3. ESTRUTURAS DE CONTROLE DE FLUXO

Os comandos de controle de fluxo 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, pois 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. Além disso, existem comandos de seleção, condição e iteração que controlam como outros comandos e operações de um programa serão executados. Tais comandos estão descritos a seguir.

3.1. 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 dos comando ***if***, ***switch*** ou através do operador ternário **?**.

Forma do comando *if*:

```
if (<condição>) <comando>
if (<condição>) <comando> else <comando>
```

Forma do comando *switch*:

```
switch (<condição>) <comando>
```

Forma do operador ternário:

```
 <expressão1> ? <expressão2> : <expressão3>;
```

No comando *if*, o primeiro <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 !, 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.

Um comando *if* pode ser aninhado, isto é, um comando *if* pode ser objeto de outro *if* ou *else*. Em C, um comando *else* sempre se refere ao comando *if* mais próximo, que está dentro do mesmo bloco do *else* e não está associado a outro *if*. Por exemplo [SCH 96]:

```
if (i)
{
        if (j) comando 1;
        if (k) comando 2;       // este if
        else comando 3;         // está associado a este else
}
else comando 4;                 // associado a if (i)
```

Outro exemplo de comandos *if* aninhados é apresentado a seguir [PIN 00]:

```
if (expr_log)
{
        comando1; // executado se expr_log for VERDADEIRA
        comando11;
        if (expr_log2)
        {
                comando2; // executado se expr_log e expr_log2
                comando21; // forem ambas VERDADEIRAS
        }
        else
        {
                comando3; // executado se expr_log for VERDADEIRA
                comando31; // e expr_log2 for FALSA
        }
        comando4; // executado se expr_log for VERDADEIRA
}
else
{
        comando5; // executado se expr_log for FALSA
        comando51;
        if (expr_log3)
        {
                comando6; // executado se expr_log for FALSA
                comando61; // e expr_log3 for VERDADEIRA
        }
        else
        {
                comando7; // executado se expr_log for FALSA
                comando71; // e expr_log3 for FALSA
        }
        comando8; // executado sempre que expr_log for FALSA
}
comando9; // executado sempre
:
```

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 *case labels*. Cada *case label* representa um valor possível para expressão de controle, e o parâmetro do *switch* deve ser uma variável do tipo *int* ou *char* [GRA 91, STR 97]. 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* (comando de desvio), 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 [SWA 93]. O comando *default* é executado se nenhuma coincidência for detectada. Tal comando é opcional e, se não estiver presente, nenhuma ação será realizada se todos os testes falharem. O padrão ANSI C especifica que um *switch* pode ter pelo menos 257 comandos *case* [SCH 96].

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 [STR 97]. Em outras palavras, o comando *switch* difere do comando *if* porque só pode testar igualdade, enquanto *if* pode avaliar uma expressão lógica ou relacional [SCH 96].

O operador ternário é muito poderoso e substitui certas sentenças da forma *if-else*. De acordo com a forma geral apresentada anteriormente (*<expressão1> ? <expressão2> : <expressão3>;*), o operador *?* funciona da seguinte maneira: <expressão1> é avaliada; se ela for verdadeira, então <expressão2> é avaliada e se torna o valor da expressão. Se <expressão1> é falsa, então <expressão3> é avaliada e se torna o valor da expressão. Por exemplo, em

```
x = 10;
y = x>9 ? 100 : 200;
```

a "*y*" é atribuído o valor 100. Se "*x*" fosse menor que 9, "*y*" teria recebido o valor 200. O mesmo código usando o comando *if-else* é [SCH 96]:

```
x = 10;
if (x>9) y = 100;
else y = 200;
```

3.2. Comandos de Repetição

Os comandos de repetição (ou iteração) especificam a execução de laços, isto é, fazem com que os outros comandos que eles controlam sejam executados zero ou mais vezes. Em C um laço 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 ) {
        printf("%d \n", contador);
        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 {
        printf("%d \n", contador);
        contador ++;
} while ( contador <= 100 );
```

Já o comando *for* é projetado para expressar laços regulares. *for* possui uma variável de laço 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++)
        printf("%d \n", contador);
```

Se nenhuma inicialização é necessária, <comando_inicialização> pode ser omitido. Se <expressão_de_controle> é omitida o comando *for* fica em um laço 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 laço em <comando>. O comando *for* também é útil para expressar um laço sem uma condição de término explícita:

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

Torna-se interessante comentar que C oferece diversas variações que aumentam a flexibilidade e a aplicação do comando *for*. Uma das variações mais comuns usa o operador vírgula para permitir que duas ou mais variáveis controlem o laço. Por exemplo, as variáveis "x" e "y" controlam o seguinte laço e ambas são inicializadas dentro do comando *for*:

```
for (x=0, y=0; x+y<10; ++x) {
        scanf("%d",&y);
        :
        :
}
```

Neste caso a vírgula separa os dois comandos de inicialização e cada vez que "x" é incrementado, o laço repete e o valor de "y" é determinado pela entrada do teclado. Embora os valores de "y" sejam definidos pela entrada do teclado, "y" deve ser inicializado com zero de forma que seu valor seja definido antes da avaliação da expressão condicional. Outra variação que poderia ser feita, é o incremento, ou decremento, da variável "y" no comando *for*. Por exemplo:

```
for (x=5, y=5; x+y>0; --x, --y) {
        :
}
```

É preferível usar o comando *while*, ao invés do *for*, quando não há uma variável de laço "óbvia" ou quando a atualização da variável de laço 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 [GRA 91, SCH 96, STR 97].

3.3. Comandos de Desvio

C tem quatro comandos que realizam um desvio incondicional: *return*, *goto*, *break* e *continue*. Os dois primeiros podem ser usados em qualquer lugar do programa, e os dois últimos podem ser usados em conjunto com qualquer dos comandos de laço.

O comando ***return*** é usado para retornar de uma função. Ele é um comando de desvio porque faz com que a execução retorne ao ponto em que a chamada à função foi feita. Se *return* tem um valor associado a ele, esse valor é o valor de retorno da função. Se nenhum valor de retorno é especificado, assume-se que apenas lixo ou zero é retornado. Pode-se usar vários comandos *return* dentro de uma

função, porém esta parará de executar tão logo encontre o primeiro *return*. A forma geral do comando *return* é

return <expressão>;

É importante lembrar que <expressão> é opcional. Entretanto, se estiver presente, ela se tornará o valor da função.

Não há nenhuma situação na programação que necessite do ***goto***, entretanto seu uso pode ser conveniente em raras situações. Este comando requer um rótulo (identificador seguido por dois-pontos) para sua operação, e sua forma geral é:

goto <rótulo>;
:
<rótulo>:

O comando ***break*** tem duas utilizações: para terminar um *case* em um comando *switch*, ou para forçar o término imediato de um laço, evitando o teste condicional normal do laço. Quando o comando *break* é encontrado dentro de um laço, o laço é imediatamente terminado e o controle do programa retorna no comando seguinte ao laço. Exemplo:

```
#include <stdio.h>
void main(void)
{
        int t;
        for (t=0; t<100; t++) {
                printf("%d", t);
                if (t==10) break;
        }
}
```

Torna-se importante comentar que da mesma forma que é possível sair de um laço, pode-se sair de um programa usando a **função *exit()*** da biblioteca padrão. Essa função provoca o término imediato do programa inteiro, forçando um retorno ao sistema operacional.

O comando ***continue*** trabalha de uma forma um pouco parecida com a do comando *break*, porém, em vez de forçar o término, *continue* força que ocorra a próxima iteração do laço, pulando qualquer código intermediário. Para o laço *for*, *continue* faz com que o teste condicional e a porção de incremento do laço sejam executados. Para os laços *while* e *do-while*, o controle do programa passa para o teste condicional. Por exemplo, o seguinte programa conta o número de espaços contidos em uma cadeia de caracteres inserida pelo usuário:

```
/* Conta Espaços */
#include <stdio.h>
void main(void)
{
        char s[80], *str;
        int space;
        printf("Digite uma string: ");
        gets(s);
        str = s;
        for (space=0; *str; str++) {
                if (*str != ' ') continue;
                space++;
        }
        printf("%d espacos \n", space);
}
```

Cada caractere é testado para ver ser é um espaço. Se não é, o comando *continue* força o *for* a iterar novamente. Se o caractere é um espaço, a variável "*space*" é incrementada.