Programação de Computadores I

Prof. Rafael Nunes

A Linguagem C

Parte 5

Comandos de *controle de fluxo*

Sumário

- O comando if
  - O if-else
  - O if-else-if
- O comando switch
- O comando for
- O comando while
- O comando do-while
- O comando break
- O comando continue
- O comando goto

O Comando *if*

O Comando if

- O comando if representa uma ***tomada de decisão*** do tipo "SE *isto* ENTÃO aquilo".

- A sua forma geral é:

  if (*condição*) declaração;

O Comando if

- A condição do ***comando if*** é uma expressão que ***será avaliada***.
  - Se o resultado for ***zero*** a declaração ***não será executada***.
  - Se o resultado for ***qualquer coisa diferente de zero*** a declaração ***será executada***.

Vamos relembrar

O Comando if

```
teste = 1;
1.  if (teste == 0)
2.      printf("ALO!")   //NÃO IRÁ EXECUTAR

//-----------------------------------------

1.  if (teste != 0)
2.      printf("ALO!")   //IRÁ EXECUTAR
```

Exemplo

O Comando if

```
1. #include <stdio.h>
2. int main ()
3. {
4.        setbuf(stdout,NULL);
5.         int num;
6.        printf ("Digite um numero: ");
7.        scanf ("%d",&num);
8.        if (num>10)
9.             printf ("\n\nO numero e maior que 10");
10.       if (num==10)
11.       {
12.                       printf ("\n\nVoce acertou!\n");
13.                       printf ("O numero e igual a 10.");
14.        }
15.       if (num<10)
16.            printf ("\n\nO numero e menor que 10");
17.       return (0);
18.}
```

O Comando if

- No programa acima ***a expressão num>10*** é avaliada e retorna um valor diferente de zero, ***se verdadeira***, e zero, ***se falsa***.

- Repare que quando queremos testar igualdades usamos o operador **==** e não **=**
  - Isto é porque o operador = representa apenas uma ***atribuição***.

O *if-else*

O Comando if-else

- Podemos pensar no comando else como sendo um ***complemento do comando*** if.
  - O ***comando if*** completo tem a seguinte forma geral:

if (*condição*) declaração_1;

else declaração_2;

O Comando if-else

- A *expressão da condição* será avaliada
- Se ela for ***diferente de zero*** (verdadeiro) a declaração 1 será executada.
  - if (*condição*!=0) executa *declaração_1*
- Se ***for zero*** (falso) a declaração 2 será executada, ou seja, o bloco do **else**
  - if (*condição*==0) executa *declaração_2*

O Comando if-else

- É importante nunca esquecer que, quando usamos a estrutura if-else, estamos garantindo que uma das duas declarações ***sempre será executada***

- Nunca serão executadas ***as duas*** ou ***nenhuma delas***

Exemplo

Exemplo: if-else

```
#include <stdio.h>
int main ()
{
    int num;
    printf ("Digite um numero: ");
    scanf ("%d",&num);
    if (num==10)
    {
        printf ("\n\nVoce acertou!\n");
        printf ("O numero e igual a 10.\n");
    }
    else
    {
        printf ("\n\nVoce errou!\n");
        printf ("O numero e diferente de 10.\n");
    }
    return(0);
}
```

Bloco Verdadeiro = if

Bloco falso = else

O *if-else-if*

O if-else-if

- A estrutura if-else-if é apenas uma extensão da estrutura if-else.
  - Sua forma geral pode ser escrita como sendo:

```
if (condição_1) declaração_1;
    else if (condição_2) declaração_2;
    else if (condição_3) declaração_3;
                . ..
    else if (condição_n) declaração_n;
     else declaração_default; //Opcional
```

O if-else-if

- Atenção!:

A última declaração (***default***) é a que será executada no caso de todas as condições falharem, ou seja, serem falsas (zero)

Ela ***é opcional !***

Exemplo

Exemplo: if-else-if

```
#include <stdio.h>
int main ()
{
    int num;
    printf ("Digite um numero: ");
    scanf ("%d",&num);
    if (num>10)
        printf ("\n\nO numero e maior que 10");
    else if (num==10)
    {
        printf ("\n\nVoce acertou!\n");
        printf ("O numero e igual a 10.");
    }
    else if (num<10)
        printf ("\n\nO numero e menor que 10");
    return(0);
}
```

A expressão condicional

A expressão condicional

- Quando o compilador avalia uma **condição**, ele quer um ***valor de retorno*** para poder tomar a decisão.
- Mas esta expressão não precisa ser uma **expressão** ***no sentido convencional***.
- Uma ***variável sozinha*** pode ser uma "expressão" e esta retorna o seu próprio valor.
- Vejamos alguns exemplos...

A expressão condicional

- Isto quer dizer que teremos as seguintes expressões:

```
int num;
if (num!=0) ....
if (num==0) ....
for (i = 0; string[i] != '\0'; i++)
```

equivalem a

```
int num;
if (num) ....
if (!num) ....
for (i = 0; string[i]; i++)
```

- Isto quer dizer que podemos simplificar algumas expressões simples.

if’s aninhados

if's aninhados

- O if aninhado é simplesmente ***um if dentro*** da declaração ***de um outro if*** externo.

- O único cuidado que devemos ter é o de saber exatamente a qual if um determinado else está ligado.

```
#include <stdio.h>
int main ()
{
    int num;
    printf ("Digite um numero: ");
    scanf ("%d",&num);
    if (num==10)
    {
        printf ("\n\nVoce acertou!\n");
        printf ("O numero e igual a 10.\n");
    }
    else
    {
        if (num>10)
        {
            printf ("O numero e maior que 10.");
        }
        else
        {
            printf ("O numero e menor que 10.");
        }
    }
    return(0);
}
```

O Comando *switch*

O Comando switch

- O comando if-else e o comando switch são os dois comandos de tomada de decisão.
- Sem dúvida alguma o mais importante dos dois é o if, mas o comando switch tem aplicações valiosas.
- O comando switch é próprio para se testar uma variável em relação a diversos valores pré-estabelecidos.

O Comando switch

- Forma geral:

switch(<expressão>){
case <const1>: < instruções1>
case <const2>: < instruções2>
...
case <constN>: < instruçõesN>
default: <instruções>
}

Exemplo

```
switch(dia){
    case 0: printf("Domingo\n"); break;
    case 1: printf("Segunda\n"); break;
    case 2: printf("Terça\n"); break;
    case 3: printf("Quarta\n"); break;
    case 4: printf("Quinta\n"); break;
    case 5: printf("Sexta\n"); break;
    case 6: printf("Sabado\n"); break;
    default: printf("Opção Inválida\n"); break;
}
```

O Comando switch

- Podemos fazer uma analogia entre o switch e a estrutura if-else-if apresentada anteriormente:
  - A diferença fundamental é que a estrutura switch não aceita expressões.
  - Somente aceita constantes.
- O switch testa a variável e executa a declaração cujo case corresponda ao valor atual da variável.
- A declaração default é opcional e será executada apenas se a variável, que está sendo testada, não for igual a nenhuma das constantes.

O Comando switch

- O comando break, faz com que o switch seja interrompido assim que uma das declarações seja executada.
- Mas ele não é essencial ao comando switch.
- Se após a execução da declaração não houver um break, o programa continuará executando.
- Isto pode ser útil em algumas situações, mas cuidado.

Exemplo

```c
#include <stdio.h>
int main ()
{
    int num;
    printf ("Digite um numero: ");
    scanf ("%d",&num);
    switch (num)
      {
          case 9:
                    printf ("\n\nO numero eh igual a 9.\n");
                    break;
          case 10:
                    printf ("\n\nO numero eh igual a 10.\n");
                    break;
          case 11:
                    printf ("\n\nO numero eh igual a 11.\n");
                    break;
          default:
                    printf ("\n\nO numero não é nem 9 nem 10 nem 11.\n");
      }
    return(0);
}
```

O Comando *for*

O Comando for

- O loop (laço) for é usado para ***repetir um comando***, ou bloco de comandos, ***diversas vezes***, de maneira que se possa ter um bom controle sobre o loop.

- Sua forma geral é:

for (*inicialização*;*condição*;*incremento*) declaração;

O Comando for

- A declaração no comando for também pode ser ***um bloco*** ({ }) e neste caso o ; é omitido.

- O melhor modo de se entender o loop for é ver de que ***maneira ele funciona*** "por dentro".

O Comando for

O ***loop for*** é equivalente a se fazer o seguinte:

1. *inicialização*;
2. ***if*** (*condição*)
3. {
4. declaração;
5. *incremento*;
6. ***"Volte para o comando if"***
7. }

O Comando for

- Podemos ver então que o for executa a *inicialização* incondicionalmente e testa a condição.
  - Se a *condição* for falsa ele não faz mais nada.
  - Se a *condição* for verdadeira ele executa a declaração (apenas uma vez), o *incremento* e volta a testar a condição.
- Ele fica repetindo estas operações até que a condição ***seja falsa***.

A seguir vemos um programa que coloca os *primeiros 100 números na tela*...

O Comando for

```
1.  #include <stdio.h>
2.  int main ()
3.  {
4.      int count;
5.      for (count=1;count<=100;count=count+1) printf ("%d ",count);
6.      return(0);
7.  }
```

O que faz o próximo exemplo...

```
1. #include <stdio.h>
2. int main ()
3. {
4.      char string[100];     /* String, ate' 99 caracteres */
5.      int i, cont;
6.      printf("\n\nDigite uma frase: ");
7.      gets(string); /* Le a string */
8.      printf("\n\nFrase digitada:\n%s", string);
9.      cont = 0;
10.     for (i=0; string[i] != '\0'; i++)
11.     {
12.         if ( string[i] == 'c' )   /* Se for a letra 'c' */
13.             cont = cont +1;   /* Incrementa o contador de caracteres */
14.     }
15.     printf("\nNumero de caracteres c = %d", cont);
16.}
```

O Comando for

- Note o teste que está sendo feito no for:
  - o caractere armazenado em string[i] é comparado com '\0' (caractere final da string).
- Caso o caractere seja diferente de '\0', a condição se torna verdadeira e o bloco do for é executado.
- Dentro do bloco existe um ***if*** que testa se o caractere é igual a 'c'.
  - Caso seja, o contador de caracteres c é incrementado.

O comando for

Loop infinito

Comando for - Loop Infinito

- O loop infinito tem a forma

  for (inicialização; ;incremento) declaração;

- Este loop chama-se loop infinito porque ***será executado para sempre*** (não existindo a condição, ela será sempre considerada verdadeira), a não ser que ele seja interrompido
  - Para interromper um loop como este usamos o comando break.
  - O comando break vai quebrar o loop infinito e o programa continuará sua execução normalmente.

Comando for - Loop Infinito

- Como exemplo vamos ver um programa que faz a leitura de uma tecla e sua impressão na tela, até que o usuário aperte uma tecla sinalizadora de final (um FLAG). O nosso FLAG será a letra 'X'.

```
#include <stdio.h>
#include <conio.h>
int main ()
{
    int Count;
    char ch;
    for (Count=1;   ;Count++)
      {
          ch = getch();
          if (ch == 'X') break;
          printf("\nLetra: %c",ch);
      }
    return(0);
}
```

O comando for

Loop sem conteúdo

Comando for - Loop sem conteúdo

- Loop sem conteúdo é aquele no qual se omite a declaração.
- Sua forma geral é (atenção ao ponto e vírgula no final!):

```
for (inicialização;condição;incremento) ;
```

Comando for - Loop sem conteúdo

- Uma das aplicações desta estrutura é gerar tempos de espera. O programa faz isto.

```
#include <stdio.h>
int main ()
{
    long int i;
    printf("\a");                 /* Imprime o caractere de alerta (um beep) */
    for (i=0; i<10000000; i++);  /* Espera 10.000.000 de iterações */
    printf("\a");                /* Imprime outro caractere de alerta */
    return(0);
}
```

O comando while

O comando while

- O comando while tem a seguinte forma geral:

- while (condição) declaração;

O comando while

- Assim como fizemos para o comando for, vamos tentar mostrar como o while funciona fazendo uma analogia.
- Então o while seria equivalente a:

1. ***if*** (*condição*)
2. {
3. declaração;
4. ***"Volte para o comando if"***
5. }

O comando while

- Podemos ver que a estrutura while testa uma condição.
- Se esta for verdadeira a declaração é executada e faz-se o teste novamente, e assim por diante.
- Assim como no caso do for, podemos fazer um loop infinito.
  - Para tanto basta colocar uma expressão eternamente verdadeira na condição.
- Pode-se também omitir a declaração e fazer um loop sem conteúdo

Exemplo

O comando while (windows)

- O programa abaixo espera o usuário digitar a tecla 'q' e só depois finaliza:

```
#include <stdio.h>
#include <conio.h>
int main ()
{
    char Ch;
    Ch='\0';
    while (Ch!='q')
    {
        Ch = getch();
    }
    return(0);
}
```

O comando while (eclipse)

- O programa abaixo espera o usuário digitar a tecla 'q' e só depois finaliza:

```
#include <stdio.h>
int main ()
{
    setbuf(stdout, NULL);
    char Ch;
    Ch='\0';
    while (Ch!='q')
    {
       Ch = getchar();
    }
    return(0);
}
```

O comando do-while

O comando do-while

- Forma geral:

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

- Mesmo que a declaração seja apenas um comando é uma boa prática deixar as chaves.
- O ponto-e- vírgula final é obrigatório.

O comando do-while

- Vemos pela análise do bloco anterior que a estrutura do-while executa a declaração, testa a condição e, se esta for verdadeira, volta para a declaração.

- A grande novidade no comando do-while é que ele, ao contrário do for e do while, garante que a declaração será executada pelo menos uma vez.

Um dos usos da estrutura do-while é em menus, nos quais você quer garantir que o valor digitado pelo usuário seja válido

Exemplo

```
int main ()
{
    int  i;
    do
     {
          printf ("\n\nEscolha a fruta pelo numero:\n\n");
          printf ("\t(1)...Mamão\n");
          printf ("\t(2)...Abacaxi\n");
          printf ("\t(3)...Laranja\n\n");
          scanf("%d", &i);
     } while ((i<1)||(i>3));

    switch (i)
     {
          case 1:
                    printf ("\t\tVoce escolheu Mamão.\n");
          break;
          case 2:
                    printf ("\t\tVoce escolheu Abacaxi.\n");
          break;
          case 3:
                    printf ("\t\tVoce escolheu Laranja.\n");
          break;
     }
    return(0);
}
```

O comando break

O comando break

- Nós já vimos dois usos para o comando break:
  - interrompendo os comandos switch e for.
- Na verdade, estes são os dois usos do comando break:
  - ele pode quebrar a execução de um comando (como no caso do switch) ou ...
  - interromper a execução de qualquer loop (como no caso do for, do while ou do do while).
- O break faz com que a execução do programa continue na primeira linha seguinte ao loop ou bloco que está sendo interrompido.

O comando continue

O comando continue

- O comando continue pode ser visto como sendo o oposto do break.
  - Ele só funciona dentro de um loop.
- Quando o comando continue é encontrado, o loop pula para a próxima iteração, sem o abandono do loop
- Ao contrário do que acontecia no comando break.

Exemplo

```
int main()
{
    int opcao;
    while (opcao != 5)
    {
                printf("\n\n Escolha uma opcao entre 1 e 5: ");
                scanf("%d", &opcao);
                /* Opcao invalida: volta ao inicio do loop */
                if ((opcao > 5)||(opcao <1)) continue;
                switch (opcao)
                {
                   case 1:  printf("\n --> Primeira opcao..");  break;
                   case 2:  printf("\n --> Segunda opcao.."); break;
                   case 3:  printf("\n --> Terceira opcao..");  break;
                   case 4:  printf("\n --> Quarta opcao..");    break;
                   case 5:  printf("\n --> Abandonando..");   break;
                }
    }
    return(0);
}
```

O comando goto

O comando goto

- Pertence a uma classe à parte: a dos comandos de salto incondicional.
  - O goto realiza um salto para um local especificado.
  - Este local é determinado por um rótulo.
- Um rótulo, na linguagem C, é uma marca no programa.
  - Você dá o nome que quiser a esta marca.
- Podemos tentar escrever uma forma geral:

```
nome_do_rótulo:
....

goto nome_do_rótulo;
....
```

O comando goto

- O comando **goto** deve ser utilizado com parcimônia, pois o abuso no seu uso tende a tornar o código confuso.
- O **goto** não é um comando *necessário*, podendo sempre ser substituído por outras estruturas de controle.
- Puristas da programação estruturada recomendam que o **goto** nunca seja usado.

Porém, em algumas *situações muito específicas* o comando **goto** pode tornar um código mais fácil de se entender ***se*** ele for bem empregado

O comando goto

- Um caso em que ele pode ser útil é quando temos vários loops e **if**s aninhados e se queira, por algum motivo, sair destes loops e **if**s todos de uma vez.
- Neste caso um **goto** resolve o problema mais elegantemente que vários **break**s, sem contar que os **break**s exigiriam muito mais testes.
- Ou seja, neste caso o **goto** é mais elegante e mais rápido. ***Mas não abuse!!!***

Exemplo

```
int main()
{
    int opcao;
    while (opcao != 5)
        {
        REFAZ: printf("\n\n Escolha uma opcao entre 1 e 5: ");
                scanf("%d", &opcao);
                /* Opcao invalida: volta ao rotulo REFAZ */
                if ((opcao > 5)||(opcao <1)) goto REFAZ;
                switch (opcao)
                {
                        case 1: printf("\n --> Primeira opcao.."); break;
                        case 2: printf("\n --> Segunda opcao.."); break;
                        case 3: printf("\n --> Terceira opcao.."); break;
                        case 4: printf("\n --> Quarta opcao.."); break;
                        case 5: printf("\n --> Abandonando.."); break;
                }
        }
    return(0);
}
```

Exercício

Exercício

- Escreva um programa que peça três inteiros, correspondentes a dia , mês e ano.
- Peça os números 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 (incluindo verificação de anos bissextos).
- Se estiver tudo certo imprima o número que aquele dia corresponde no ano. Comente seu programa.
- Obs.: Um ano é bissexto se for divisível por 4 e não for divisível por 100, exceto para os anos divisíveis por 400, que também são bissextos.

Até a próxima...