ESTRUTURAS DE CONTROLE

Os comandos de controle de fluxo são a essência de qualquer linguagem de programação, pois governam o fluxo da execução do programa. É possível dividi-los em três categorias. A primeira categoria consiste em instruções condicionais (`if` e `switch`). A segunda são os comandos de controle de *loop* (`for`, `while` e `do-while`). A terceira contém instruções de desvio incondicional (`goto`).

ESTRUTURA CONDICIONAL

A estrutura condicional permite a escolha de ações a serem executadas quando determinadas condições (*expressões lógicas*) são ou não satisfeitas.

Sintaxe:

```
if (condicao) comando-1;
```

ou

```
if (condicao) comando-1;
else comando-2;
```

ou

```
if (condicao) {
  comando-1;
  comando-2;
  /* ... */
  comando-n;
}
```

ou

```
if (condicao) {
  comando-1;
  comando-2;
  /* ... */
  comando-n;
}
else  {
  comando-1;
  comando-2;
  /* ... */
  comando-n;
}
```

Se a `condicao`, que pode ser uma operação relacional, avaliar em verdadeiro (*qualquer coisa menos 0*), será executado o comando-1 ou o bloco, de outro modo, se a cláusula `else` existir, será executado o comando-2 ou o bloco que é seu objetivo.

O `else` é opcional quando se tratar de uma condição simples de teste.

Exemplo:

```
#include <stdio.h>
#include <conio.h>
int main () {
    int a, b;
    printf ("\nDigite dois numeros inteiros: ");
    scanf ("%d %d", &a, &b);
    if (b != 0) // if (b)
      printf ("\nDivisao = %f",(float)a/b);
    else
      printf ("\nDivisao por zero.");
    getch();
}
```

ESTRUTURA CONDICIONAL ENCADEADA

Dentro de uma estrutura condicional, podem-se ter várias estruturas condicionais.

Sintaxe:
```
if (condicao) comando-1;
   else if (condicao) comando-2;
        else if (condicao) comando-3;
             else /* ... */
```

Exercícios

1 Desenvolva um programa que leia um número inteiro e verifique se o número é par ou ímpar.

2 Escreva um programa em linguagem C para ler 3 valores distintos (considere que não serão informados valores iguais) e escrever a soma dos 2 maiores.

3 Faça um programa para ler o número de lados de um polígono regular, e a medida do lado. O programa deve calcular e imprimir o seguinte:
- Se o número de lados for igual a 3 escrever TRIÂNGULO e o valor do seu perímetro;
- Se o número de lados for igual a 4 escrever QUADRADO e o valor da sua área;
- Se o número de lados for igual a 5 escrever PENTÁGONO.
- Em qualquer outra situação escrever Polígono não identificado.

4 Escreva um programa que leia as medidas dos lados de um triângulo e escreva se ele é EQUILÁTERO, ISÓSCELES ou ESCALENO.
Observação:

- Triângulo equilátero: Possui os 3 lados iguais.
- Triângulo isósceles: Possui 2 lados iguais.
- Triângulo escaleno: Possui 3 lados diferentes.

5 Escreva um programa que leia o valor de 3 ângulos de um triângulo e escreva se o triângulo é acutângulo, retângulo ou obtusângulo.
Observação:
- Triângulo retângulo: possui um ângulo reto (90 graus).
- Triângulo obtusângulo: possui um ângulo obtuso (ângulo maior que 90 graus).
- Triângulo acutângulo: possui 3 ângulos agudos (ângulo menor que 90 graus).

6 Escreva um programa em C que leia a idade de 2 homens e 2 mulheres (considere que a idade dos homens será sempre diferente, assim como das mulheres). Calcule e escreva a soma das idades do homem mais velho com a mulher mais nova, e o produto das idades do homem mais novo com a mulher mais velha.

7 Escreva um programa em C que leia as notas das 2 avaliações normais e a nota da avaliação optativa. Caso o aluno não tenha feito a optativa deve ser fornecido um valor negativo. Calcular a média do semestre considerando que a prova optativa substitui a nota mais baixa entre as 2 primeiras avaliações. Escreva a média e uma mensagem indicando se o aluno foi aprovado (media ≥ 5), reprovado (media < 3) ou está em exame (3 ≤ media < 5).

8 Faça um programa que receba os coeficientes a, b e c da equação do segundo grau, calcule e imprima suas raízes reais (caso existam) com precisão de 4 casas decimais.

Fórmula geral: $ax^2 + bx + c = 0$, com $a \in R^*$ e $b, c \in R$.
Cálculo do Delta: $b^2 - 4ac$.
Fórmula de Baskara: $x = (-b \pm \sqrt{\Delta})/(2a)$.
Raízes:
Se $\Delta < 0$, não existem raízes reais.
Se $\Delta = 0$, existem duas raízes reais e iguais.
Se $\Delta > 0$, existem duas raízes reais e distintas.

OPERADOR TERNÁRIO

É uma maneira compacta da expressão `if-else`.

Sintaxe: `condicao ? expressao-1 : expressao-2;`

Exemplo:

```
#include <stdio.h>
#include <conio.h>
int main () {
    float x, y, min;
    printf ("\nDigite dois numeros: ");
    scanf ("%f %f", &x, &y);
    min = (x < y)? x : y;
    printf ("\nO menor valor e' %f",min);
    getch();
}
```

OPERADOR switch

É uma instrução que permite a seleção de várias opções que dependam do resultado de uma condição ou entrada pelo usuário. Esta instrução pode substituir uma sequencia de condicionais `if` encadeados.

Sintaxe:
```
switch (expressao) {
          case constante-1: instrucao;
                            break;
          case constante-2: instrucao;
                            break;
          case constante-3: instrucao;
                            break;
          /* ... */
          case constante-n: instrucao;
                            break;
          default: instrucoes;
       }
```

O teste é feito para `n` condições. Se não foi encontrada nenhuma das condições executa-se o `default`. O `default` é opcional.

A declaração `break` utilizada em cada `case` faz com que o fluxo do programa saia da declaração `switch`. Caso não seja incluída a declaração `break`, todas as declarações abaixo da coincidência (inclusive) serão executadas.

Exemplo:

```
#include <stdio.h>
#include <conio.h>
int main ( ) {
  int a, b, op;
  printf ("Digite dois numeros inteiros: ");
  scanf ("%d %d",&a,&b);
  printf("\n1. Adicao \n");
  printf("2. Subtracao \n");
  printf("3. Multiplicacao \n");
  printf("4. Divisao \n");
  printf("\nDigite sua opcao: ");
  scanf ("%d",&op);
  switch(op) {
    case 1: printf("\n%d + %d = %d",a,b,a+b);
            break;
    case 2: printf("\n%d - %d = %d",a,b,a-b);
            break;
    case 3: printf("\n%d * %d = %d",a,b,a*b);
            break;
    case 4: if (b) printf("\n%d/%d= %f",a,b,(float)a/b);
            else printf ("\nDivisao por zero!");
            break;
    default: printf ("\nOpcao invalida!");
  }
  getch();
}
```

ESTRUTURA DE REPETIÇÃO

Muitas vezes tem-se a necessidade de repetir uma sequência de ações primitivas (bloco). O número de repetições pode ser indeterminado, porém finito (caso contrário o algoritmo não teria sentido, pois sua execução gastaria um tempo infinito).

Pode-se classificar tal estrutura como:

- **Repetição(*) contada**: Quando se conhece previamente quantas vezes o comando no interior da construção será executado.

- **Repetição condicionada**: Quando não se sabe de antemão o número de vezes que os comandos do laço serão repetidos pelo fato do mesmo estar "amarrado" a uma condição sujeita a modificação feita pelas instruções do interior do laço. Pode ser com interrupção no início ou no final.

(*) *loop*: laço de repetição

REPETIÇÃO CONTADA

Estrutura `for`

O comando `for` permite que um comando ou bloco de comandos seja repetido um número específico de vezes.

Sintaxe: `for (iniciacao; condicao; incremento) comando;`

ou

```
for (iniciacao; condicao; incremento) {
    comando-1;
    comando-2;
    /* ... */
}
```

Em sua forma mais simples, a `iniciação` é um comando de atribuição que o compilador usa para estabelecer a variável de controle do *loop*. A `condição` é uma expressão de relação que testa a variável de controle do *loop* contra algum valor para determinar quando o *loop* terminará. O `incremento` define a maneira como a variável de controle do *loop* será alterada cada vez que o computador repetir a sequência de comandos.

O comando `for` executa a iniciação incondicionalmente e testa a condição. Se a `condição` for falsa ele não faz mais nada. Se a `condição` for verdadeira ele executa a instrução, ou um bloco de instruções, faz o incremento e volta a testar a `condição`. Ele fica repetindo estas operações até que a condição seja falsa.

Exemplos:

1)
```
#include <stdio.h>
int main () {
  int i;
  for (i = 1; i <= 10; i++)
    printf ("Bem vindos!\n");
}
```

2)
```
#include <stdio.h>
int main() {
  int i;
  for (i = 100; i >= 80; i--)
    printf ("%d ",i);
}
```

3)
```
#include <stdio.h>
int main() {
  float x;
  for (x = 10; x >= 1; x-=0.2)
    printf ("%5.2f",x);
}
```

4)
```
#include <stdio.h>
int main () {
  for ( ; ; )
    printf ("loop infinito\n");
}
```

5)
```
#include <stdio.h>
int main() {
  int x, xq;
  printf ("\n\t Numero  Quadrado \n\n");
  for(x = 1; x < 15; x++) {
    xq = x*x;
    printf("\t %6d %9d \n", x, xq);
  }
}
```

6)
```
#include <stdio.h>
int main() {
  int x,y;
  for (x = 0, y = 0; x+y<20; ++x,++y)
    printf("%d + %d = %d\n", x, y, x+y);
}
```

7)
```
#include <stdio.h>
int main() {
  int linha,coluna;
  for(linha = 1; linha <= 15; linha++) {
    for(coluna=1; coluna<40; coluna++)
```

```
            printf("+");
          printf ("\n");
        }
    }
```

Exercícios

1) Escreva um programa que calcule o total da soma obtida dos cem primeiros números inteiros.

2) Faça um programa que leia os limites inferior e superior de um intervalo e imprima todos os números pares no intervalo aberto e seu somatório.

3) Escreva um programa que apresente o cubo dos números inteiros de 15 a 150.

4) Escreva um programa que mostre todos os números inteiros positivos divisíveis por 4 que sejam menores que 200.

5) Faça um programa que gere uma tabela de conversão de temperaturas em Fahrenheit para graus Celsius. Os valores inicial e final da temperatura em Fahrenheit deve ser fornecido pelo usuário. A variação da temperatura em Fahrenheit deve ser 0.5.

$$C = \frac{(F - 32)}{1.8}$$

6) Faça um programa que calcule e imprima o valor de S:

$$S = 1 + \frac{3}{2} + \frac{5}{3} + \frac{7}{4} + \ldots + \frac{99}{50}$$

7) O numero π pode ser calculado utilizando a seguinte série.

$$S = 1 - \frac{1}{3^3} + \frac{1}{5^3} - \frac{1}{7^3} + \cdots$$

Esta série é calculada para os N primeiros inteiros, sendo N um número determinado pelo usuário. Quanto maior N, melhor a precisão obtida. Escreva um programa que calcule e imprima o valor de π sabendo que

$$\pi = \sqrt[3]{32 * S}$$

REPETIÇÃO CONDICIONADA

Comando `while`

Uma maneira possível de executar um laço é utilizando o comando `while`. Ele permite que o código fique sendo executado numa mesma parte do programa de acordo com uma determinada condição.

• o comando pode ser vazio, simples ou bloco;

• ele é executado desde que a condição seja verdadeira;

• testa a condição **antes** de executar o laço.

Sintaxe: `while (condicao) comando;`

ou

```
while (condicao) {
    comando-1;
    comando-2;
    /* ... */
    comando-n;
}
```

A estrutura `while` é usada para repetir um ou mais comandos várias vezes, enquanto a `condicao` for verdadeira.

Se logo no primeiro teste a condição for falsa, os comandos não serão executados nenhuma vez. Deve-se ter em mente que é preciso que a condição imposta torne-se *falsa* pelo menos uma vez, senão a repetição dos comandos especificados não termina nunca (*loop* infinito).

Exemplos:

1)
```
#include <stdio.h>
int main () {
  int i = 1;
  while (i <= 10)
    printf ("%d ",i++);
}
```

2)
```
#include <stdio.h>
int main() {
  int x = 20;
  while (x != 0)
    printf ("loop infinito! ");
}
```

```
3) #include <stdio.h>
   int main() {
     int x = 20;
     while (x != 0)
     {
       printf ("\nloop finito");
       x--;
     }
   }
```

Comando `do-while`

O comando `do-while` é uma estrutura de repetição usada quando não se conhece o número de vezes que o loop se reproduz, mas se sabe que o corpo da iteração deve ser processado pelo menos uma vez.

A principal diferença entre os comandos `while` e `do-while` é que no comando `do-while` o conjunto de instruções do bloco será executado pelo menos uma vez, obrigatoriamente, enquanto no comando `while` pode acontecer do bloco de instruções não ser executado nenhuma vez.

Sintaxe:
```
do {
        comando-1;
        comando-2;
        /* ... */
        comando-n;
   } while (condicao);
```

Após a execução do grupo de comandos a condição é testada, podendo acontecer uma das situações a seguir:

- `condicao` ser **verdadeira:** o grupo de comandos é processado novamente e a condição avaliada novamente, podendo ocorrer sucessivas vezes o mesmo procedimento;

- `condicao` ser **falsa:** é assinalado o término das iterações e ocorre o desvio da execução para o comando abaixo do `do-while`.

Portanto, deve-se ter em mente que, para sair da estrutura `do-while` é preciso que a condição imposta se torne **falsa** pelo menos uma vez, senão o programa nunca termina (*loop* infinito).

Exemplos:

```
1) #include <stdio.h>
   int main () {
     int i = 1;
     do {
       printf ("%d ",i++);
     } while (i <= 10);
   }

2) #include <stdio.h>
   int main() {
     int x = 20;
     do {
       printf ("loop infinito! ");
     } while (x != 0);
   }

3) #include <stdio.h>
   int main() {
   int x = 20;
     do {
       printf ("loop finito! ");
       x--;
     } while (x != 0);
   }
```

INSTRUÇÃO `break`

Quando o comando **`break`** é encontrado em qualquer lugar do corpo do **for**, ele causa seu término imediato. O controle do programa passará então imediatamente para o código que segue o *loop*.

Exemplo:
```
#include <stdio.h>
int main() {
  int x;
  for(x = 0; x < 100; x++) {
    if (x == 20) break;
    printf ("%d\n", x);
  }
}
```

INSTRUÇÃO `continue`

Algumas vezes torna-se necessário "saltar" uma parte do programa, para isso utiliza-se o `continue`. O comando `continue` força a próxima iteração do *loop* (pula o código que estiver em seguida).

Exemplo:
```
#include <stdio.h>
int main() {
  int x;
  for(x = 0; x < 10; x++)
  {
    if (x%2) continue;
    printf ("%d\n", x);
  }
}
```

Exercícios

1) Solicite um número entre 1 e 4. Se a pessoas digitar um número diferente, mostrar a mensagem "entrada inválida" e solicitar o número novamente. Se digitar correto mostrar o número digitado.

2) Elabore um programa que efetue a leitura sucessiva de valores numéricos e apresente no final o total do somatório, a média e o total de valores lidos. O programa deve fazer as leituras dos valores enquanto o usuário estiver fornecendo valores positivos. Ou seja, o programa deve parar quando o usuário fornecer um valor negativo.

3) Elabore um programa que efetue a leitura de valores positivos inteiros até que um valor negativo seja informado. Ao final devem ser apresentados o maior e menor valores informados pelo usuário.

4) Faça um programa, com reprocessamento, que calcule o quociente e o resto da divisão de dois números inteiros usando apenas a adição e subtração.

5) Faça um programa que receba um valor *n* e calcule o valor de n! (fatorial de n). O programa deve permitir o reprocessamento.

$$0! = 1$$
$$n! = n * (n-1) * \ldots * 1$$