Introdução à Linguagem C

A linguagem C, desenvolvida em paralelo com o UNIX é atualmente uma das linguagens de programação mais utilizadas. Suas principais características são:

- Simples e de aprendizado fácil.
- Uso amplamente difundido.
- Baseada em construções simples, que usam os recursos da máquina de forma eficiente.
- Oferece recursos de modularização necessários ao desenvolvimento de aplicações de grande porte.
- Disponível em várias plataformas.

Um programa simples

O programa abaixo é o clássico "Hello, world", usado na maioria dos textos introdutórios para descrever a "cara" de um programa em C:

```
#include <stdio.h>
int main()
{
   printf("Hello, world\n");
}
```

O que este programa faz é basicamente escrever na saída padrão o *string* "Hello, world". Alguns detalhes deve ser explicados:

```
#include <stdio.h>   Indica que este programa deve utilizar as
                     definições feitas no arquivo 'sdtio.h' para
                     as  funções básicas de entrada e saída.

int main() { ...}    Corresponde ao "corpo" do programa. É a partir
                     dele que se inicia a execução do programa.

printf(...)          Comando que indica "escreva o texto entre
                     parêntesis através de um dispositivo de saída
                     (neste caso, a saída padrão).

"Hello, world\n"     Cadeia de caracteres (string). A seqüência de
                     caracteres '\n' representa o caracter newline.
```

Outro exemplo: Variáveis, constantes, expressões e atribuição

O exemplo a seguir ilustra alguns conceitos importantes em C:

```
#include <stdio.h>
int main()
{
  int a,b,c,soma;

  a = 1; b = 2; c = 3;
  soma = a + b + c;
  printf("soma = %d\n",soma);
}
```

Nesse exemplo, o comando

```
int a, b, c, soma;
```

declara *a*, *b*, *c* e *soma* como *variáveis* do tipo *inteiro*. Uma variável do tipo inteiro pode receber qualquer valor inteiro permitido na máquina. Os comandos

```
a = 1; a = 2; c = 3;
```

são comandos de *atribuição*. Um comando de atribuição indica que uma *variável* assume passa a assumir o valor de uma *expressão*. Nos comandos acima, as *expressões* são definidas pelas *constantes inteiras* 1, 2 e 3. Esses comandos são executados na ordem em que aparecem. Ao atribuir um *valor* a uma variável, o valor anterior da mesma é simplesmente perdido (uma variável sempre tem um *único* valor. Considere-se por exemplo a sequência de comandos abaixo:

```
a = 1; a = 2; a = 3;
```

Após a execução dessa sequência de comandos, o valor de *a* será igual a 3 (a cada atribuição, o "valor anterior" da variável *a* é destruído). O valor de uma variável pode ser utilizado numa expressão (soma=a+b+c). Isso vale também para a variável à esquerda do sinal de atribuição ("=").

Exemplo:

```
a = 1;
a = a + 2;
a = a + 3;
```

Em C, todo comando encerra com um ";". A separação de linhas tem a mesma função que um espaço em branco. A sequência de comandos acima é equivalente a

```
a = 1; a = a + 2; a = a + 3;
```

O comando *printf*

O comando *printf* é utilizado para escrever dados em forma de texto na saída padrão ou num arquivo. Por enquanto, estamos utilizando a forma mais simples desse comando:

```
printf(< sequência de caracteres entre aspas > , variável1, variável2, ...);
```

Nessa forma, os caracteres da sequência são escritos na saída padrão. Ao serem escritos, alguns caracteres têm um significado especial. Por exemplo o caracter '%' é usado para indicar que o caracter seguinte é um *especificador de formato*, e ambos devem ser substituídos pelo valor de uma variável da lista de variáveis. Por exemplo, no comando

```
printf("o valor de a é : %d", a)
```

o par de caracteres '%d' será substituído pelo valor da variável *a*. O caracter 'd' indica que esse valor deve ser escrito como um número inteiro *decimal*. Outro exemplo(completo):

```
#include <stdio.h>
int main()
{
  int a,b,c;

  a = 1; b = 2; c = 3;
  printf("valor de a:%d valor de b: %d valor de c:%d \n ",a,b,c);
}
```

Neste exemplo, a primeira ocorrência de "%d" se refere à primeira variável da lista (*a*), a segunda à segunda variável (*b*) e assim sucessivamente. O par "\n" está sendo utilizado para representar o caracter *new line* que causa a mudança de linha na saída padrão. O resultado da execução desse programa deverá ser:

```
valor de a:1 valorde b:2 valor de c:3
```

Entrada de Dados

No exemplo abaixo, o comando scanf está sendo usado para entrada de dados. Seu uso é semelhante ao uso do printf: o primeiro argumento é uma cadeia de caracteres na qual o par "%d" indica um valor inteiro decimal. Os valores lidos são associados às variáveis que aparecem a seguir. Os nomes das variáveis aparecem precedidos de '&'.

```
#include
int main()
{
 int a,b,c;
 scanf("%d %d %d", &a,&b,&c);
 printf("a:%d b:%d c:%d \n",a,b,c);
}
```

Os 'comandos' *scanf* e *printf* na verdade são funções definidas na biblioteca *stdio* que internamente chamam funções da *biblioteca padrão* que por sua vez se utilizam dos serviços oferecidos pelo sistema operacional para realizar as operações de leitura e escrita.

As funções *scanf* e *printf* trabalham basicamente com sequências de caracteres. Elas têm a capacidade de converter caracteres para outros tipos de dados e vice-versa. Essa conversão é indicada na sequência de caracteres através de "especificadores de formato", sempre precedidos de '%'. Nos exemplos apresentados até aqui, utilizamos apenas "%d" para indicar a conversão de caracteres para inteiros (no caso de *scanf*) ou de inteiros para caracteres (no caso de *printf*).

O caracter '&' antes do nome de cada variável a ser lida indica *referência* ou *endereço* da variável. É através dessa referência ou endereço que o valor da mesma será aualizado após a leitura do dado correspondente.

Um pouco mais sobre expressões

Nos exemplos anteriores, foram utilizadas expressões aritméticas simples como "a+1", "a+b+c". Expressões são formadas por *operandos* ou *operadores*. Operandos são valores (até o momento, *inteiros*), que aparecem na expressão como *constantes*, *variáveis* ou *expressões*. A linguagem dispõe de um grande número de operadores, mas temporariamente vamos nos restringir aos *operadores aritméticos*:

```
+   soma
-   subtração
*   multiplicação
/   divisão
%   resto da divisão
```

Parentesis podem ser utilizados para forçar o cálculo de uma expressão numa determinada ordem. Exemplos:

```
a + (b / 2)
(a * 2) + 1
((((a*2) + 1)*2)+1)*2+1)
```

Uma expressão como *a + b * c* será calculada como *a + (b * c)* porque o cálculo da mesma leva em conta a *precedência* entre os operadores. Em C essa precedência é definida da seguinte forma (para os operadores aritméticos):

```
+, -    menor precedência
*, /,%  maior precedência
```

Os "operadores combinados"

Em programação, são frequentes comandos como:

```
a = a + b;
x = x - y;
p = p * 2;
```

Nesses comandos, o valor a ser atribuido à variável à esquerda é o resultado de uma operação na qual o valor dessa mesma variável é utlilizado. Em C é possível escrever esse comando de forma mais compacta através da "combinação" da operação\ com a atribuição (+=, -=, *=, etc.). Os comandos mostrados acima, por exemplo, poderiam ser reescritos como:

```
a += b;
x -= y;
p *= 2;
```

Os operadores ++ e --

Outro tipo de comando comum em programação é aquele no qual apenas se incrementa ou decrementa o valor de uma variável de 1. Em C é possível escrever esse tipo de comando através dos operadores "++" e "--". Exemplos:

```
++i;  equivalente a i = i + 1;
++i;  equivalente a i = i - 1;
```

Esses operadores podem ser utilizados de duas formas: antes ou depois do operando (p. ex. ++i ou i++). No primeiro a operação é feita *antes* do uso do valor do operando. No segundo caso, a operação é feita *depois* do uso do valor. Exemplos:

```
i = 10; v = ++i; /* o valor atribuído a v é 11 */
i = 10; v = i++; /* o valor atribuido a v é 10 */
                 /* nos 2 casos, o valor final de i é 11 */
```

Comentários

Um comentário em C tem a seguinte forma:

```
/* texto qualquer */
```

Comentários podem aparecer em qualquer lugar do programa onde caberia um espaço (' '). Seu significado é exatamente o mesmo de um espaço em branco. Comentários delimitados por "/*" e "*/" podem ocupar várias linhas.

Um exemplo: Cálculo do peso ideal

O programa abaixo calcula o "peso ideal" de uma pessoa a partir da sua altura, segundo uma fórmula. Note que se trata apenas de um exemplo de programação e não deve ser utilizado para orientar uma dieta.

```
#include <stdio.h>

/*
   Determina o "peso ideal" de uma pessoa a partir da sua altura.
   Trata-se apenas de um exemplo de programação - não segue nenhuma
   orientação médica.
*/

int main()
{
  int a; /* altura */
  int p; /* peso */

  printf("Altura em centímetros(entre 120 e 200):");
  scanf("%d",&a);

  p = (a-119)*5/8+45;
  printf("Peso ideal: %d \n",p);
}
```

O Comando Condicional

O comando condicional permite alterar a sequência de execução dos comandos com base numa condição. O comando condicional em C tem duas formas básicas:

```
if( condição ) comando
if( condição ) comando1 else comando2
```

A *condição* é uma expressão qualquer, cujo valor é calculado ao se executar o comando condicional. Um valor diferente de zero para *condição* indica *verdadeiro* e um valor igual a zero indica *falso*.

Na primeira forma a execução do comando condicional implica em:

1. Cálculo da expressão que define a condição.
2. *comando* será executado somente se o valor dessa expressão indicar *verdadeiro*.

A execução do comando condicional na segunda forma implica em:

1. Cálculo da expressão que define a condição.
2. Se a expressão indicar *verdadeiro*, *comando1* será executado.
3. Se a expressão indicar *falso*, *comando2* será executado.

Exemplos:

```
...
if(a - 10) b = 0;
...
if(a) b = 1; else b = 2;
...
```

Operadores Relacionais

Os operadores relacionais permitem a comparação de valores. São os seguintes:

```
<     "menor que"
<=    "menor ou igual a"
==    "igual a"
!=    "diferente de"
>=    "maior ou igual a"
>     "maior que"
```

Exemplos:

```
...
a < (h+5)
...
a <= h
...
x == 10
...
if ( a > 200) printf("altura acima do limite\n");
...
if ( a < 40) printf("altura abaixo do limite\n");
...
```

Esses operadores têm todos a mesma precedência, que é menor que a precedência dos operadores aritméticos. O resultado de uma comparação será um valor igual a zero de a condição testada resultar em *falso* ou um valor diferente de zero se a condição testada resultar em *verdadeiro*. O valor diferente de zero indicando *verdadeiro* não é especificado na definição da linguagem.

Mais exemplos

```
...
if(a != b) a = a-b; else b = b - a;
...
if(a != b) a -= b; else b -= a;
...
if(x == 10) y = 1;
else if (x == 11) y = 2;
     else y = 0;
...
```

Comando Composto

Em algumas situações é necessário tratar uma sequência de comandos como um único comando. Isso é

possível através de '{' e '}', que delimitam um *comando composto*. Exemplo:

```
...
if(a < b) { t = a; a = b; b = t; }
else i = i+1;
...
```

O Comando *while*

O comando *while* tem a forma

```
while( condição ) comando
```

e indica que *comando* deve ser executado repetidamente enquanto *condição* tiver tiver um valor diferente de zero (*verdadeiro*). A condição é calculada *antes* da execução de comando e recalculada antes de cada repetição.

Exemplo

```
#include <stdio.h>

/* calculo da soma dos números de 1 a 10 */

int main()
{
  int s,i;

  s = 0; i = 1;
  while(i <= 10) { s = s + i; i = i+1; }

  printf("A soma dos inteiros de 1 a %d é %d\n",i,s);
}
```

O comando *while* do exemplo acima poderia ser escrito de forma mais compacta:

```
while(i <= 10) s += i++;
```

Neste exemplo, o *comando* que se repete é um *comando composto*. Qualquer comando da linguagem poderia ser utilizado nesse contexto (até mesmo um comando *while*).

Outro exemplo: máximo divisor comum

```
#include <stdio.h>

/* cálculo do máximo divisor comum de dois valores lidos */

int main()
{
   int A,B, a,b;

   /* leitura dos dois valores A e B */
   scanf("%d %d",A,B);
   a = B; b = B;
```

```
    while(a!=b) if(a > b) a -= b; else b -= a;

    printf("O mdc entre %d e %d é: %d\n",A,B,a);
}
```

Neste exemplo, os nomes 'A' e 'a' representam nomes diferentes, já que em C, os caracteres maiúsculos são diferentes dos minúsculos ("Nome" e "nome" são identificadores diferentes).

Ao executar *while(a!=b)...*, se a condição testada for falsa na primeira vez, o comando *if(a>b) a -= b; else b -= a;* não será executado nenhuma vez. Isso vai acontecer se os valores lidos para 'A' e 'B' forem iguais.

O comando *for*

O comando *for* é outro tipo de comando repetitivo, com uma estrutura um pouco mais elaborada. Sua forma geral é a seguinte:

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

onde

- *expressão1* é calculada uma vez antes do início da execução
- *expressão2* indica a condição de repetição
- *expressão3* é executada a cada passo, depois de *comando*

O comando *for*, como mostrado acima é equivalente a um comando *while* da forma:

```
expressão1;
while( expressão2 )
{
  comando
  expressão3;
}
```

O termo *expressão1* está sendo utilizado acima em lugar de *comando*. A rigor um comando em C é uma expressão, que devolve um valor. Essa característica ainda não está sendo utilizada neste texto.

Exemplo

```
#include <stdio.h>

/* calculo da soma dos números de 1 a 10 */

int main()
{
  int s,i;

  s = 0;
  for(i=1; i <= 10; ++i) s+=i;

  printf("A soma dos inteiros de 1 a 10 é %d\n",i,s);
}
```

Esta forma de uso do comando *for*, na qual uma variável é usada como contador do número de vezes que o comando é repetido ocorre com frequência na prática.

Constantes simbólicas

Em C é possível definir constantes usadas com frequência num programa através da diretiva *define*. Sua forma geral é

```
#define nome valor
```

Após a sua definição, a constante pode ser usada no programa através do seu nome. Esse recurso torna os programas mais legíveis e se bem utilizado, facilita a sua manutenção. Exemplo:

```
#include <stdio.h>

/* calculo da soma dos números de 1 a N */

# define N 10

int main()
{
  int s,i;

   s = 0;
   for(i=1; i <= N; ++i) s+=i;

   printf("A soma dos inteiros de 1 a %d é %d\n",N,s);
}
```

[proxima]