Estruturas de Repetição

- Os programas passam a maior parte do tempo repetindo tarefas até que uma condição seja satisfeita (ou um número fixo de vezes).

for

- Sintaxe: `for (inicialização; condição; incremento) comando;`
  - *inicialização* – comando de atribuição para a variável de controle de término do loop
  - *condição* – proposição que testa a variável de controle para determinar quando o loop deve terminar
    - caso a `condição` não seja verdadeira logo no início, o bloco de comandos do loop não é executado nem mesmo uma única vez
  - *incremento* – maneira como a variável de controle deve ser alterada a cada repetição do loop
- Exemplos:

```
for (i = 0; i < 10; i++)
   printf("%d\n", i);
```

```
for (i = 0; i < 10; i++) {
   soma = soma + i;
   printf("%d\n", soma);
}
```

- Como nenhum dos três parâmetros do `for` é necessário, pode-se construir loops infinitos como a seguir (um loop infinito também pode ser construído quando o incremento não converge a variável de controle para a condição de término do loop)

```
for (;;)
   printf("loop infinito\n");
```

- Exemplo de uso de `for`'s aninhados (neste exemplo, as variáveis `linha` e `coluna` foram declaradas na própria estrutura de loop `for`)

```
for (int linha = 1; linha < 10; linha++) {
   for (int coluna = 1; coluna < 10; coluna++) {
      printf("%d%d ", linha, coluna);
   }
   printf("\n");
}
```

while

- Sintaxe: `while (condição) comando;`
  - `comando` é executado até que a `condição` não seja mais verdadeira
  - caso a `condição` não seja verdadeira logo no início, o bloco de comandos do loop não é executado nem mesmo uma única vez

- Exemplos:

```
while (i < 10)
   printf(“%d\n”, i++);
```

```
while (i < 10) {
   soma = soma + i;
   printf(“%d\n”, soma);
   i++;
}
```

do while

- Sintaxe:

```
do {
   comando;
} while(condição);
```

  - o `comando` é executado até que a `condição` não seja mais verdadeira
  - ao contrário das estruturas de repetição `for` e `while`, o `do while` executa pelo menos uma vez o bloco de comandos, mesmo que a `condição` não seja verdadeira logo no início.

- Exemplo:

```
char opcao = ‘1’; // para exemplificar 1ª execução incondicional do do-while
char lixo;

do {
   printf("1. Incluir\n");
   printf("2. Excluir\n");
   opcao = getchar();
   lixo = getchar(); // para ler o [ENTER] depois do numero
   switch(opcao) {
      case '1':
         printf("Escolheu incluir\n");
         break;
      case '2':
         printf("Escolheu excluir\n");
         break;
   }

} while ( (opcao != '1') && (opcao != '2') );
```

Exercício: Fazer um programa em linguagem C para cada um dos seguintes problemas:

a) *Menor da lista* – Receber um inteiro positivo não nulo correspondente ao *tamanho* da lista de números a ser fornecida pelo usuário. Em seguida, receber todos os números da lista e devolver o menor deles.

```
#include <stdio.h>
int main() {
   int tamanho, indice, numero, menor;

   scanf("%d", &tamanho);
   if (tamanho > 0)
   {
      scanf("%d", &menor);
      for (indice = 2; indice <= tamanho; indice++)
      {
         scanf("%d", &numero);
         if (numero < menor)
            menor = numero;
      }
      printf("O menor número é %d", menor);
   }
   else
      printf("Lista vazia");
   return 0;
}
```

b) *Quadrado* – Receber um inteiro positivo não nulo $i < 20$ e imprimir um quadrado de lado *i* formado apenas por símbolos de asterisco (“*”). Se *i* não for positivo não nulo menor que 20, emitir um aviso ao usuário e repetir o processo (use uma constante para o limite do inteiro).

```
#include <stdio.h>
int main() {
   const int LIMITE = 20;
   int linha, coluna;
   int i;

   do {
      scanf("%d", &i);
      if ((i > 0) && (i < LIMITE))
      {
         for (linha = 1; linha <= i; linha++) {
            for (coluna = 1; coluna <= i; coluna++)
               printf("*");
            printf("\n");
         }
      }
      else
         printf("Numero deve ser positivo nao nulo menor que %d\n", LIMITE);
   } while ((i <=0) || (i >= LIMITE));
   return 0;
}
```

c) *Triângulo reto* – Receber um inteiro positivo não nulo $i < 20$ e imprimir um triângulo reto formado apenas por símbolos de asterisco (“*”) e de espaço (“ ”) como no exemplo a seguir (para $i = 5$):

```
*
**
***
****
*****
```

Se *i* não for positivo não nulo menor que 20, emitir um aviso ao usuário e repetir o processo (use uma constante para o limite do inteiro).

```
#include <stdio.h>

int main()
{
   const int LIMITE = 20;
   int linha, coluna;
   int i, asterisco, espaco;

   do {
      scanf("%d", &i);
      if ((i > 0) && (i < LIMITE))
      {
         asterisco = 0;
         for (linha = 1; linha <= i; linha++) {
            asterisco++;
                espaco = i - asterisco;
            for (coluna = 1; coluna <= asterisco; coluna++)
               printf("*");
            for (coluna = 1; coluna <= espaco; coluna++)
               printf(" ");
            printf("\n");
         }
      }
      else
         printf("Numero deve ser positivo nao nulo menor que %d\n", LIMITE);

   } while ((i <=0) || (i >= LIMITE));

   return 0;
}
```

d) *Conta caracteres* – Contar os caracteres de uma linha de texto fornecida via teclado.

e) *Fatorial* – Receber um inteiro positivo não nulo e calcular iterativamente seu fatorial.

f) *Máximo Divisor Comum (MDC)* – Receber dois inteiros positivos não nulos e calcular o MDC entre eles.

g) *Maior segmento homogêneo* – Receber um número natural correspondente ao *tamanho* da lista de números a ser fornecida pelo usuário. Em seguida, receber todos os números da lista e retornar o tamanho do maior segmento homogêneo. Exemplo:

20 12 22 22 15 44 44 44 63 63 75 34 44 76 76 76 76 76 5 5 5

Neste caso, o programa deveria identificar que a lista possui 20 números e retornar o valor 5, correspondente ao maior segmento homogêneo (composto de 76's).

h) *Raiz quadrada* – Receber um inteiro positivo não nulo e calcular iterativamente sua raiz quadrada aproximada. A raiz quadrada de um número *n* é igual à quantidade de números ímpares consecutivos (a partir do 1) que, somados, resultam *n*. Por exemplo, a raiz quadrada de

- 25 é 5 (1 + 3 + 5 + 7 + 9)
- 36 é 6 (1 + 3 + 5 + 7 + 9 + 11)
- 30 é 5, pois 25 < 30 < 36

i) *Triângulo reto invertido* – As mesmas especificações do problema "Triângulo reto", porém com a seguinte saída na tela (para *i* = 3):

```
  *
 **
***
```

j) *Triângulo eqüilátero* – Receber um inteiro ímpar positivo não nulo *i* < 20 e imprimir um triângulo eqüilátero formado apenas por símbolos de asterisco ("*") e de espaço (" ") como a seguir (para *i* = 7):

```
    *
   ***
  *****
 *******
```

Se *i* não for ímpar positivo não nulo menor que 20, emitir um aviso ao usuário e repetir o processo (use uma constante para o limite do inteiro).

k) *Losango* – Receber um inteiro ímpar positivo não nulo *i* < 20 e imprimir um losango formado apenas por pontos (".") envolvidos por asteriscos ("*") como a seguir (para *i* = 5):

```
*******
***.***
**...**
*.....*
**...**
***.***
*******
```

Se *i* não for ímpar positivo não nulo menor que 20, emitir um aviso ao usuário e repetir o processo (use uma constante para o limite do inteiro).