4. ESTRUTURAS DE CONTROLE

Os comandos de fluxo de controle de uma linguagem de programação especificam a ordem em que processamento é feito. Uma expressão, tal como x = 0, torna-se um comando quando seguida por ponto e vírgula. Tanto em C, como em C++, o ponto e vírgula é o "terminador" de comandos, que controla a seqüência de execução dos programas. As chaves, "{" e "}", são usadas para agrupar declarações e comandos num comando composto ou bloco, de modo que são sintaticamente equivalentes a um único comando. Neste capítulo serão descritos os comandos de seleção, condição e iteração da linguagem C++. Estes comandos controlam como outros comandos e operações de um programa serão executados.

4.1. Comandos de Seleção

Estes comandos, como o próprio nome diz, permitem fazer uma seleção, a partir de uma ou mais alternativas, da ação que o computador deve tomar. A seleção é baseada no valor de uma expressão de controle. Em C++, um valor pode ser testado através do comando ***if*** ou do comando ***switch***.

O comando *if* possui a seguinte forma:
if (<condição>) <comando>
if (<condição>) <comando> *else* <comando>

Já o comando *switch* possui a seguinte forma:
switch (<condição>) <comando>

Os operadores de comparação, que são ==, !=, <, <=, > e >=, retornam *true* (verdadeiro) se a comparação é verdadeira e *false* (falso) caso contrário. No comando *if*, o primeiro (ou somente) <comando> é executado se a expressão resulta em não zero (verdadeiro) e o segundo <comando> (*else*) é executado caso contrário. É importante salientar que os operadores lógicos && (e), || (ou) e ! (não), são muito usados em condições. Exemplo:

```
if ( (a<=b) && (a > 0) )
        max = b;
else
        max = 100;
```

Os parênteses da condição não são obrigatórios, mas são geralmente utilizados para tornar o código mais legível.

O comando *switch* é usado para várias seleções, o que corresponde em outras linguagem aos comandos *case* ou *select*. O <comando> normalmente corresponde a um bloco, cujos comandos são precedidos por *labels case*. Cada *case label* representa um valor possível para expressão de controle [3, 8]. Por exemplo:

```
switch (val) {
        case <valor inteiro 1> : <comando>
                                 break;
        case <valor inteiro 2> : <comando>
                                 break;
        default : <comando>
                  break;
}
```

No exemplo anterior, pode-se notar que cada *case* acaba com um *break*, o que faz a execução pular para o final do corpo *switch*. Esse é o modo convencional de utilização do comando, mas não obrigatório. Se o comando *break* não é encontrado, o código para os comandos *case* seguintes são executados até ele ser achado [4].

Torna-se interessante comentar que o comando *switch* pode alternativamente ser escrito como um conjunto de comandos *if*:

```
if ( val == 1 )
        <comando>
else if ( val == 2 )
        <comando>
else
        <comando>
```

O significado é o mesmo, mas preferencialmente utiliza-se o *switch* devido a natureza da operação, deixando o código mais legível [8].

4.2. Comandos de Iteração

Os comandos de iteração (ou repetição) especificam a execução de *loops*, isto é, fazem com que os outros comandos que eles controlam sejam executados zero ou mais vezes. Em C++ um *loop* pode ser determinado através dos comandos ***while***, ***do*** e ***for***, que possuem as seguintes formas, respectivamente:

while (<expressão_de_controle >) { <comando> }

do { <comando> } *while* (<expressão_de_controle >)

for (<comando_inicialização>; <expressão_de_controle>; <controle_de_passo>) { <comando> }

Em todos estes comandos, enquanto a <expressão_de_controle> é diferente de zero, <comando> é executado repetidamente. Ao se pensar em <expressão_de_controle> como uma condição, com o valor zero representando falso e não zero representando verdadeiro, então o <comando> controlado será executado enquanto a condição é verdadeira.

No comando *while* antes de cada possível execução a <expressão_de_controle> é avaliada. Se o valor é zero (falso), <comando> não é executado, se for não zero (verdadeiro), <comando> é executado e depois <expressão_de_controle> é avaliada novamente, e assim por diante. O exemplo abaixo ilustra o comando *while*:

```
int contador = 1;
while ( contador <= 100 ) {
        cout << contador << "\n";
        contador ++;
}
```

O comando *do* trabalha da mesma maneira que *while*, exceto pelo fato de que o valor da <expressão_de_controle> é testado depois de cada execução de <comando> ao invés de ser testado antes. Assim, <comando> é sempre executado pelo menos uma vez; depois de cada execução a <expressão_de_controle> é avaliada para verificar se o *loop* continua ou não. Este comando é útil quando <comando> deve ser executado uma vez antes que <expressão_de_controle> seja avaliada. O exemplo abaixo, equivalente ao anterior mostra o funcionamento deste comando de iteração

```
int contador = 1;
do {
        cout << contador << "\n";
        contador ++;
} while ( contador <= 100 )
```

Já o comando *for* é projetado para expressar *loops* regulares. *For* possui uma variável de *loop* que é inicializada em <comando_inicialização>, usada em <expressão_de_controle> e incrementada ou decrementada em <controle_de_passo>, como mostra o exemplo abaixo, que também é equivalente aos anteriores:

```
for ( contador = 1; contador <= 100; contador++)
        cout << contador << "\n";
```

Se nenhuma inicialização é necessária, <comando_inicialização> pode ser omitido. Se <expressão_de_controle> é omitida o comando *for* fica em *loop* infinito, a menos que o usuário determine uma

saída explicitamente através do comando *break*, *return* ou *exit*, por exemplo. Se <controle_de_passo> é omitido, deve-se, então, atualizar a variável de *loop* em <comando>. O comando *for* também é útil para expressar um *loop* sem uma condição de término explícita:

```
for ( ; ; ) { ... }  // loop infinito
```

É preferível usar o comando *while*, ao invés do *for*, quando não há uma variável de *loop* "óbvia" ou quando a atualização da variável de *loop* ocorre naturalmente ao longo de <comando>. O comando *do* pode gerar erro e confusão, porque <comando> é sempre executado uma vez antes da <expressão_de_controle> ser avaliada [3, 8].