5章　意味解析

変数名や関数名などの識別子を**記号表**に登録し、型の一貫性の検査を行います。

5.1　記号表

原始プログラムに現れる識別子（変数名や関数名など）をそれがもつ様々な属性（変数や関数の別、型、内部変数か外部変数の別など）とともに保持する表を**記号表**といいます。原始プログラムの中で識別子が宣言されるとコンパイラはこれを記号表に登録し、原始プログラムの中で識別子が参照されると、コンパイラはその情報を記号表から探索します。

5.2　記号表の内容

記号表がもつ情報は個々のプログラミング言語やコンパイラに依存しますが、一般的に次のようなものを持ちます。

（1）識別子名

（2）識別子の種別（変数、関数、定数などの別）

（3）種別毎の属性

（変数の場合）	（関数の場合）	（定数の場合）
・型	・仮引数の個数と型	・型
・バイト数	・戻り値の型	・値のある番地など
・内部と外部の別	・コードの番地など	
・通常の変数か仮引数か		
・変数のある番地など		

例えば、int num; の場合

num，変数，int，4，内部，0x00f2，～

また、void show(int i); の場合

show，関数，1，int，void，0x0fe，～

5.3　記号表のデータ構造

記号表に対する操作として、新しい識別子の**登録**と、既存の識別子の**探索**があります。識別子の登録を行う場合も、同じ名前がすでに登録されているかをみるために探索が行われることが多いです。識別子の探索は、識別子が宣言される場合も使用される場合も含めて、原始プログラムの中に出現するたびに行われるので、効率が良いことが望ましいです。

記号表の主な実現方法には、**配列**、**木構造**、**ハッシュ法**があります。コンパイラでは、データの探索時間の比較的短いハッシュ法がよく用いられます。

配列を用いた記号表

識別子の登録は、識別子が宣言される順番に並べて登録します。識別子の宣言の順番がそのまま記号表の識別子の順番となります。ただし、新しい識別子を登録するときそれがすでに記号表にあるかを探索し、もしなければそれを記号表に登録します。

識別子の探索は、記号表の先頭から順番に調べていく**順次（線形）探索**を用います。見つかれば**探索成功**です。終端まで来たら**登録無し**となります。探索回数は、n 個の識別子が記号表に登録されていたとして、識別子 x を記号表から探索するとき、

・もし、識別子 x が記号表に登録されていなかったとき　→ n 回の探索が必要です

・もし、識別子 x が記号表に登録されていたとき　　→ 平均 (n+1)/2 回の探索が必要です

（図）

木構造（2分木）を用いた記号表

識別子の登録は、まず探索により登録したい識別子 x がまだ登録されていないことを調べます。もし、登録されていなければ、最後に大小比較を行った識別子より小さければ左側へ、大きければ右側へ識別子 x を表す節（ノード）を加えます。

識別子の探索は、識別子の大小比較によって次に探索する場所を決めていきます。まず、根（ルート）の識別子と大小比較します。もし、それより小さければ左側の部分木に、それより大きければ右側の部分木に移動します。もし、等しければ**探索成功**です。次に、部分木の根の識別子と大小比較します。これを繰り返します。葉まで来たとき、等しいものが無ければ、**登録無し**となります。探索回数は、ほぼ $[\log_2 n] + 1$ となります。$[x]$は床関数であり、xを超えない最大の整数です。

（図）

ハッシュ法を用いた記号表

ハッシュ法　識別子からデータが格納されている（又は、データを格納する）場所を関数によって求める方法です。この関数を**ハッシュ関数**といいます。ハッシュ法により探索の時間を大幅に減らすことができます。

（図）

ハッシュ関数　識別子 x（文字コードなど）からその属性データを格納するハッシュ表の添え字（位置）を返す関数 h(x)です。**ハッシュ関数**により得られる値を**ハッシュ値**といいます。

識別子の登録は、まず探索により登録した識別子 x が登録されていないことを調べます。もし、登録されていなければ、なにも無かったその場所へ識別子 x を登録します。

識別子の探索は、識別子 x とハッシュ関数 h(x)により、探索する場所を決めます。まず、識別子 x からハッシュ値 n = h(x)を求めます。添え字 n でハッシュ表を調べ、

(a)そこに識別子 x があったら、**探索成功**です。

(b)そこになにも無かったら、**登録無し**です。

(c)そこに別の識別子 y があったら、**衝突**です。

衝突の場合は、新しい候補（ハッシュ値）を決め、(a)または(b)になるまで繰り返します。

衝　　突　異なる識別子 x と y に対してそのハッシュ値が同じ、すなわち h(x)==h(y)となった場合を**衝突**といいます。衝突が起きた場合は新しい候補（ハッシュ値）を決める必要があります。よく用いられる方法に**線形走査法**があります。この他、衝突したものをポインタで繋いでいく**連鎖法**があります。

線形走査法　識別子 x に対して、h(x)に既にデータがある場合は h(x)のすぐ次の空いている場所を次の探索または登録の候補とします。n 個の識別子がサイズ M のハッシュ表に登録されているとして、識別子が登録されている場合の平均探索回数は、

$\left(1-\frac{\alpha}{2}\right)/(1-\alpha)$　ここで、$\alpha=\frac{n}{M}$（ハッシュ表占有率）です

（例）

ハッシュ関数のもつべき特徴は一般的に次の2点です。
・計算しやすいこと
・いろいろな識別子に対してできるだけ一様なハッシュ値をもつこと

ハッシュ法ではどのようにハッシュ関数を決めるかが重要となります。具体的なハッシュ関数として次の3つがあります。

1. **除算法**
2. **平方採中法**
3. **折り返し法**

除算法　識別子 x をハッシュ表のサイズ M で割った余りをハッシュ値とします。
h(x) = x % M

平方採中法　識別子 x の値を二乗して、その結果の中ほどのビット列を取り出して
ハッシュ値とします。
h(x) = int(x^2 / 2^m) % M
但し、x^2 ≒ 2^(2m+n)、M=2^n です。
ここで、M はハッシュ表のサイズです。

折り返し法　識別子 x の各桁または各部分桁同士を加えたり、排他的論理和を
とったりしてハッシュ値とします。