Introducao a Programacao: Estruturas de repetição

periclesmiranda@gmail.com

Slides fornecidos pelo prof. Gurvan Huiban

Plano de aula

Estruturas de repetição

- Repetir a execução de um conjunto de instruções por um número de vezes:
  - Fixo;
  - Variável.
- Na linguagem C: `for`, `while`, `do-while`.

Exemplos

- Somar os *n* primeiros números ímpares;
- Repetir um menu até o usuário digitar `x`.

Programa: Mostrar a soma dos *n* primeiros inteiros, para *n* de 1 até 10

```
main()
{
  int soma = 0;
  soma = soma + 1;
  printf("Valor da soma ate 1: %d\n", soma);
  soma = soma + 2;
  printf("Valor da soma ate 2: %d\n", soma);
  soma = soma + 3;
  printf("Valor da soma ate 3: %d\n", soma);
  soma = soma + 4;
  printf("Valor da soma ate 4: %d\n", soma);
  soma = soma + 5;
  printf("Valor da soma ate 5: %d\n", soma);
  soma = soma + 6;
  printf("Valor da soma ate 6: %d\n", soma);
  soma = soma + 7;
  printf("Valor da soma ate 7: %d\n", soma);
  soma = soma + 8;
  printf("Valor da soma ate 8: %d\n", soma);
  soma = soma + 9;
  printf("Valor da soma ate 9: %d\n", soma);
  soma = soma + 10;
  printf("Valor da soma ate 10: %d\n", soma);
}
```

Programa: Mostrar a soma dos *n* primeiros inteiros, para *n* de 1 até 10

```
int main(void)
{
   int soma = 0;
   int i;

   for (i = 1; i <= 10; i = i+1)
   {
      soma = soma + i;
      printf("Valor da soma ate %d: %d\n", i, soma);
   }
}
```

Sintaxe

```
for (inicializacoes; condicoes; incrementos)
{
  sequencia de comandos
}
```

- Inicializações: valores iniciais das variáveis;
- Condições: condições para continuar a execução do laço (expressão lógica);
- Incrementos: incrementação das variáveis;
- Sequência de comandos: comandos a serem repetidos.

Repetição com a estrutura `for`

- Para a variável `x` de *i* até *j*, fazer alguma coisa;
- Número de repetições pode ser antecipado no momento da execução.

Passos de execução

```
for (inicializacoes; condicoes; incrementos)
{
  sequencia de comandos
}
```

1. `Inicializações;`
2. Verificação das `condições;`
3. Se verdadeiras: `sequência de comandos;`
4. `Incrementos;`
5. Verificação das `condições;`
6. Se verdadeiras: `sequência de comandos;`
7. ...

Exercícios

- Entrar com um inteiro *n*, e imprimir na tela os números ímpares menores ou iguais a *n*;

Exercícios

- Entrar com um inteiro *n*, e imprimir na tela os números ímpares menores ou iguais a *n*;
- Faça um programa que imprima em ordem decrescente todos os valores inteiros maiores que zero a partir de um número fornecido pelo usuário;

Observações

- Não tem `;` no fim da linha do `for`;
- Se tiver uma instrução só, as chaves `{` e `}` são opcionais;
- É possível usar caracteres no lugar de inteiros:

```
for(c='a'; c <= 'z'; c++)
  printf("O valor ASCII de %c e:%d\n",c,c);
```

- Qualquer uma das três expressões de laço pode conter várias instruções separadas por vírgulas:

```
for(x=0, y=0; x+y<100; x++, y++)
   printf("%d+%d=%d\n",x,y,x+y);
```

Observações

- Qualquer uma das três expressões de laço pode chamar funções:

```
for(c=getchar(); c != 'x'; c=getchar())
  printf("Digitou o caracter %c\n",c);
```

- É possível omitir qualquer uma das três expressões desde que os ponto e vírgulas permaneçam:

```
for(; (c=getchar()) != 'x'; )
  printf("Digitou o caracter %c\n",c);
```

- O corpo do laço pode ser vazio:

```
for(; (c=getchar()) != 'x'; printf("%c\n",c));
```

Ou seja...

Com a estrutura `for`, podemos descrever laços com código muito ~~(complicado)~~ conciso.

Recomendações

- Não sobrecarregar os três elementos `inicializacoes`, `condicoes` e `incrementos`;
- Usar a estrutura `for` quando a quantidade de laços é previsível no momento da execução;
- Nos outros casos, usar uma estrutura `while` ou `do-while`.

Sintaxe

```
while (expressao_de_teste)
{
  sequencia de comandos
}
```

Repetição com a estrutura while

- Usa apenas uma expressão de teste;
- Se tiver uma instrução só, as chaves { } são opcionais;
- Mais apropriado que a estrutura for em situações em que o laço pode ser terminado inesperadamente, em consequência das operações do corpo do laço;
- O corpo do laço será executado sempre que a expressão for verdadeira.

Passos de execução

```
while (expressao_de_teste)
{
  sequencia de comandos
}
```

1. Verificação da `expressão_de_teste`;
2. Se verdadeira: `sequência de comandos`;
3. Verificação da `expressão_de_teste`;
4. Se verdadeira: `sequência de comandos`;
5. ...

Programa: Imprimir a tecla digitada até o usuário digitar x:

```
printf("Entre com um caracter (x para sair): ");
c = getchar();
while(c != 'x')
{
  printf("Digitou o caracter %c\n",c);
  printf("Entre com outro caracter (x para sair): ");
  c = getchar();
}
```

Exercício: simular um `for`

Faça um programa usando a estrutura `while` que imprima em ordem decrescente todos os valores inteiros maiores que zero a partir de um número fornecido pelo usuário.

Exercício: simular um `for`

Faça um programa usando a estrutura `while` que imprima em ordem decrescente todos os valores inteiros maiores que zero a partir de um número fornecido pelo usuário.

Exercício: Algarismos

Escreva um programa para determinar o número de algarismos de um número inteiro positivo dado.

Sintaxe

```
do
{
   sequencia de comandos
}
while (expressao de teste)
```

Repetição com a estrutura do-while

- Como a estrutura `while-do`, é usado quando a quantidade de repetições não é previsível no momento da execução;
- Se tiver uma instrução só, as chaves `{ }` são opcionais;
- Diferente do `while-do`, a expressão de teste fica no fim da estrutura;
- Repetição da `sequência de comandos` até a `expressão_de_teste` ser false;
- Consequentemente, o laço é sempre executado pelo menos uma vez.

Passos de execução

```
do
{
  sequencia de comandos
}
while (expressao de teste)
```

1. Sequência de comandos;
2. Verificação da expressão_de_teste;
3. Se verdadeira: sequência de comandos;
4. Verificação da expressão_de_teste;
5. ...

Exercício: Jogo de sorte

Escrever um programa em C que peça ao jogador para adivinhar o *número da sorte* (entre 0 e 100) gerado aleatoriamente pelo programa. O jogador vai entrando com números, e o programa vai informando se o número do jogador é maior ou menor que o número da sorte.
Quando o jogador acertar o número, o programa deve imprimir 'ACERTOU' e informar o número de tentativas do jogador.
Para gerar um número entre 0 e 100, use o comando:

```
num = rand()*100.0/RAND_MAX;
```

Estruturas aninhadas

Obviamente, estas estruturas podem ser aninhadas...
Por exemplo:

```
do
   sequencia de comandos (1)
   for (i=n1; i <= n2; i++)
   {
      sequencia de comandos (1.1)
   }
   sequencia de comandos (2)
while (expressao_de_teste);
```

```
do
  sequencia de comandos (1)
  for (i=n1; i <= n2; i++)
  {
    sequencia de comandos (1.1)
  }
  sequencia de comandos (2)
while (expressao_de_teste);
```

1. `Sequência de comandos (1);`
2. Execução da estrutura `for`:
   1. `i = n1`
   2. Verificação se `i` $\leq$ `n2`;
   3. Se verdadeira: `sequência de comandos (1.1);`
   4. `...`
3. `Sequência de comandos (2);`
4. Verificação da `expressão_de_teste`;
5. Se verdadeira: `sequência de comandos(1);`
6. Execução da estrutura `for`:
7. `...`

Exercício: Estrelas

Usando apenas os comandos `printf(".")`, `printf("*")` e `printf("\n")` e usando laços aninhados, faça o seguinte aparecer na tela.

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

Exercício: Tabuadas

Imprimir na tela as tabuadas de 1 até 8 de uma das formas seguintes:

```
Tabuada de 1
1 x  1 =   1
1 x  2 =   2
1 x  3 =   3
1 x  4 =   4
1 x  5 =   5

Tabuada de 2
2 x  1 =   2
2 x  2 =   4
2 x  3 =   6
```

```
Tabuada de 1   Tabuada de 2   Tabuada de 3   Tabuada de 4
1 x  1 =   1   2 x  1 =   2   3 x  1 =   3   4 x  1 =   4
1 x  2 =   2   2 x  2 =   4   3 x  2 =   6   4 x  2 =   8
1 x  3 =   3   2 x  3 =   6   3 x  3 =   9   4 x  3 =  12
1 x  4 =   4   2 x  4 =   8   3 x  4 =  12   4 x  4 =  16
1 x  5 =   5   2 x  5 =  10   3 x  5 =  15   4 x  5 =  20

Tabuada de 5   Tabuada de 6   Tabuada de 7   Tabuada de 8
5 x  1 =   5   6 x  1 =   6   7 x  1 =   7   8 x  1 =   8
5 x  2 =  10   6 x  2 =  12   7 x  2 =  14   8 x  2 =  16
5 x  3 =  15   6 x  3 =  18   7 x  3 =  21   8 x  3 =  24
```

O comando `break`

O comando `break` causa a saída imediata do laço.

O comando `continue`

Pula o código que estiver abaixo e força a próxima iteração do laço.

Observações

- Podem ser usado no corpo de qualquer estrutura (`for`, `while`, `do-while`);
- Se o `break` ou o `continue` estiver em laços aninhados, afetará somente o laço mais interno onde ele está.

Evitar o uso dos comandos `break` e `continue`

Deve ser evitado, pois pode causar dificuldade de leitura e confusão na manutenção o programa.

Exemplo

```
while (1)
{
   printf("Digite um numero maior que zero: ");
   scanf("%d",&num);
   if(num<0)
   {
      printf("numero errado\n");
      continue;
   }
   printf("Numero correto\n");
}
```