSTANTES

São valores que são mantidos fixos e inalterados. Seu conteúdo não pode ser modificado durante a execução de um programa.

Sintaxe: *#define nome_constante valor*

1.3 VARIÁVEIS

Aspecto fundamental de qualquer linguagem de computador. É um espaço de memória reservado para armazenar certo tipo de dado e tendo um nome para referenciar o seu conteúdo. Em C, toda variável precisa ser declarada e o armazenamento de um valor é realizado através do comando de atribuição.

1.3.1 Declaração de Variáveis:

Deve ser efetuada antes da utilização da variável. Consiste em um nome de um tipo seguido do nome da variável e é finalizada com ";". Instrução usada para reservar uma quantidade de memória apropriada para um tipo específico de dado.

Sintaxe: <*tipo da variável*> *nome da variável*

As variáveis no C podem ter qualquer nome se duas condições forem satisfeitas:

- o nome deve começar com uma letra ou sublinhado (_);
- os caracteres subseqüentes devem ser letras, números ou sublinhado (_).

Há apenas mais duas restrições:

- o nome de uma variável não pode ser igual a uma palavra reservada, nem igual ao nome de uma função declarada pelo programador, ou pelas bibliotecas do C.
- variáveis de até 32 caracteres são aceitas.

Sugestões:

- Usar letras minúsculas para nomes de variáveis e maiúsculas para nomes de constantes. Isto facilita na hora da leitura do código.
- O tipo de uma variável informa a quantidade de memória, em bytes, que esta irá ocupar e a forma como o seu conteúdo será armazenado.
- C tem 5 tipos básicos: char, int, float, void, double.

O caracter é um tipo de dado: o char. O C trata os caracteres ('a', 'b', 'x', etc ...) como sendo variáveis de um byte (8 bits). Também podemos usar um char para armazenar valores numéricos inteiros, além de usá-lo para armazenar caracteres de texto. Para indicar um caractere de texto usamos apóstrofes.

String é uma cadeia de caracteres que será discutida posteriormente.

O double é o ponto flutuante duplo e pode ser visto como um ponto flutuante com muito mais precisão. O void é o tipo vazio, ou um "tipo sem tipo". A aplicação deste "tipo" será vista posteriormente.

Para cada um dos tipos de variáveis existem os modificadores de tipo. Os modificadores de tipo do C são quatro: signed, unsigned, long e short. Ao float não se pode aplicar nenhum e ao double pode-se aplicar apenas o long. Os quatro modificadores podem ser aplicados a inteiros. A intenção é que short e long devam prover tamanhos diferentes de inteiros, onde isto for prático. Inteiros menores (short) ou maiores (long). int normalmente terá o tamanho natural para uma determinada máquina. Assim, numa máquina de 16 bits, int provavelmente terá 16 bits. Numa máquina de 32, int deverá ter 32 bits. Na verdade, cada compilador é livre para escolher tamanhos adequados para o seu próprio hardware, com a única restrição de que shorts ints e ints devem ocupar pelo menos 16 bits, longs ints pelo menos 32 bits, e short int não pode ser maior que int, que não pode ser maior que long int. O modificador unsigned serve para especificar variáveis sem sinal. Um unsigned int será um inteiro que assumirá apenas valores positivos. A seguir estão listados os tipos de dados permitidos e seu valores máximos e mínimos em um compilador típico para um hardware de 16 bits. Também nesta tabela está especificado o formato que deve ser utilizado para ler os tipos de dados com a função scanf():

Tipo	Num de bits	Formato para leitura com scanf	Intervalo	
			Inicio	Fim
char	8	%c	-128	127
unsigned char	8	%c	0	255
signed char	8	%c	-128	127
int	16	%i	-32.768	32.767
unsigned int	16	%u	0	65.535
signed int	16	%i	-32.768	32.767
short int	16	%hi	-32.768	32.767
unsigned short int	16	%hu	0	65.535
signed short int	16	%hi	-32.768	32.767
long int	32	%li	-2.147.483.648	2.147.483.647
signed long int	32	%li	-2.147.483.648	2.147.483.647
unsigned long int	32	%lu	0	4.294.967.295
float	32	%f	3,4E-38	3.4E+38
double	64	%lf	1,7E-308	1,7E+308
long double	80	%Lf	3,4E-4932	3,4E+4932

O tipo long double é o tipo de ponto flutuante com maior precisão. É importante observar que os intervalos de ponto flutuante, na tabela acima, estão indicados em faixa de expoente, mas os números podem assumir valores tanto positivos quanto negativos.

1.3.2 Atribuição:

Comando através do qual um valor é atribuído a uma variável. O operador de atribuição é o sinal "=". Sempre finalizado com ";".

A interpretação do sinal = na linguagem C é diferente da matemática. Ele representa a atribuição da experessão à direita ao nome da variável à esquerda.

Exemplos:

```
int A,B; /* duas variáveis do tipo inteiro são declaradas *8
A=12; /* o valor inteiro 12 é atribuído para a variável A */
A=23; /* o valor 12 foi perdido e agora A está com o valor 23 */
B=A; /* recebe o mesmo valor de A*/
```

Observações:

- 23=A; é um comando que não tem sentido em C pois é impossível atribuir uma valor a uma constante.
- A=B=C=D=43; a linguagem C aceita várias atribuições na mesma instrução.

1.3.3 Inicializando Variáveis:

É possível combinar uma declaração de variável com o operador de atribuição para que a variável já tenha uma valor a partir da sua declaração, é o que é definido como inicialização de variável.

Exemplo:

```
int A=12;
char resp='S';
```

1.4 FUNÇÃO scanf()

Também é uma função de E/S implementada em todos compiladores C. Ela é o complemento de printf() e nos permite ler dados formatados da entrada padrão (teclado). Sua sintaxe é similar a printf(), sendo utilizado os mesmos códigos.

Sintaxe: *scanf("expressão de controle", argumentos);*

1.5 OPERADORES ARITMÉTICOS

Os operadores aritméticos são usados para desenvolver operações matemáticas:

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:

- unários agem sobre uma variável apenas, modificando ou não o seu valor, e retornam o valor final da variável.
- 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". 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. Oque eles fazem é incrementar ou decrementar, a variável sobre a qual estão aplicados, de 1. Assim:

x++; ... é equivalente a .... x = x + 1;

x--; ... é equivalente a .... 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.

1.6 OPERADORES ARITMÉTICOS DE ATRIBUIÇÃO

+=, - =, *=, /=, % =

Cada um destes operadores é usado com um nome de variável à sua esquerda e uma expressão à sua direita. A operação consiste em atribuir um novo valor à variável que dependerá do operador e da expressão à direita.

Exemplos:

`i += 2;`	equivale a	`i = i+2;`
`x *= y+1;`	equivale a	`x = x*(y+1);`
`t /= 2.5;`	equivale a	`t = t/2.5;`
`p%=5;`	equivale a	`p =p%5;`
`d -= 3;`	equivale a	`d = d-3;`

As expressões com estes operadores são mais compactas e normalmente produzem um código de máquina mais eficiente.

1.7 OPERADORES RELACIONAIS E OPERADORES 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):

```
int main() {
    int i, j;
    printf("\nEntre com dois numeros inteiros: ");
    scanf("%d%d", &i, &j);
    printf ("\n\n0 = falso e 1 = verdadeiro\n\n");
    printf("\n%d == %d resulta %d\n", i, j, i==j);
    printf("\n%d != %d resulta %d\n", i, j, i!=j);
```

```
    printf("\n%d <= %d resulta %d\n", i, j, i<=j);
    printf("\n%d >= %d resulta %d\n", i, j, i>=j);
    printf("\n%d < %d resulta %d\n", i, j, i<j);
    printf("\n%d > %d resulta %d\n", i, j, i>j);
    getch();
}
```

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:

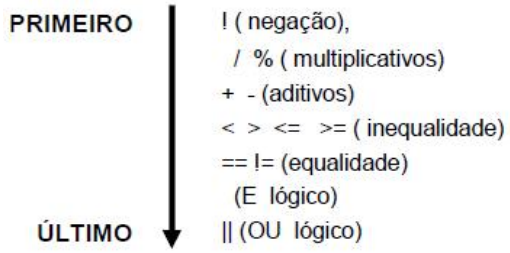
p	q	p AND q	p OR q
falso	falso	falso	falso
falso	verdadeiro	falso	verdadeiro
verdadeiro	falso	falso	verdadeiro
verdadeiro	verdadeiro	verdadeiro	verdadeiro

p	NÃO p
verdadeiro	falso
falso	verdadeiro

Exemplo:

```
int main(){
    int i, j;
    printf("Informe dois números (cada um sendo 0 ou 1): ");
    scanf("%d%d", &i, &j);
    printf ("\n\n0 = falso e 1 = verdadeiro\n\n");
    printf("%d AND %d resulta %d\n", i, j, i && j);
    printf("%d OR %d resulta %d\n", i, j, i || j);
    printf("NOT %d resulta %d\n", i, !i);
    getch();
}
```

PRECEDÊNCIA: os operadores aritméticos têm maior precedência que a dos relacionais. Isto significa que serão avaliados antes.



1.8 MODELADORES (*CASTS*)

Um modelador é aplicado a uma expressão. Ele força a mesma a ser de um tipo especificado. Sua forma geral é: (tipo)expressão

Um exemplo:

```
#include <stdio.h>
main ()
{
    int num;
    float f;
    num=10;
    f=(float)num/7;
    printf ("%f",f);
}
```

Se não tivéssemos usado o modelador no exemplo acima o C faria uma divisão inteira entre 10 e 7. O resultado seria um e este seria depois convertido para float mas continuaria a ser 1.0. Com o modelador temos o resultado correto.

2 Comando de Decisão (*if*)

Há duas variações do comando if:

2.1 DECISÃO SIMPLES: if (Se)

Tem a finalidade de tomar uma decisão e efetuar um desvio no processamento do programa, dependendo, é claro, de a condição ser Verdadeira ou Falsa.

Sendo a condição Verdadeira, será executada a instrução que estiver escrita após a instrução if. Caso seja necessário executar mais de uma instrução para uma condição verdadeira, elas deverão estar escritas dentro de um bloco, ou seja, devem estar ente "{" e "}".

Exemplo 1: Ler dois números. Se X for maior que 10, imprimir a variável X.

```
int main() {
     int A, B , X;
     printf ("Informe um valor para a variável A: "); scanf ("%d", &A);
     printf ("Informe um valor para a variável B: "); scanf ("%d", &B);
     X = A + B;
     if (X > 10)
          printf ("\nO resultado da variável X = %d, sendo maior que 10", X);
     getch();
}
```

Exemplo 2: Verificar de o valor de A é maior que o valor de B. Se for Verdadeiro, efetuar a troca de valores entre as variáveis. Se for Falso, apenas imprimir os valores das variáveis.

```
int main() {
     int A, B , X;
     printf ("Informe um valor para a variável A: "); scanf ("%d", &A);
     printf ("Informe um valor para a variável B: "); scanf ("%d", &B);
     if (A > B)
     {
         X = A;
         A = B;
         B = X;
     }
     printf ("\nO valor da variável A agora equivale: %d", A);
     printf ("\nO valor da variável B agora equivale: %d", B);
     getch();
}
```

2.2 DECISÃO COMPOSTA: if ... else (Se ... Senão)

Sendo a condição Verdadeira, será executada a instrução que estiver posicionada entre as instruções if e else. Sendo a condição Falsa, será executada a instrução que estiver posicionada logo após a instrução else.

Exemplo 1: Ler dois números, efetuar a soma deles e armazenar na variável X. Verificar se X for maior ou igual a 10, mostrar X + 5, senão mostrar X-7.

```
int main() {
     int A, B , X;
     printf (“Informe um valor para a variável A: “); scanf (“%d”, &A);
     printf (“Informe um valor para a variável B: “); scanf (“%d”, &B);
     X = A + B;
     if (X > 10)
         printf (“\n%d”, X+5);
     else
         printf (“\n%d”, X-7);
     getch();
}
```

Exemplo 2: Ler quatro notas. Calcular a média MD = (N1+N2+N3+N4) / 4. Se a média for maior ou igual a 7, apresentar a mensagem “Aluno Aprovado”; senão, apresentar a mensagem “Aluno Reprovado”.

```
int main() {
     float N1, N2, N3, N4, MD;
     printf (“Entre com a Nota 1: “); scanf (“%f”, &N1);
     printf (“Entre com a Nota 2: “); scanf (“%f”, &N2);
     printf (“Entre com a Nota 3: “); scanf (“%f”, &N3);
     printf (“Entre com a Nota 4: “); scanf (“%f”, &N4);
     MD = (N1 + N2 + N3 + N4) / 4;
     if (MD > 7)
         printf (“\nAluno Aprovado”);
     else
         printf (“\nAluno Reprovado”);
     printf (“Media = %.2f”, MD);
     getch();
}
```

3 Comando de Seleção (*switch*)

Neste comando a execução segue os seguintes passos:

1. A expressão é avaliada;

2. O resultado da expressão é comparado com os valores das constantes que aparecem nos comandos case;

3. Quando o resultado da expressão for igual a uma das constantes, a execução se inicia a partir do comando associado com esta constante. A execução continua com a execução de todos os comandos até o fim do comando switch, ou até que um comando break seja encontrado;

4. Caso não ocorra nenhuma coincidência o comando default é executado. O comando default é opcional e se ele não aparecer nenhum comando será executado.

O comando break é um dos comandos de desvio da linguagem C. O break se usa dentro do comando switch para interromper a execução e pular para o comando seguinte ao comando switch.

Sintaxe:

switch (variável) {
case constante1: seqüência de comandos;
break;
case constante2: seqüência de comandos;
break;
... ...
case constante_n: seqüência de comandos;
break;
default: seqüência de comandos;
}

Há alguns pontos importantes que devem ser mencionados sobre o comando switch.

- O resultado da expressão deve ser um tipo compatível com um inteiro, isto é, expressõecom resultados tipo char também podem ser usadas;
- Notar que caso não apareca um comando de desvio todas as instruções seguintes ao teste case que teve sucesso serão executadas, mesmo as que estejam relacionadas com outros testes case;
- O comando switch só pode testar igualdade;
- Não podem aparecer duas constantes iguais em um case;

Exemplo 1:

```
int main() { // inicio do programa
     int opcao;
     printf ("1. inclusão\n");
     printf ("2. alteração\n");
     printf ("3. exclusão\n");
     printf ("\nDigite sua opção:");
     scanf (“%d”,&opcao);
     switch (opcao) { // início do swtich
        case 1:
             printf ("voce escolheu inclusao\n");
             break;
        case 2:
             printf ("voce escolheu alteracao\n");
             break;
        case 3:
             printf ("voce escolheu exclusao\n");
             break;
        default: printf ("opcao invalida\n");
     } // fim do switch
     getch();
} // fim do programa principal
```

Exemplo 2:

```
int main() { // inicio do programa
     char opcao;
     printf ("I - Inclusão\n");
     printf ("A - Alteração\n");
     printf ("E - Exclusão\n");
     printf ("\nDigite sua opção:");
     scanf ("%C",&opcao);
     opcao=toupper(opcao); //transformar o caracter em letra maiuscula
     switch (opcao) { // início do swtich
          case 'I':
             printf ("voce escolheu inclusao\n");
             break;
          case 'A':
```

```
            printf ("voce escolheu alteracao\n");
            break;
        case 'E':
            printf ("voce escolheu exclusao\n");
            break;
        default: printf ("opcao invalida\n");
    } // fim do switch
    getch();
} // fim do programa principal
```

4 Comando de Repetição (for)

O comando for é um comando de repetição (também chamado de iteração ou laço).

Os comandos de repetição permitem que um conjunto de instruções seja executado até que satisfaça uma determinada condição.

Sintaxe:

for (inicialização ; condição ; incremento)
{
    comandos;
}

Normalmente, a inicialização é realizada através de um comando de atribuição, que é usado para colocar um valor na variável de controle do laço. A condição determina quando a repetição acaba. O incremento define como a variável de controle do laço varia cada vez que o laço é repetido.

Exemplo 1: Este programa imprime os números de 1 a 20 na tela

```
#include <stdio.h>
#include <conio.h>

int main()
{
      int x;
      for (x=1; x <=20; x++)
      {
            printf (“%d  “, x);
      }
      getch();
      return 0;
}
```

No programa acima, a variável x é inicialmente ajustada para 1 (inicialização). Uma vez que x é menor que 20 (condição), o comando printf é executado. A variável x é incrementada em 1 (incremento), e é testado se o valor de x ainda é menor ou igual a 20. Esse processo se repete até que o valor de x fique maior que 20. Nesse instante, o laço termina e o programa segue a execução das instruções abaixo do laço.

Exemplo 2: Este programa imprime os números de 1 a 20 na tela em ordem decrescente.

```
#include <stdio.h>
#include <conio.h>

int main()
{
      int x;
      for (x=20; x >=1; x--)
```

```
      {
            printf (“%d  “, x);
      }
      getch();
      return 0;
}
```

5 Comando de Repetição com condição no início (*while*)

Esta estrutura de laço de repetição caracteriza-se por efetuar um teste lógico no início do laço de repetição, verificando se é permitido executar o trecho de instruções subordinado a ele.

A estrutura while tem o seu funcionamento controlado por condição. Desta forma, pode executar um determinado conjunto de instruções enquanto a condição verificada permanecer Verdadeira. No momento em que esta condição se torna Falsa, o processamento da rotina é desviado para fora do laço de repetição.

Caso seja a condição Falsa logo no início do laço de repetição, as instruções contidas nele são ignoradas. Caso seja necessário executar mais de uma instrução para uma condição verdadeira dentro de um laço, elas devem estar definidas dentro de um bloco por meio dos símbolos de chaves.

Sintaxe:

while (<condição>)
{
    Instruções para condição verdadeira;
}

Exemplo 1: Programa que imprime os números de 1 a 20.

```
#include <stdio.h>
#include <conio.h>

int main()
{
      int num=1;

      while (num<=20)
      {                              // inicio do while
            printf (“%d  “, num);
            num=num+1;               // num++
      }                              // fim do while

      getch();
      return 0;
}
```

Exemplo 2: Programa que imprime os números de 1 a 20, identificando-os sendo par ou ímpar.

```
#include <stdio.h>
#include <conio.h>

int main() {
      int num=1;

      while (num<=20)   {                  //inicio do while
            if ( ( num % 2 ) == 0 ) {
                  printf  (“\n O número %d = par”, num);
            }
            else  {
                  printf  (“\n O número %d = impar”, num);
```

```
            }
            num++;
      }                                                    // fim do while

      getch();
      return 0;
}
```

Em muitos algoritmos é necessário processar-se um conjunto de dados, e ao mesmo tempo contar o número de elementos deste conjunto.

Para realizarmos esta contagem, devemos utilizar uma variável especialmente criada para este fim. Uma variável utilizada para este fim é chamada de cont.

Exemplo 3: Programa que verifica os números pares e ímpares existentes de 0 a 500, e mostre o resultado de quantos números pares e ímpares existem neste intervalo.

```
#include <stdio.h>
#include <conio.h>

int main() {
      int num=0, cont_par=0, cont_impar=0;

      while (num<=500)  {             //inicio do while
            if ( ( num % 2 ) == 0 ) {
                  cont_par++;
            }
            else {
                  cont_impar++;
            }
            num++;
      }                               // fim do while

      printf (“No intervalo de 0 a 500 existem: \n”);
      printf (“%d números pares e %d números impares.”,cont_par, cont_impar);

      getch();
      return 0;
}
```

6 Comando de Repetição com condição no fim (*do...while*)

Esta estrutura faz um teste lógico no final de um laço de repetição. Ela é parecida com a estrutura while. Seu funcionamento é controlado também por condição. Esse tipo de laço executa um conjunto de instruções pelo menos uma vez antes de verificar a validade da condição estabelecida. Diferente da estrutura while que executa somente um conjunto de instruções enquanto a condição é verdadeira.

Desta forma do...while irá processar um conjunto de instruções, no mínimo uma vez, até enquanto a condição for verdadeira. Então, a instrução do... while deve ser escrita:

Sintaxe:

```
do{
      Realiza instruções enquanto verdadeiras;
}
while (<condição>);
```

Exemplo 4: Programa que pede a leitura de dois valores inteiros para o usuário, efetua a adição e apresenta o resultado por cinco vezes.

```
#include <stdio.h>
#include <conio.h>

int main() {
      int A, B, RESULT, i=6;

      do
      {
            printf (“Digite um valor para A: ”); scanf (“%d”, &A);
            printf (“Digite um valor para B: ”); scanf (“%d”, &B);
            RESULT = A + B;
            printf (“\n O resultado correspondente = %d ”, RESULT);
            i++;
      }
      while (i<=5);

      getch();
      return 0;
}
```

EXERCÍCIO PARA LABORATÓRIO

EL_01) Faça um programa completo em C que solicita a digitação da quantidade de linhas que se deseja que tenha o triângulo a ser exibido na tela, escreve na tela um triângulo formado por letras 'O', conforme abaixo à esquerda (note que a quantidade de 'O' por linha vai aumentando 1 a cada linha).
O triângulo da direita está presente no exercício apenas para ilustrar, com as ↓s, o local onde um espaço em branco deveria ser escrito para que o triângulo tenha a aparência desejada (que fique claro que nenhuma ↓ deve de fato ser escrita).

```
    O                ↓...↓↓↓↓↓O
   O O               ↓...↓↓↓↓O O
  O O O              ↓...↓↓↓O O O
 O O O O             ↓...↓↓O O O O
O O O O O            ↓...↓O O O O O
```

Obs: Neste exercício, você deverá utilizar alguns conceitos vistos nesta aula de revisão e também o conceito de laços de repetição. Ou seja, deverá utilizar um dos comandos de repetição vistos em Programação I (FOR, WHILE ou DO...WHILE) e que será assunto de nossa próxima aula de revisão.

EL_02) Utilizando o mesmo conceito anterior, solicite ao usuário um número e com este número construa um quadrado conforme exemplo abaixo:

```
O-----O
-     -
-     -
-     -
O-----O
```

DESAFIO

ED_02) Faça com que um número natural Q≥3 (que expressa uma quantidade) seja digitado. Em seguida, Q linhas com espaços em branco e letras O apropriadamente intercalados devem ser escritas na tela, de modo a formar um X, conforme podemos observar à esquerda, em cada um dos exemplos abaixo. Note que:

a) A quantidade de espaços em branco entre as letras O de cada linha vai diminuindo de 2 em 2 e a quantidade de espaços antes da primeira letra O de cada linha vai aumentando de 1 em 1 até a metade do X; e

b) A quantidade de espaços em branco entre as letras O de cada linha vai aumentando de 2 em 2 e a quantidade de espaços antes da primeira letra O de cada linha vai diminuindo de 1 em 1 após a metade do X até concluir sua formação).

Observe atentamente os exemplos abaixo que são respectivamente, para quando for digitado 5 (primeiro exemplo) e 6 (segundo exemplo). Note que quando a quantidade de linhas for par, no centro do X haverá a repetição de duas linhas (sempre contendo duas letras O consecutivas). Já quando a quantidade de linhas for ímpar, essa repetição não ocorrerá; teremos no centro do X uma linha contendo uma única letra O.

Em cada um dos exemplos abaixo, o X da direita está presente apenas para ilustrar, com as ↓s, o local onde um espaço em branco deveria ser escrito para que o X tenha a aparência desejada (que fique claro que nenhuma ↓ deve de fato ser escrita).

```
O   O      O↓↓↓O
 O O       ↓O↓O
  O        ↓↓O
 O O       ↓O↓O
O   O      O↓↓↓O
```

Exemplo de Quantidade de Linhas IMPAR

```
O     O    O↓↓↓↓O
 O   O     ↓O↓↓O
  OO       ↓↓OO
  OO       ↓↓OO
 O   O     ↓O↓↓O
O     O    O↓↓↓↓O
```

Exemplo de Quantidade de Linhas PAR

Em cada um dos exemplos abaixo, o X da direita está presente apenas para ilustrar, com as ↓s, o local onde um espaço em branco deveria ser escrito para que o X tenha a aparência desejada (que fique claro que nenhuma ↓ deve de fato ser escrita).