C
Comandos de Controle

Adriano Cruz
adriano@nce.ufrj.br

Instituto de Matemática
Departamento de Ciência da Computação
UFRJ

15 de agosto de 2013

Section Summary

1 Introdução

2 Comandos de Teste

3 Comandos de Repetição

4 Comandos de Desvio

Bibliografia

1. Adriano Cruz. *Curso de Linguagem* C, Disponível em `http://equipe.nce.ufrj.br/adriano`
2. Ulysses de Oliveira. *Programando em* C, Editora Ciência Moderna.

Bloco de Comandos

- Grupos de comandos que devem ser tratados como uma unidade lógica.
- O início de um bloco em C é marcado por uma chave de abertura ({) e o término por uma chave de fechamento (}).
- Serve para agrupar comandos que devem ser executados juntos.
- Por exemplo, usa-se bloco de comandos quando em comandos de teste deve-se escolher entre executar dois blocos de comandos.
- Pode ser utilizado em qualquer trecho de programa onde se pode usar um comando C.
- É interessante observar que um bloco de comandos pode ter zero comandos C.
- Um bloco de comandos com 0 ou 1 comando pode dispensar as chaves.

Exemplo

```
/* bloco_de_comandos */
{
     i = 0;
     j = j + 1;
     printf("%d %d\n", i, j);
}
```

Section Summary

1. Introdução
2. Comandos de Teste
3. Comandos de Repetição
4. Comandos de Desvio

Comandos de Teste

Utilidade

Os comandos de teste permitem ao computador decidir o caminho a seguir, durante a execução do programa, independentemente do usuário.

O que testam?

Estes testes são baseados em estados internos disponíveis ao processador. Estes estados podem ser resultantes de uma operação aritmética anterior, de uma operação anterior etc.

Desvios

Comando if

- O comando `if` é utilizado quando for necessário escolher entre dois caminhos.
- A forma geral do comando `if` é a seguinte:

```
if (expressão)
  bloco_de_comandos1;
else
  bloco_de_comandos2;
```

Comando if operação

- Neste comando primeiro a expressão é avaliada.
- Caso o resultado seja `verdadeiro` (QUALQUER RESULTADO DIFERENTE DE ZERO) o `bloco_de_comandos1` é executado, caso contrário o `bloco_de_comandos2` é executado.
- Pela definição do comando a expressão deve ter como resultado um valor diferente de zero para ser considerada verdade.
- Observar que somente um dos dois blocos será executado.
- Em `C` um resultado diferente de zero significa verdadeiro e zero significa falso.

Comando if sem else

- Como a cláusula `else` é opcional a forma abaixo do comando `if` é perfeitamente válida.

```
if (expressão)
  bloco_de_comandos;
```

Exemplos de if I

```
scanf("%d", &dia);
if ( dia > 31 || dia < 1 )
    printf("Dia invalido\n");

scanf("%d", &numero);
if ( numero > 0 )
  printf("Numero positivo\n");
else
  printf("Numero negativo\n");
```

Exemplos de if II

```
scanf("%f", &salario);
if (salario < 800.00)
{
  printf("Aliquota de imposto = 0.1\n");
  imposto = salario * 0.1;
}
else
{
  printf("Aliquota de imposto = 0.25\n");
  imposto = salario * 0.25;
}
```

Comandos if em escada

```
if (expressão)
    bloco_de_comandos
else if (expressão1)
    bloco_de_comandos1
else if (expressão2)
    bloco_de_comandos2
...
else
    bloco_de_comandosn
```

Exemplo: lê dados

```c
#include  <stdio.h>
int main (void)
{
    float num1, /* primeiro operando */
          num2, /* segundo operando */
          res; /* resultado da operacao */
    char oper; /* caractere que define a operacao */

    printf("\nPrograma calculadora simples.\n");
    printf("Entre com os dois operandos.\n");
    scanf("%f %f", &num1, &num2);
    getchar(); /* tirar o cr */
    printf("Qual a operacao? \n");
    oper = getchar();
```

Exemplo: Calcula

```
    if (oper == '+')
        res = num1 + num2;
    else if (oper == '-')
        res = num1 - num2;
    else if (oper == '*')
        res = num1 * num2;
    else if (oper == '/') {
             if (num2 == 0.0)
             {
                 printf("Divisao por 0!\n");
                 return 1;
             }
             else res = num1 / num2;
    else /* Nao era nenhuma das opcoes */
    {
        printf("Operacao invalida!\n");
        return 1;
    }
```

Exemplo: Resultados

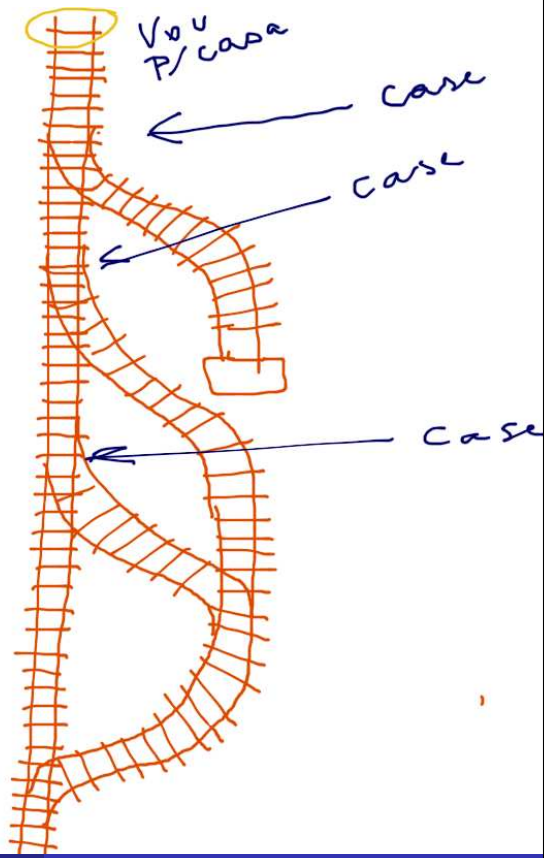
```
    printf("Resultado = %f.\n", res);
    return 0;
} /* End of main */
```

Comando `switch`

- O comando `if`, em todas suas formas, é suficiente resolver problemas de seleção de comandos.
- Porém em alguns casos o programa se torna mais trabalhoso para ser escrito e entendido.
- O comando `switch` facilita a escrita de trechos de programa em que a seleção deve ser feita entre várias alternativas.

Forma Geral

```
switch (expressão)
{
    case constante1:
        seqüência_de_comandos↩
            ;
        break;
    case constante2:
        seqüência_de_comandos↩
            ;
        break;
    case constante3:
        seqüência_de_comandos↩
            ;
        break;
      ...
    default:
    seqüência_de_comandos;
}
```



Ordem de Execução

1. Expressão é avaliada.
2. Resultado da expressão é comparado com valores dos comandoscase.
3. Se resultado da expressão for verdadeiro, executa comandos.
4. A execução continua até o fim do `switch`, ou até que um `break` seja encontrado.
5. Caso não ocorra nenhuma coincidência, os comandos associados ao comando `default` são executados.
6. O comando `default` é opcional.
7. Se não houver coincidência e o `default` não aparecer nenhum comando será executado.

1. **switch**`(expressao)`
2. `valor = expressao`
3. **case** `valor:`
4. `executa comandos`
5. `até achar um` **break**

Cuidado!

- Uma seqüência de comandos é diferente de um bloco de comandos.
- Um bloco de comandos inicia com uma chave e termina com uma chave, enquanto que uma seqüência é apenas uma série de comandos.
- Por exemplo, uma vez que um bloco de comandos foi selecionado por um comando `if` ele será executado até a última instrução do bloco, a menos que haja um comando de desvio.
- Uma série de comandos são apenas comandos colocados um após outro.

break

- O comando `break` é um dos comandos de desvio da linguagem `C`.
- O `break` é usado dentro do comando `switch` para interromper a execução da seqüência de comandos e pular para o comando seguinte ao comando `switch`.

Pontos Importantes

- O resultado da expressão deve ser um tipo enumerável, por exemplo o tipo `int`.
- Também podem ser usados tipos compatíveis com `int`, isto é, expressões com resultados tipo `char` podem ser usadas.
- Notar que caso não apareça um comando de desvio, todas as instruções seguintes ao teste `case` que teve sucesso serão executadas, mesmo as que estejam relacionadas com outros testes `case`.
- O comando switch só pode testar igualdade.

Exemplo: lê dados

```
#include  <stdio.h>
int main (void)
{
    float  num1,   /* primeiro operando */
           num2,   /* segundo operando */
           res;    /* resultado da operacao */
    char  oper;     /* caracter que define a ↩
       operacao */

    printf("Por favor entre com os operandos.\n");
    scanf("%f %f", &num1, &num2); getchar();
    printf("Qual a operacao \n");
    oper = getchar();
    printf("A operacao e %c\n", oper);
```

Exemplo: testa

```
    switch (oper) {
        case '+':
            res = num1 + num2;
            break;
        case '-':
            res = num1 - num2;
            break;
        case '*':
            res = num1 * num2;
            break;
        case '/':
            if (num2 == 0.0){
                printf("Divisao por zero.\n");
                return 1;
            }
            else {
                res = num1 / num2;
                break;
```

Exemplo: fim do teste e imprime

```
        default:
            printf("Operacao invalida!\n");
            return 2;
    } /* fim do switch */

    printf("O resultado da %c vale %f.\n",
           oper, res);
    return 0;
}
```

Comando Ternário

- O comando ternário tem este nome porque necessita de três operandos para ser avaliado.
- O comando ternário tem a seguinte forma:
  `expressão1 ? expressão2 : expressão3`
- Para avaliar o resultado total da expressão, primeiro a `expressão1` é avaliada.
- Caso este resultado seja correspondente ao valor `verdadeiro` então o resultado da expressão será igual ao resultado da `expressão2`.
- Caso contrário a `expressão3` é avaliada e se torna o resultado.

Exemplo

```c
#include <stdio.h>
int main (void) {
    float num1,  /* primeiro operando */
          num2,  /* segundo operando */
          max; /* resultado da operacao */

    printf("Imprime o maior valor de dois numeros.\n↩
       ");
    printf("Por favor entre com os dois mumeros.\n")↩
       ;
    scanf("%f %f", &num1, &num2);
    max = (num1>num2)?num1:num2;
    printf("O maior dos numeros lidos e %f.\n", max)↩
       ;
    return 0;
}
```

Section Summary

1. Introdução
2. Comandos de Teste
3. Comandos de Repetição
4. Comandos de Desvio

Comandos de Repetição

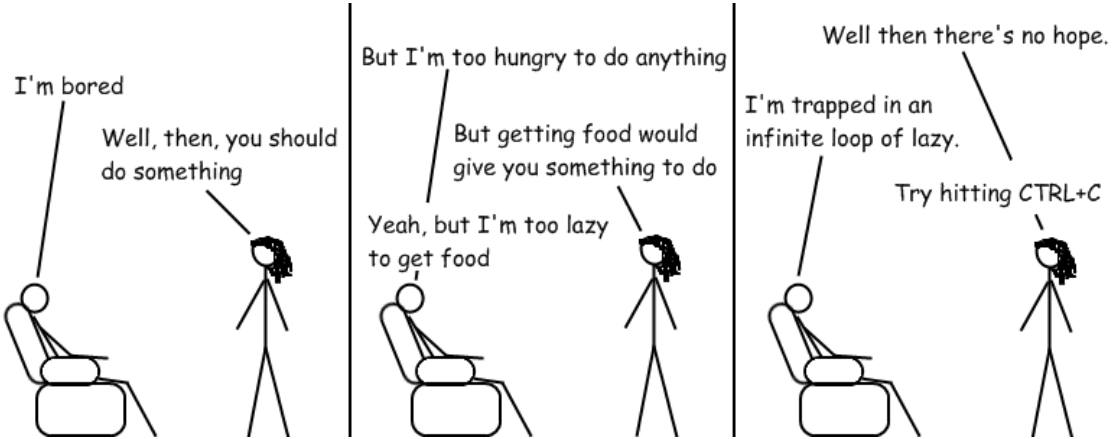
O que fazem?

Estes comandos permitem que trechos de programa sejam repetidos um certo número de vezes controlado pelo programa.

Repetir até quando?

O número de vezes que um laço será executado pode ser fixo ou depender de condições que mudam durante a execução do laço.

Laços infinitos



Comando for

- Este comando aparece em várias linguagens de programação, mas na linguagem C ele apresenta um grau maior de flexibilidade.
- Uma variável de controle, geralmente um contador, recebe um valor inicial.
- O trecho de programa que pertence ao laço é executado.
- Ao final a variável de controle é incrementada ou decrementada e comparada com o valor final.
- Caso a condição de término tenha sido atingida o laço é interrompido.

Forma Geral

```
for (expressão1; expressão2; expressão3)
     blocodecomandos;
```

- As três expressões geralmente têm os seguintes significados:
  - A `expressão1` é utilizada para inicializar a variável de controle do laço;
  - A `expressão2` é um teste que controla o fim do laço;
  - A `expressão3` normalmente faz um incremento ou decremento da variável de controle.

Passos

1. A `expressão1` é avaliada.
2. A `expressão2` é avaliada para determinar se o comando deve ser executado.
3. Se o resultado da `expressão2` for `verdadeiro` executa o o bloco de comandos
4. caso contrário o laço é terminado.
5. A expressão3 é avaliada.
6. Voltar para o passo 2.

Exemplo de for

```
for (i = 1; i <= 100; i++)
{
      printf("Numero %d\n", i);
}
```

Outro exemplo de for

```c
#include <stdio.h>
int main(void)
{
      int numero, f=1, i;

      printf("Entre com um numero positivo.\n");
      scanf("%d", &numero);
      for (i=numero; i>1; i--)
      {
          f = f * i;
      }
      printf("O f de %u vale %u.", numero, f);
      return 0;
}
```

Mais de um comando por expressão

Outra possibilidade que o comando `for` em `C` permite é a inclusão de vários comandos, separados por vírgulas, nas expressões.

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

Testes usando outras variáveis

A expressão de controle não precisa necessariamente envolver somente um teste com a variável que controla o laço. O teste de final do laço pode ser qualquer expressão relacional ou lógica. No exemplo abaixo o laço pode terminar ou porque a variável de controle já chegou ao seu valor limite ou foi batida a tecla '*', e neste caso o laço termina antecipadamente.

```c
#include<stdio.h>
int main()
{
    char c = ' ';
    int i;
    for (i=0 ; (i<5) && (c != '*'); i++ )
    {
        printf("%c\n", c);
        c = getchar();
    }
    return 0;
```

Expressões faltando

Um outro ponto importante do for é que nem todas as expressões precisam estar presentes. No exemplo abaixo a variável de controle não é incrementada. A única maneira do programa terminar é o usuário bater o número -1.

```
#include<stdio.h>
int main()
{
    int i;
    for (i=0 ; i != -1 ; )
    {
        printf("%d\n",i );
        scanf ("%d", &i);
    }
    return 0;
}
```

Outras expressões faltando

```c
#include <stdio.h>
int main()
{
    int i;
    int n;
    for (i = 0; ; i++)
    {
        scanf("%d", &n);
        printf("numero %d\n", n);
        if (n == 5) break;
    }
    return 0;
}
```

Laço Infinito

Uma construção muito utilizada é o laço infinito. No laço infinito o programa pára quando se executa o comando `break`. O trecho de programa a seguir somente pára quando for digitada a tecla `'s'` ou `'S '`.

```
for ( ; ; )
{
    printf("\nVoce quer parar?\n");
    c = getchar();
    if (c == 'S' || c == 's') break;
}
```

Laços `for` aninhados

Uma importante construção aparece quando colocamos como comando a ser repetido um outro comando `for`.

```
#include <stdio.h>
int main(void)
{
    int i, j;

    printf("Imprime tabuada de multiplicacao.\n");
    for (i=1 ; i<10 ; i++)
    {
        printf("Tabuada de %d\n", i);
        for (j=1; j<10; j++)
        {
            printf("%d x %d = %d\n", i, j, i * j);
        }
    }
```

while

O comando `while` tem a seguinte forma geral:

```
while (expressão)
    bloco_de_comandos
```

Passos

1. A expressão é avaliada.
2. Se o resultado for `verdadeiro` então o bloco de comandos é executado, caso contrário a execução do bloco é terminada.
3. Voltar para o passo 1.

Uma característica do comando `while`, é que o bloco de comandos pode não ser executado caso a condição seja igual a `falso` logo no primeiro teste.

Exemplo I

```
i = 1;
while (i <= 100)
{
    printf("Numero %d\n", i);
    i++;
}
```

Exemplo II

```c
#include <stdio.h>
int main(void)
{
    int c;

    puts("Tecle c para continuar.\n");
    while ((c=getchar()) == 'c')
    {
        puts("Nao Acabou.\n");
        getchar(); /* tira o enter */
    }
    puts("Acabou.\n");
    return 0;
}
```

do - while()

O comando do-while() tem a seguinte forma geral:

```
do
    bloco_de_comandos
while (expressão);
```

Passos

1. Executa o bloco de comandos.
2. Avalia a expressão.
3. Se o resultado da expressão for `verdadeiro` então volta para o passo 1, caso contrário interrompe o `do-while`.

Observar que neste comando a expressão de teste está após a execução do comando, portanto o bloco de comandos é executado pelo menos uma vez. A execução do comando segue os seguintes passos:

Exemplo

```
i = 1;
do
{
    printf("Numero %d\n", i);
    i++;
} while (i <= 100);
```

Section Summary

1. Introdução
2. Comandos de Teste
3. Comandos de Repetição
4. Comandos de Desvio

Break

1. O comando `break` pode ser tanto usado para terminar um teste `case` dentro de um comando `switch` quanto interromper a execução de um laço.
2. Quando o comando é utilizado dentro de um comando `for` o laço é imediatamente interrompido e o programa continua a execução no comando seguinte ao comando `for`.

Break

```c
for (i = 0; i < 100; i++)
{
    scanf("%d", &num);
    if (num < 0) break;
    printf("%d\n", num);
}
```

Break

1. O comando `continue` é parecido com o comando `break`.
2. A diferença é que o comando `continue` simplesmente interrompe a execução da iteração corrente passando para a próxima iteração do laço, se houver uma.
3. No comando `for` o controle passa a execução da `expressão3`.
4. Nos comandos `while` e `do-while` o controle passa para a fase de testes.

Exemplo

```c
for (i = 0; i < 100; i++)
{
    scanf("%d", &num);
    if (num < 0) continue;
    printf("%d\n", num);
}
```

goto

1. O comando `goto` causa um desvio incondicional para um outro ponto da função em que o comando está sendo usado.
2. O comando para onde deve ser feito o desvio é indicado por um rótulo, que é um identificador válido em `C` seguido por dois pontos.
3. É importante notar que o comando `goto` e o ponto para onde será feito o desvio podem estar em qualquer ponto dentro da mesma função.
4. A forma geral deste comando é:

```
goto rótulo;
...
rótulo:  comando;
```

Atenção

Este comando durante muito tempo foi associado a programas ilegíveis. O argumento para esta afirmação se baseia no fato de que programas com comandos `goto` perdem a organização e estrutura porque o fluxo de execução pode ficar saltando erraticamente de um ponto para outro.

Atualmente as restrições ao uso do comando tem diminuído e seu uso pode ser admitido em alguns casos.

exit()

1. A função exit provoca a terminação de um programa, retornando o controle ao sistema operacional.
2. O protótipo da função é a seguinte:
   `void exit (int codigo);`
3. Observar que esta função interrompe o programa como um todo.
4. O código é usado para indicar qual condição causou a interrupção do programa.
5. Usualmente o valor 0 indica que o programa terminou sem problemas.
6. Um valor diferente de 0 indica um erro.

return

1. O comando `return` é usado para interromper a execução de uma função e retornar um valor à função que chamou esta função.
2. Caso haja algum valor associado ao comando `return` este é devolvido para a função, caso contrário um valor qualquer é retornado.
3. A forma geral do comando é:
   `return expressão;`
4. Notar que a expressão é opcional.
5. A chave que termina uma função é equivalente a um comando `return` sem a expressão correspondente.
6. É possível haver mais de um comando `return` dentro de uma função.
7. O primeiro que for encontrado durante a execução causará o fim da execução.
8. Uma função declarada como do tipo `void` não pode ter um comando `return` que retorne um valor.

The End