Pascal, basta utilizar a função ord que recebe um argumento de um tipo escalar qualquer e retorna o número ordinal dentro do tipo (por exemplo, ord (true) é 1 desde que o tipo boolean seja definido como (false, true)).

Várias funções de transformação têm sido estudadas (Knott, 1975; Knuth, 1973). Um dos métodos que funciona muito bem é o que utiliza o resto da divisão por M[5]:

$$h(K) = K \bmod M,$$

no qual K é um inteiro correspondente à chave, obtido mediante uma soma envolvendo um conjunto de pesos p:

$$K = \sum_{i=1}^{n} \text{Chave}[i] \times p[i],$$

em que n é o número de caracteres da chave, Chave[i] corresponde à representação ASCII do i-ésimo caractere da chave, e $p[i]$ é um inteiro de um conjunto de pesos gerados randomicamente para $1 \leq i \leq n$. A vantagem de usar pesos é que dois conjuntos diferentes de pesos $p_1[i]$ e $p_2[i]$, $1 \leq i \leq n$, levam a duas funções de transformação $h_1(K)$ e $h_2(K)$ diferentes. O Programa 5.22 gera um peso para cada caractere de uma chave constituída de n caracteres.

Programa 5.22 *Geração de pesos para a função de transformação*

```
type TipoPesos = array [1..N] of integer;

procedure GeraPesos (var p: TipoPesos);
var i: integer;
begin
  randomize;
  for i:= 1 to N do
    p[i] := trunc (1000000 * random + 1);
end;
```

Este é um método muito simples de ser implementado, conforme ilustra o Programa 5.23. O único cuidado a tomar é na escolha do valor de M. Por exemplo, se M é par, então $h(K)$ é par quando K é par, e $h(K)$ é ímpar quando K é ímpar. Resumindo, M deve ser um número primo, mas não qualquer primo: devem ser evitados os números primos obtidos a partir de

$$b^i \pm j,$$

[5]Para números reais x e y, a operação binária mod é definida como $x \bmod y = x - y\lfloor x/y \rfloor$, se $y \neq 0$. Quando x e y são inteiros, então $5 \bmod 3 = 2$, $6 \bmod 3 = 0$.

em que b é a base do conjunto de caracteres (geralmente $b = 64$ para BCD, 128 para ASCII, 256 para EBCDIC, ou 100 para alguns códigos decimais), e i e j são pequenos inteiros (Knuth, 1973, p. 509).

Programa 5.23 *Implementação da função de transformação*

```
type TipoChave   = packed array [1..N] of char;
     TipoIndice  = 0..M - 1;
function h (Chave: TipoChave; p: TipoPesos): TipoIndice;
var i, Soma: integer;
begin
  Soma := 0;
  for i := 1 to N do Soma := Soma + ord (Chave[i]) * p[i];
  h := Soma mod M;
end; { h }
```

A seguir vamos apresentar uma modificação no cálculo da função h do Programa 5.23 para evitar a multiplicação da representação ASCII de cada caractere pelo peso. Esta proposta foi apresentada por Zobrist (1990). Este é um caso típico de **troca de espaço por tempo**. Para isso nós vamos gerar randomicamente um peso diferente para cada um dos 256 caracteres ASCII possíveis na $i-$ésima posição da chave, para $1 \leq i \leq n$. O Programa 5.24 apresenta a geração de pesos para a nova função.

Programa 5.24 *Geração de pesos para a função de transformação h de Zobrist*

```
const TAMALFABETO = 256;
type TipoPesos = array[1..N, 1..TAMALFABETO] of integer;

procedure GeraPesos (var p: TipoPesos);
var i, j: integer;
begin
  randomize;
  for i:= 1 to N do
    for j := 1 to TAMALFABETO do
      p[i, j] := trunc(1000000 * random + 1);
end; {GeraPesos}
```

O Programa 5.25 mostra a implementação da função *hash* de Zobrist. Para obter h é necessário o mesmo número de adições da função gerada pelo Programa 5.23, mas nenhuma multiplicação é efetuada. Isto faz com que h seja computada de forma mais eficiente. Nesse caso, a quantidade de espaço para armazenar h é $O(n \times |\Sigma|)$, onde $|\Sigma|$ representa o tamanho do alfabeto, enquanto que, para a função do Programa 5.23, é $O(n)$.