Escopo (visibilidade) das variáveis

- Declarando variáveis:
  - Variável local
    - Declarada dentro da função
  - Variável global
    - Declarada fora de qualquer função (inclusive da função main).
  - Argumento de função
    - Parâmetro da função (escopo local, dentro da função)

Escopo (visibilidade) das variáveis

- Exemplo de variável local

```
  funcao1( )
{
  int varlocal = 1;
}
/* --------------------- */
  funcao2( )
{
  int varlocal = -1;
}
```

As variáveis varlocal1 em funcao1 e funcao2 não tem nenhuma relação.

A variável local, é armazenada na pilha pelo compilador, sendo que só passa a existir, quando a função é utilizada, sendo "destruída" após o retorno ao bloco de código que efetuou a chamada. Exceto se for usado o modificador ***static***.

Escopo (visibilidade) das variáveis

Exemplo de variável global

```
#include <stdafx.h>
#include <stdlib.h>

int var;  /*var é global */
int funcao1(); /* protótipo */
int funcao2(); /* protótipo */
int main() {
    var = 1;
    funcao1();
    return 0;  }
int funcao1() {
  funcao2();
  /* var continua com o valor 1 */
  printf("*** Estamos na funcao1()\n");
  printf("var é global: %d\n", var);
  system("pause");
  return 0; }
int funcao2() {
  int var = 2;  /* var aqui é local */
  printf("*** Estamos na funcao2()\n");
  printf("var é local: %d\n", var);  /* var com o valor 2 */
  system("pause");
  return 0; }
```

Sempre que possível evite declarar variáveis globais.

Se variáveis local e global têm o mesmo nome, a função sempre referencia a variável local dentro do bloco de código e não a variável global.

Variáveis globais, são criadas em um local fixo da memória e existem durante todo o tempo de execução do programa.

Escopo (visibilidade) das variáveis

Exemplo de variável local *static*

```
Int ChamaSerial1();
Int ChamaSerial2();
int Serial();
main() {
    ChamaSerial1();
    ChamaSerial2();
    return 0;  }
int Serial () {
  static int NumSerial = 100;
  NumSerial = NumSerial + 10;
  return (NumSerial);  }
ChamaSerial1() {
  Serial();  /* Aqui NumSerial = 110 */
  return 0;  }
ChamaSerial2() {
  Serial(); /* Aqui NumSerial = 120 */
  Return 0;  }
```

A variável ***NumSerial*** terá seu valor preservado, apesar de só ter visibilidade dentro da função ***serial()***.

A variável local ***static*** pode ter seu valor iniciado; esse valor é atribuído somente uma vez.

Um pouco mais sobre Variáveis

```
#include "stdio.h"
#include <stdlib.h>

int main() {
  int oct;
  int hex;
  for (oct = 00; oct <= 0100; oct++) {
            /* Para imprimir em Octal: %o */
    printf("%o em Octal eh: %d em decimal \n", oct,oct);
  }
  system("pause");
  for (hex = 0x0; hex <= 0x100; hex++) {
            /* Para imprimir em Hexadecimal: %x */
    printf("%x em Hexa eh: %d em decimal \n", hex,hex);
  }
  system("pause");   }
```

Formatos:
Hexadecimal (16)
E Octal (8)

Funções

Forma geral de uma função

Tipo Identificador(TipoArg1 Arg1, ... ,TipoArg2 Arg2)
{
  CORPO DA FUNÇÃO
   [return <valor>;]
}

Funções

Exemplo:

```
int soma(int a,int b)
{
    return (a+b);
 }
```

Funções

- **Escopo da Função**
- **O corpo da função fica escondido do resto do programa**
- **Um argumento pode ser passado por valor, recebido por uma variável interna da função, chamado parâmetro formal**
- **Vetores são passados por referência ao endereço**

Funções - protótipos

A chamada a uma função deve vir, a princípio, após sua definição para o compilador conhecer os tipos de parâmetros e o tipo de retorno da função. Para isso declare apenas um protótipo da função, o qual tem apenas o valor de retorno e os parâmetros da função. Observe o exemplo abaixo:

```
#include <stdio.h>
int soma(int , int ); /* protótipo da função */
int main()
{
  x = soma(1,3)
{
```

Funções

- **Chamada por valor e chamada por referência**
  - A chamada por valor é a passagem normal do valor dos argumentos para a função. Utilizando esta chamada os valores dos argumentos passados não são modificados. Na realidade é passada uma cópia dos valores para a função.
  - Na chamada por referência são passados os endereços de memória onde estão os argumentos. Neste tipo de chamada os valores podem ser modificados.

Funções – por valor

```
/* Testando a chamada por valor */
#include <stdafx.h>
#include <stdlib.h>
/* Função com chamada por valor */
int valor(int a, int b) {
  a = a + 3; /* Modificando o primeiro argumento */
  b = b + 2; /* Modificando o segundo argumento */
  printf("Valores modificados dentro da função:\n");
  printf("nr1 = %d\n",a);
  printf("nr2 = %d\n",b);
  return 0;  }
int main() {
  int nr1 = 2, nr2 = 3;
  printf("\n");
  printf("Chamada por valor\n");
  printf("=================\n");
  printf("Valores iniciais de nr1 e nr2\n");
  printf("nr1 = %d\n",nr1);      printf("nr2 = %d\n",nr2);
  printf("\n\nChamando a função\n");
  valor(nr1,nr2); /* Na verdade a função recebe uma cópia destes argumentos */
  printf("\n\nValores após a chamada da função\n");
  printf("nr1 = %d\n",nr1);     printf("nr2 = %d\n",nr2);
  system("pause");     return(0);   }
```

Funções – por referência

```
#include <stdio.h>
#include <stdlib.h>
int valor();
/* Função com chamada por referência */
int valor(int *a, int *b)     {
    *a = *a + 3; /* Modificando o primeiro argumento */
    *b = *b + 2; /* Modificando o segundo argumento */
    printf("Valores modificados dentro da função:\n");
    printf("nr1 = %d\n",*a);      printf("nr2 = %d\n",*b);
    system("pause");   return (0);     }
int main()   {
    int nr1 = 2, nr2 = 3;
    printf("\n");
    printf("Chamada por referência\n");
    printf("======================\n");
    printf("Valores iniciais de nr1 e nr2\n");
    printf("nr1 = %d\n",nr1);      printf("nr2 = %d\n",nr2);
    valor(&nr1,&nr2); /* Neste tipo de chamada é passado o endereço do
                       * argumento. Neste tipo de chamada os valores
                       * podem ser modificados */
    printf("\n\nValores após a chamada da função\n");
    printf("nr1 = %d\n",nr1);      printf("nr2 = %d\n",nr2);
     system("pause");   return (0);     }
```

Funções

As variáveis criadas numa função são locais, assim serão destruídas após o término da função.

Caso você queira manter o valor de uma variável entre as chamadas a uma função você deve declarar esta variável como ***static***.

Funções - recursão

Recursão é o ato de uma função chamar ela mesma. Uma função que chama a ela mesma é chamada função recursiva.

O exemplo padrão de função recursiva é uma função que calcula o fatorial de um número. O fatorial de um número é igual ao produto dos números inteiros de 1 até o número.

Exemplo: fatorial de 5 = 5 * 4 * 3 * 2 * 1

Se você observar com cuidado verá que:

fatorial de 5 = 5 * fatorial de 4
fatorial de 4 = 4 * fatorial de 3
fatorial de 3 = 3 * fatorial de 2
fatorial de 2 = 2 * fatorial de 1

Ou seja

fatorial de um número = número * (fatorial de número - 1)

Exercícios

Exercício

Crie um programa que inicia uma variável com 1 e chama por 3 vezes a função ***soma1***, que deve sempre retornar a função principal, o valor anterior mais hum. Mantendo o valor da variável entre as chamadas da função.

Funções

```
/* Mantendo o valor de uma variável entre as chamadas de uma função */
#include <stdio.h>
int soma_1(int a); /* protótipo da função soma_1 */
int main() {
  int nr = 1;
  printf("Chamando a função a primeira vez: valor + 1 = %d\n",soma_1(nr));
  printf("Chamando a função pela segunda vez: : valor + 1 =
     %d\n",soma_1(nr));
  printf("Chamando a função pela terceira vez: : valor + 1 = %d\n",soma_(nr));
return(0);
}
int soma_1(int a) {
static int valor = 1;
printf("valor = %d\n",valor);
valor = valor + a;
return(valor);
}
```

Exercício - recursão

- Crie uma função recursiva que é chamada por um programa para cálculo do fatorial de um número inteiro.

Resolução

```
#include <stdafx.h>          /* Igual a “stdio.h” */
#include <stdlib.h>
/* Exemplo de função recursiva */
int fatorial(int nr) {
     int resposta;
     if(nr == 1)
       return(1);
     resposta = nr * fatorial(nr-1);
     return(resposta);
}
int main() {
    int a;
    printf("\nEntre com um valor inteiro :");
    scanf("%d",&a);
    printf("O fatorial de %d eh %d\n\n",a,fatorial(a));
    system("pause");
    return(0);
}
```

Exercício

Crie uma função que retorne o valor de x^y

Resolução

```
#include <stdio.h>
#include <stdlib.h>
int elevado(int, int);          /* protótipo da função elevado */
int main(){
  int b,e;
  printf("Digite a base e expoente x,y: ");
  scanf("%d,%d", &b,&e);
  printf("valor=%d\n",elevado(b,e) );
  system("PAUSE"); return 0; }
int (int base, int expoente) {
  int i=1;
  if (expoente == 0)
    return (1);
  if  (expoente == 1)
    return (base);
  for(;expoente;expoente--)
    i=base*i;
  return i;  }
```

Exercício - Funções

Escreva uma função que recebe como parâmetros dois números ***a*** e ***b*** e devolve o *mdc* (máximo divisor comum) de ***a*** e ***b***, calculado por meio do algoritmo de Euclides.

Pseudocódigo

```
AlgoritmoDeEuclides(inteiro a, inteiro b)
  dividendo ← a
  divisor ← b
  Enquanto resto(dividendo/divisor) ≠ 0
    c ← resto(dividendo/divisor)
    dividendo ← divisor
    divisor ← c
  retornar divisor
```

Estrutura de controle de fluxo

Fonte: Wikipédia

Exercício - Resolução

```
int mdc(int a, int b) {
  int r;
  while (a%b != 0) {
    r = a%b;
    a = b;
    b = r;
  }
return b;
}
```

Exercício - Funções

- **Crivo de Eratóstenes** é um método bastante prático para encontrar os primos de *2* até um *valor limite*, que pode ser feito a mão e é fácil de implementar.

  Para exemplificá-lo, vamos determinar a lista de números entre 2 e 30.
  - Inicialmente, determina-se o maior número a ser checado. Ele corresponde à raiz quadrada do valor limite, arredondado para baixo. No caso, a raiz de 30, arredondada para baixo, é 5.
  - Crie uma lista de todos os números inteiros de 2 até o valor limite: 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30.
  - Encontre o primeiro número primo da lista. No caso, este número é 2.
  - Remova da lista todos os múltiplos do número primo encontrado. No nosso exemplo, a lista fica: 2 3 5 7 9 11 13 15 17 19 21 23 25 27 29.
  - Verifique se o próximo número da lista é primo. Se for, repita o procedimento. No caso, o próximo número da lista é 3, que é primo. Removendo seus múltiplos, a lista fica: 2 3 5 7 11 13 17 19 23 25 29. O próximo número, 5, também é primo; a lista fica: 2 3 5 7 11 13 17 19 23 29. 5 é o último número a ser verificado, conforme determinado inicialmente. Assim, a lista encontrada contém somente números primos.

Exercício (Resolução)

```
#include <stdafx.h>
#include <stdlib.h>
#include <math.h> // necessário para raiz

#define NMAX 100000 // valor máximo 100.000

int main() {
    int i, j, vetor[NMAX];  float valorMaximo, raiz;
    // Primeiro passo
    scanf("%f", &valorMaximo);
    // Segundo passo
    raiz=sqrt(valorMaximo);
    // Terceiro passo
    for (i=2; i<=valorMaximo; i++) {
        vetor[i]=i;   }
    // Quarto passo
    for (i=2; i<=raiz; i++) {
        for (j=i+i; j<=valorMaximo; j+=i) {
            vetor[j]=0; // removendo os não primos }    }
    // Quinto passo
    for (i=0; i<=valorMaximo; i++) {
        if (vetor[i] > 0) {
            printf("%d eh primo!\n", vetor[i]); }  }
    system("pause");   return 0;  }
```

Exercício - Funções

Pseudocódigo:

```
função Converte(entradas: Far)
início
   Converte ← ((Far - 32) * 5) / 9
fim

programa teste
início
   imprima 'Temperatura em Fahrenheit: '
   leia Far
   Cel  ← Converte(Far)
   imprima 'Celsius: ', Cel
fim
```

Crie um programa que inicia uma variável com 1 e chama por 3 vezes a função soma1, que deve sempre retornar a função principal, o valor anterior mais hum. Mantendo o valor da variável entre as chamadas da função.

As estruturas de controle de fluxo são fundamentais para qualquer linguagem de programação. Sem elas só haveria uma maneira do programa ser executado: de cima para baixo comando por comando. Não haveria condições, repetições ou saltos. A linguagem C possui diversos comandos de controle de fluxo. É possível resolver todos os problemas sem utilizar todas elas, mas devemos nos lembrar que a elegância e facilidade de entendimento de um programa dependem do uso correto das estruturas no local certo.

Estrutura condicional simples

- Comando **if**

```
if (condição)
    comando;

if (condição) {
    comando1;
    comando2;
    comando3;
}
```

Estrutura condicional composta

- Comando **if...else**

```
if (condição)
    comando;
else
    comando;

if (condição) {
    comando1;
    comando2;
} else {
    comando3;
    comando4;
}
```

Executa o comando se a condição for qualquer coisa diferente de zero!

```
if (peso= =peso_ideal)
    printf (“Vc está em forma!”);
else
    printf (“Necessário fazer dieta!”);
```

Estrutura condicional aninhada

Ifs aninhados

```
/* programa do número mágico */
#include <stdio.h>
#include <stdlib.h>
void main() {
    int magico;    /* número mágico */
    int palpite;    /* número do palpite do usuário */
    magico = rand() % 10; /* gera inteiro aleatório */
    printf("Adivinhe o numero magico: 0 .. 9");
    scanf("%d", &palpite);
    if( palpite == magico) {
       printf(" ** Certo **");
       printf(" %d eh o numero magico\n", magico);
    }
    else {
      printf("Errado, ");
      if (palpite > magico)
         printf(" numero muito alto\n")
      else
         printf("muito baixo\n");
    }
    system("pause");
}
```

O operador ?(ternário)

```
#include <stdafx.h>
#include <stdlib.h>

void main() {
   int x = 10;
   int y = (x > 9) ? 100 : 200;
   printf("O valor de y eh: %d\n",y);
   system("pause");
}
```

Expressão a ser avaliada

Avaliação Verdadeira

Avaliação Falsa

O operador ?(ternário)

```
/* programa do número mágico */
#include <stdafx.h>
#include <stdlib.h>
void main() {
    int magico;             /* número mágico */
    int palpite; /* número do palpite do usuário */
    magico = rand(); /* gera inteiro aleatório */
    printf("Adivinhe o numero magico: ");
    scanf("%d", &palpite);
    if( palpite == magico) {
      printf(" ** Certo **");
      printf(" %d eh o numero magico\n", magico);
    }
    else {
    printf("Errado, ");
    palpite > magico ? printf(" numero muito alto\n"): printf("muito baixo\n");
    }
    system("pause");    }
```

Seleção múltipla

```
switch (expressão) {
      case item_1:
        instrução_1;
        break;
      case item_2:
        {
            instrução_2a;
            instrução_2b;
            break;
        }
      ...
      case item_n:
        instrução_n;
        break;
      default:
        instrução;
}
```

O uso de chaves na estrutura de controle *case*, não é obrigatório mesmo para blocos de códigos.

Switch

– Efeito: Avalie a expressão do *switch*.
– Vá para: o *case* que é igual a expressão avaliada ou *default* se não existe nenhum *case*;
– termine o *switch* se não existe *default*.
– termine o *switch* quando encontrar um *break.*

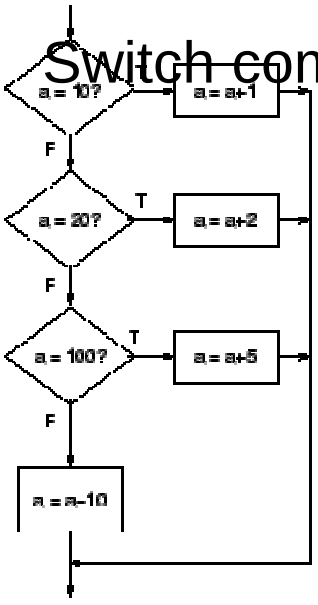
Switch

- Importante:
- O parâmetro do *switch* deve ser *int* ou *char.*
- O valor após o *case* deve ser uma **CONSTANTE.**
- Instrução *break* quando executada, faz com que aconteça a saída imediata daquela estrutura (no caso, o comando *switch*) A execução do programa continua com a primeira instrução depois da estrutura.
- A falta do *break* faz o controle passar ao próximo *case*, o que na maioria das vezes, não é o desejado.

Switch sem *break*

```
char tecla;
tecla = getchar();
switch (tecla) {
    case 'A':   case ‘a':
    case ‘E':   case ‘e':
    case ‘I':    case ‘i':
    case ‘O':   case ‘o':
    case ‘U':   case ‘u':
       printf(“Eh uma vogal\n”);
       break;
     case ‘ ’:  case ‘\n’
       printf(“Eh um espaco ou uma nova linha\n”);
       break;
      default:
       printf(“Não sei o que eh isso\n”);
}
```

Switch com o break

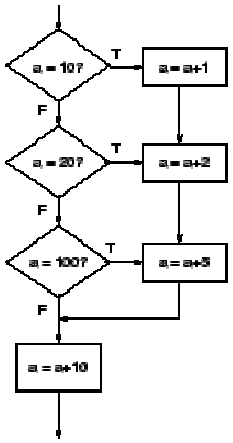


(a)

```
switch (a) {
  case 10:  a = a+1;
            break;
  case 20:  a = a+2;
            break;
  case 100: a = a+5;
            break;
  default:  a = a+10;
}
```

(b)

Switch omitindo o break



(a)

```
switch (a) {
  case 10:  a = a+1;
  case 20:  a = a+2;
  case 100: a = a+5;
  default:  a = a+10;
}
```

(b)

Um exemplo

```
switch(dia % 7){
  case 6:  /*** sábado ***/
           tarifa = 8;
           break;
  case 0:  /*** domingo ***/
           tarifa = 5;
           break;
  default: /*** demais dias ***/
           tarifa = 10;
           break;
}
```

Outro exemplo - switch

```
switch(dia % 7){
  case 1:
  case 2:
  case 3:
  case 4:
  case 5:  tarifa = 10;  break;
  case 6:  tarifa =  8;  break;
  case 0:  tarifa =  5;  break;
}
```

Menu de opções com *Switch*

```
main(){
    char x;
    printf("1. inclusao\n");
    printf("2. alteracao\n");
    printf("3. exclusao\n");
    printf("Digite sua opcao:\n");
    x=getchar();

    switch(x){
          case '1': inclusao() ; break;
          case '2': alteracao() ; break;
          case '3': exclusao() ;break;
          default:  invalida();
     }
}
```

Mais um switch

```
#include "stdio.h"
#include "stdlib.h"

void main()
{
    char opcao='0';
    do {
        opcao = getchar();
        switch (opcao) {
            int a;
            case '1':
                int b;
                printf("Opcao 1\n");        break;
            case '2':
                int c;
                printf("Opcao 2\n");        break;
            case '9':
                int d;
                printf("sair do switch\n"); break;
            case '\n':
                int e;
                printf("Opcao nova linha\n");       break;
            default:
                int f;
                printf("Opcao invalida!\n");
        }
        system("pause");
    } while (opcao != '9');
}
```

Pega o <enter> em getchar()

Variável declarada dentro do *case*

Não é necessário criar bloco de códigos {}

Repetição contada - Comando *For*

for (iniciação ; condição ; incremento)
comando ou bloco;

- Iniciação: iniciar uma ou mais variáveis de controle do laço.
- Condição: Uma expressão condicional (resulta verdadeiro ou falso). O laço é executado enquanto a condição for verdadeira.
- Incremento: Define como a(s) variável(eis) de controle vai(ão) variar.

Iteração (Laços) - Comando *For*

for (iniciação;condição;incremento)
{
declaração;
}

O **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 declaração, faz o incremento e volta a testar a condição. Ele fica repetindo estas operações até que a condição seja falsa.

Podemos definir como sendo um laço de execução com um número pré definido de vezes.

```
#include <stdio.h>
main () {
      int count;
      for (count=1; count<=100; count++)
      printf ("%d ",count);
}
```

Esse exemplo imprime na tela os valores de 1 a 100.

Exemplo - Tabuada

```
inteiro n, cont, r;

escrever("Informe o nro do qual deseja a tabuada : ");
ler(n);

para (cont ← 1 até 10) faça
inicio
    r ← cont*n;
    escrever (cont, " *" , n, "= ", r);
fim
```

Exemplo - Tabuada

```
void main() {
  int n, cont;

  printf("\nInforme o nro do qual deseja a tabuada : ");
  scanf("%i",&n);

  for (cont = 1; cont <=10; cont++)
     printf("\n%i * %i = %i",cont, n, (cont*n));

  getch();
  getch();
}
```

Outros exemplos

- for (y=1 ; y!=x; y++) printf(“%d”, y);
  printf(“Último y: %d\n”, y);
- for (i=1, j=10; i<j ; i++, j--)
  printf(“I: %d J: %d\n”, i,j);
- for (x=0; x!=123;) scanf(“%d”, &x);
- for (tempo=0; tempo<valor; tempo++) ;
- for(;;)
  printf(“Forever running...”);

Estruturas de Iteração *while*

while (condição) {
declaração;
}

O ***while*** testa uma condição. Se esta for verdadeira a declaração é executada e faz-se o teste novamente, e assim por diante. Podemos definir como sendo um laço com teste no início. Isso indica que a declaração pode não ser executada.

```
main () {
  char Ch;
  Ch='\0';
  while (Ch!='s') {
    Ch = getch();
  }
}
```

Em condições que avaliam lógica (V ou F), use sempre a constante do lado esquerdo do sinal de igualdade:
Ex: while (1 == x)
ao invés de
while (x == 1)

Estruturas de Iteração *do...while*

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

Apesar de não ser necessário para uma única declaração de comando, sempre utilize as chaves na estrutura do...while.

O **do-while** executa a declaração, testa a condição e, se esta for verdadeira, volta para a declaração.Podemos definir como sendo um laço com teste no final. Isso indica que a declaração será executada pelo menos uma vez.

```
#include <stdio.h>
main ()
{   int i;
    do
    {   printf ("Escolha a fruta pelo nº:");
        printf ("[1]...Mamão\n");
        printf ("[2]...Abacaxi\n");
        printf ("[3]...Laranja\n\n");
        scanf("%d", &i);
    } while ((i<1)||(i>3));
```

Esse exemplo mostra como fazer um menu.

```
    switch (i)
    {   case 1:  printf ("Você escolheu Mamão");
                 break;
        case 2:  printf ("Você escolheu Abacaxi");
                 break;
        case 3: printf ("Você escolheu Laranja");
                 break;
    }
}
```

Para melhorar a performance da aplicação, utilize a expressão lógica c/ maior possibilidade de ser falsa do lado esquerdo da condição.

Exercícios

Exercícios residencial:

Chico tem 1,50 metro e cresce 20 centímetros por ano, enquanto Zé tem 1,10 metro e cresce 30 centímetros por ano. Construa um programa que calcule e imprima quantos anos serão necessários para que Zé seja maior que Chico.

Exercícios de fixação

Explique porque está errado fazer:

if (num=10)

... O que irá acontecer?

Exercícios de fixação

- Sendo h= 1 + ½ + 1/3 + ¼ + ... + 1/N, prepare um algoritmo para calcular o número h, sendo o número N fornecido pelo usuário.
- Elabore um algoritmo que calcule N! (fatorial de N), sendo que o valor inteiro de N é fornecido pelo usuário. Sabendo que:

  N! = 1 x 2 x 3 x ... x (N-1) x N;

  0! = 1, por definição.

Exercícios de fixação

- Faça um programa que apresente na tela a tabela de conversão de graus Celsius para Fahrenheit, de -100 C a 100 C. Use um incremento de 10 C.

  OBS: Fahrenheit = (9/5)*(Celsius) + 32
- Escreva um programa que coloque os números de 1 a 100 na tela na ordem inversa (começando em 100 e terminando em 1).
- Calcular e listar todos os múltiplos positivos do número 7 menores ou iguais a 100.

Exercícios de fixação

Escreva um programa que peça ao usuário que digite três números inteiros, correspondentes a dia , mês e ano. Teste os números recebidos, e em caso de haver algum inválido, repita a leitura até conseguir valores que estejam na faixa correta (dias entre 1 e 31, mês entre 1 e 12 e ano entre 1900 e 2100). Verifique se o mês e o número de dias batem. Se estiver tudo certo imprima o número que aquele dia corresponde no ano. Comente seu programa.

Exercícios de fixação

- Faça um programa que leia números digitados pelo usuário até ele digitar -1 (utilizando um laço while). No final, calcule a média dos números fornecidos.
- Perguntar ao usuário quantos números deseja somar. Em seguida, ler estes N números e apresentar o valor da soma. (Fazer 3 versões deste programa: usando FOR, usando WHILE e usando DO...WHILE).

Exercícios de fixação

- Encontre o erro:

```
x = 1;
while (x<=10);
    x++; }
```

- Encontre o erro:

```
for (x=0.1; x!=1.0; x+=0.1)
    printf("%f\n", x);
```

- Encontre o erro:

O código a seguir deve imprimir valores de 1 a 10

```
int n =1;
while (n < 10)  printf("%d ", n++);
printf("\n");
system("pause");
```

Exercícios de fixação

- Escrever instruções *for* que imprimam as seguintes seqüências:
  - 1,2,3,4,5,6,7
  - 3,8,13,18,23
  - 1,1,2,3,5,8,13,21 (lembra de Fibonacci?!!)
  - 2,4,8,16,32,64,128
  - 7,7,7,7,...,7,...

Estrutura de Repetição

Estrutura de Repetição

- Repetir um trecho de um programa por determinado número de vezes;
- Laços contados ➔ o número de vezes que o trecho irá se repetir é conhecido;
  - Contador ➔ seu próprio valor é adicionado ou subtraído de um valor *constante*;
- Laços condicionais ➔ quando existe uma condição para que um trecho se repita;
  - O valor de uma variável informada pelo usuário é usado em uma condição que finaliza a repetição.

2

Estrutura de Repetição

- Enquanto – **while ...**
  - Sintaxe:

```
while (condição)
{
    <seqüência de comandos separados por ;>
}
```

- Exemplo do *while* contado: Programa para ler duas notas de 30 alunos, calcular e imprimir a média de cada um deles.

3

Estrutura de Repetição

```
#include <stdio.h>
int main()
{
   float n1, n2, media;
   int cont = 0;
   while (cont < 30)
   {
      printf("Digite duas notas: ");
      scanf("%f %f", &n1, &n2);
      media = (n1+n2)/2;
      printf("Media = %f", media);
      cont = cont + 1;
   }
}
```

4

Estrutura de Repetição

- Exemplo do *while* condicional: Programa para ler um número enquanto o número digitado não for maior que zero.

5

Estrutura de Repetição

```
#include <stdio.h>
int main()
{
   int n;
   printf("Digite um número maior que zero: ");
   scanf("%d", &n);
   while (n <= 0)
   {
      printf("Numero invalido!!! \n Digite um número maior que zero: ");
      scanf("%d", &n);
   }
   printf("Parabéns! Você sabe o que é um número maior que zero");
}
```

6

Estrutura de Repetição

- Faça...enquanto – do ... while ...
  - Sintaxe:

```
do
{
    <seqüência de comandos separados por ;>
}while (condição);
```

- Exemplo do *do...while* contado: Programa para ler duas notas de 30 alunos, calcular e imprimir a média de cada um deles.

7

Estrutura de Repetição

```
#include <stdio.h>
int main()
{
   float n1, n2, media;
   int cont = 0;
   do
   {
      printf("Digite duas notas: ");
      scanf("%f %f", &n1, &n2);
      media = (n1+n2)/2;
      printf("Media = %f", media);
      cont = cont + 1;
   }while (cont < 30);
}
```

8

Estrutura de Repetição

- Exemplo do *do...while* condicional: Programa para ler um número até que o número digitado seja maior que zero.

9

Estrutura de Repetição

```
#include <stdio.h>
int main()
{
    int n;
    do
    {
       printf("Digite um número maior que zero: ");
       scanf("%d", &n);
    }while (n <= 0);
    printf("Parabéns! Você sabe o que é um número maior que zero");
}
```

10

Estrutura de Repetição

- Para...faça – for ...
  - Sintaxe:

```
for (inicialização;condição;incremento)
{
    <seqüência de comandos separados por ;>
}
```

- Exemplo do *for* contado: Programa para ler duas notas de 30 alunos, calcular e imprimir a média de cada um deles.

11

Estrutura de Repetição

```
#include <stdio.h>
int main()
{
    float n1, n2, media;
    int cont;
    for (cont=0;cont<30;cont++)
    {
        printf("Digite duas notas: ");
        scanf("%f %f", &n1, &n2);
        media = (n1+n2)/2;
        printf("Media = %f", media);
    }
}
```

12

Estrutura de Repetição

- Exemplo do *for* condicional: Programa para ler um número até que o número digitado seja maior que zero.

13

Estrutura de Repetição

```
#include <stdio.h>
int main()
{
    float n1, n2, media;
    int n;
    for (;n<=0;)
    {
       printf("Digite um número maior que zero: ");
       scanf("%d", &n);
    }
    printf("Parabéns! Você sabe o que é um número maior que zero");
}
```

14

Exercícios

Exercício 1

- Escreva um programa que armazena uma determinada letra no código e o usuário têm que tentar adivinhar qual letra é esta. O programa deve imprimir a cada erro uma mensagem e no acerto uma mensagem correspondente e o número de tentativas.

Exercício 2

- Imprimir os números pares entre 1 e 18.