UnB Departamento de Ciência da Computação

2015/2

Fluxo de Controle

Prof. Guilherme N. Ramos

O *fluxo de controle* de um algoritmo é a ordem em que são executadas suas instruções [2], que - portanto - define seu comportamento. Todo algoritmo computacional é baseado em 3 conceitos: instruções sequenciais, bifurcação, e repetição.

1 Sequenciais

Em um algoritmo, as instruções são executadas de forma *sequencial*, uma após a outra, na mesma ordem em que se encontram na representação do algoritmo. Esta execução ordenada é uma condição essencial para funcionamento correto da maioria dos algoritmos, pois o comportamento só é bem definido neste caso. No exemplo abaixo, as horas vividas só podem ser [corretamente] calculadas porque em etapas anteriores obteve-se todos os dados necessários.

1-horas.c

```c
    const int DIAS_NO_ANO = 365, HORAS_NO_DIA = 24;
    int idade_em_anos, idade_em_dias, idade_em_horas;

    printf("Olá! Quantos anos você tem?\n");
    scanf("%d", &idade_em_anos);

    /* Agora que se tem a idade em anos, é possível calcular quantos dias já
     * foram vividos (aproximadamente). */
    idade_em_dias = idade_em_anos*DIAS_NO_ANO;

    /* E agora que se sabe quantos foram os dias, pode-se calcular quantas horas
     * foram vividas (aproximadamente). */
    horas_vividas = idade_em_dias*HORAS_NO_DIA;

    /* Só agora é possível mostrar o resultado. */
    printf("Sabia que já viveu %d horas?\n", horas_vividas);
```

É claro, a execução sequencial possibilita a resolução de problemas mais interessantes como a aproximação do valor da raiz quadrada de um número. Conforme o método babilônico, o valor da raiz pode ser encontrado com uma estratégia de aproximações sucessivas. Assim, dados um número inteiro positivo n, e uma estimativa inicial r de sua raiz quadrada, também inteiro e positivo, pode-se aproximar o valor de r para o valor correto por meios de instruções sequenciais.

2-raiz2-0.c

```c
  printf("Qual o valor de n?\n");
  scanf("%d", &n);

  printf("Qual sua estimativa inicial para a raiz de %d?\n", n);

  /* 1a tentativa */
  scanf("%d", &r);
  printf("%d*%d = %d\n", r, r, r*r);

  /* 2a tentativa */
  r = (r+(n/r))/2; /* Algo estranho com esta instrução... */
  printf("%d*%d = %d\n", r, r, r*r);

```

```
14  /* 3a tentativa */
15  r = (r+(n/r))/2;
16  printf("%d*%d = %d\n", r, r, r*r);
17
18  printf("Depois de 3 tentativas, a aproximação da raiz de %d é %d.\n", n, r);
```

O custo envolvido na execução instruções sequenciais é proporcional a quantidade de instruções.

2 Bifurcação

A evolução do processo de um algoritmo é sequencial, mas a sequência a ser executada pode ser alterada com uma estrutura de *bifurcação* (ou *seleção/decisão*). Desta forma, um conjunto de instruções é executado apenas se certa condição lógica for avaliada for verdadeira [1].

Em uma estrutura condicional *simples* (`if`), há um teste que define se determinado conjunto de instruções será executado ou não. Já em uma condicional *dupla* (`if/else`) há dois conjuntos de instruções que são exclusivos, apenas um deles é executado (sequencialmente) conforme o teste da condição. Por fim, há *condicional múltipla* (`switch`), em que, dentre múltiplas opções, apenas um bloco é executado conforme o resultado do teste.

A bifurcação do fluxo possibilita um processamento mais adequado, pois pode-se verificar certas condições para decidir como prosseguir, bem como uma interação mais interessante com o usuário, que pode ser informado dos acontecimentos. Considerando novamente o problema de aproximação do valor da raiz quadrada de um número pelo método babilônico, pode-se verificar os valores recebidos do usuário de modo que os cálculos só seja realizados caso estes valores sejam válidos.

2-raiz2-1.c

```
1   printf("Qual o valor de n?\n");
2   scanf("%d", &n);
3
4   if(n < 0) {
5     printf("Não sei calcular a raiz quadrada de número negativo.\n");
6   }
7   else {
8     printf("Qual sua estimativa inicial para a raiz de %d?\n", n);
9     scanf("%d", &r);
10
11    if(r <= 0)
12      printf("A raiz não pode ser menor que zero.\n");
13    else {
14      r = (r+(n/r))/2;
15      r = (r+(n/r))/2;
16
17      printf("Depois de 3 tentativas, a aproximação da raiz de %d é %d.\n", n, r);
18    }
19  }
```

Caso haja algo errado, a informação do que está incorreto é passada ao usuário, para que tome as devidas providências. A bifurcação possibilita garantir que certos cálculos sejam feitos, como também pode-se garantir que certas condições inaceitáveis nunca sejam processadas (por exemplo, divisão por zero).

O custo envolvido na execução de instruções com bifurcações é proporcional a quantidade de instruções. Veja mais em `1_Bifurcacao`.

3 Repetição

Certos problemas exigem a execução de um mesmo conjunto de instruções [sequenciais] *repetidas* vezes, um procedimento que deve ser automatizado! Um *laço de repetição* contém um conjunto de instruções que são executadas (sequencialmente) enquanto determinada condição for verdadeira. Cada execução é chamada de *iteração* [1].

O laço *Enquanto-Faça* (`while`) avalia a condição e, se verdadeira, executa as instruções. Já o laço *Faça-Enquanto* (`do-while`) avalia a condição de parada *após a execução das instruções*. Sabendo *quantas são as iterações*, pode-se simplesmente definir um laço *Para-Até-Faça* (`for`), que é executado exatamente este número de vezes.

```c
for(i = 1; i <= 7; ++i)
    printf("Goooooooool da Alemanha!\n");

/* "for" é mais compacto que o código equivalente */

i = 1;
while(i <= 7) {
    printf("Goooooooool da Alemanha!\n");
    ++i;
}
```

Os laços de repetição vistos são quase intercambiáveis entre si, todos têm:

- inicialização das condições de parada;
- teste das condições de parada; e
- atualização da condições de parada.

Os laços de repetição possibilita uma abordagem muito interessante a algoritmos de aproximação. Considerando novamente o problema do valor da raiz quadrada de um número pelo método babilônico, pode-se repetir as iterações uma quantidade arbitrária de vezes sem ter de repetir o código. Este tipo de análise numérica é muito utilizado. Por exemplo, para a proximar o valor $r = \sqrt{n}$:

`6-raiz2-2.c`

```c
      do {
        r = (r+(n/r))/2;
        ++i;
      } while(i < iteracoes);
```

É preciso muita atenção ao utilizá-los para evitar implementar um laço infinito e a execução um número incorreto de vezes. O custo envolvido na execução de instruções com repetições é proporcional a quantidade de instruções e a quantidade de repetições. Veja mais exemplos em `2_Repeticao`.

Todos os algoritmos computacionais são implementados pela combinação de instruções sequenciais, bifurcações, e repetições, desde os mais simples até os mais complicados.

Referências

[1] Luis Joyanes Aguilar. *Fundamentos de Programação: Algoritmos, estruturas de dados e objetos.* McGraw-Hill, 3a edition, 2008.

[2] Brian W. Kernighan and Dennis M. Ritchie. *C: a linguagem de programação padrão ANSI.* Campus, Rio de Janeiro, 1989.