

Bab 5

Symbol Table dan Scope Management

Sub-CPMK yang Dicakup dalam Bab Ini:

- **Sub-CPMK 3.1:** Mengimplementasikan symbol table dengan nested scopes
- **Sub-CPMK 3.2:** Melakukan type checking untuk ekspresi kompleks

5.1 Dasar Symbol Table dan Perannya

Symbol Table adalah struktur data fundamental yang menyimpan informasi tentang identifiers (variabel, fungsi, tipe) yang muncul dalam kode sumber.

5.1.1 Kebutuhan Symbol Table

Dalam fase analisis semantik, kompilator perlu menjawab pertanyaan:

- Apakah variabel x sudah dideklarasikan?
- Apa tipe data dari y ?
- Apakah z merupakan variabel lokal atau global?

5.1.2 Komponen Entry

Setiap entri dalam symbol table minimal menyimpan:

- **Name:** Nama identifier.
- **Type:** Tipe data (misal: `int`, `float`).
- **Scope:** Level nesting tempat identifier berada.
- **Location:** Alamat memori atau offset (untuk fase *code generation*).

5.2 Struktur Data: Hash Table dan Scope Stack

Agar operasi pencarian (*lookup*) berjalan efisien, symbol table biasanya diimplementasikan menggunakan **Hash Table**.

5.2.1 Implementasi Hash Table

Dalam C++, kita dapat menggunakan `std::unordered_map` untuk memetakan nama identifier ke objek `Symbol`.

```
1 struct Symbol {
2     string name;
3     string type;
4     int line;
5 };
6
7 class Scope {
8     unordered_map<string, Symbol*> table;
9     Scope* parent;
10 public:
11     Symbol* lookup(string name) {
12         if (table.count(name)) return table[name];
13         if (parent) return parent->lookup(name);
14         return nullptr;
15     }
16 };
```

5.2.2 Manajemen Scope Stack

Setiap kali kompilator menemukan blok baru (`{}`), scope baru dibuat dan ditumpuk di atas scope sebelumnya. Saat keluar dari blok (`}`), scope teratas akan dihapus atau dinonaktifkan.

5.3 Shadowing dan Resolusi Nama

5.3.1 Shadowing

Shadowing terjadi ketika sebuah identifier di scope dalam memiliki nama yang sama dengan identifier di scope luar. Dalam kasus ini, referensi ke nama tersebut akan merujuk pada deklarasi yang paling dekat (paling dalam).

5.3.2 Resolusi Nama (Name Resolution)

Proses ini mencocokkan setiap penggunaan nama variabel dengan deklarasi yang tepat. Algoritma lookup akan mencari secara bertahap:

1. Cari di scope saat ini.

2. Jika tidak ada, naik ke parent scope.
3. Ulangi sampai ke scope global.
4. Jika tetap tidak ditemukan, laporkan *Undeclared Identifier Error*.



Gambar 5.1: Alur resolusi nama pada nested scopes

5.4 Penanganan Scope Entry dan Exit

5.4.1 Function Scope

Saat mendeklarasikan fungsi, parser harus membuat scope baru untuk menampung parameter fungsi dan variabel lokal di dalamnya.

5.4.2 Block Scope

Setiap blok kode yang dibatasi kurung kurawal menciptakan scope sementara. Kompilator memanggil `beginScope()` saat menemukan `{` dan `endScope()` saat menemukan `}`.

5.4.3 Contoh Kasus

```

1 int x = 1; // Global
2 void f() {
3     int x = 2; // Shadows global x
4     {
5         int x = 3; // Shadows f's x
6     }
7 }
  
```

Symbol table akan mengelola tiga versi variabel `x` tersebut pada level nesting yang berbeda.

5.5 Implementasi Symbol Table yang Lengkap

Implementasi yang robust memerlukan manajemen memori yang baik untuk setiap entri simbol.

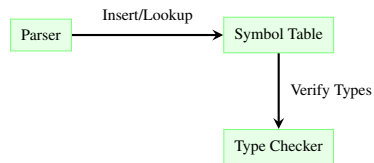
```

1 class SymbolTable {
2     Scope* current;
3 public:
4     void beginScope() { current = new Scope(current); }
5     void endScope() {
6         Scope* old = current;
  
```

```

7      current = current->parent;
8      delete old;
9  }
10     bool insert(string name, Symbol* s) {
11         return current->add(name, s);
12     }
13     Symbol* lookup(string name) {
14         return current->find(name);
15     }
16 };

```



Gambar 5.2: Interaksi Symbol Table dalam integrasi sistem

Aktivitas Pembelajaran

1. **Hash Table Implementation:** Implementasikan symbol table dengan hash table dan chaining.
2. **Scope Testing:** Buat program dengan nested scopes dan test symbol table operations.
3. **Type Checker:** Implementasikan type checker untuk ekspresi aritmatika dan assignment.
4. **Debug Visualization:** Buat tool untuk visualisasi symbol table dan scope hierarchy.
5. **Performance Analysis:** Bandingkan berbagai hash functions untuk symbol table.

Latihan dan Refleksi

1. Implementasikan symbol table with support for nested scopes menggunakan hash table!
2. Buat type checker untuk bahasa dengan int, float, dan boolean types!
3. Analisis complexity dari symbol table operations (insert, lookup, delete)!
4. Implementasikan scope stack untuk bahasa dengan function definitions!

5. Desain error reporting system untuk type errors!
6. **Refleksi:** Bagian mana dari symbol table implementation yang paling menantang?

Asesmen (Evaluasi Kinerja)

Instrumen Penilaian untuk Sub-CPMK 3.1-3.2

A. Pilihan Ganda

1. Data structure yang PALING cocok untuk symbol table adalah:
 - (a) Linked list
 - (b) Array
 - (c) Hash table
 - (d) Binary tree
2. Scope management menggunakan:
 - (a) Queue
 - (b) Stack
 - (c) Priority queue
 - (d) Heap
3. Type checking memastikan:
 - (a) Syntax correctness
 - (b) Type compatibility
 - (c) Runtime efficiency
 - (d) Code optimization

B. Essay

1. Jelaskan implementasi symbol table dengan nested scopes dan berikan contoh kode!
2. Desain type system untuk bahasa dengan arrays, functions, dan pointers!

Rubrik Penilaian: Lihat Lampiran A

Checklist Pencapaian Kompetensi

Centang item berikut setelah Anda yakin telah menguasainya:

- ☐ Saya dapat mengimplementasikan symbol table dengan nested scopes
- ☐ Saya dapat melakukan type checking untuk ekspresi kompleks
- ☐ Saya memahami berbagai implementasi symbol table
- ☐ Saya dapat mengelola scope dengan stack structure
- ☐ Saya dapat mendesain type system yang sederhana
- ☐ Saya dapat mengimplementasikan type inference algorithm

Rangkuman

Bab ini membahas symbol table dan scope management, termasuk implementasi hash table, nested scopes, type system, dan type checking. Mahasiswa belajar membangun data structure fundamental untuk semantic analysis.

Poin Kunci:

- Symbol table menyimpan informasi identifiers dengan akses cepat
- Nested scopes dikelola menggunakan stack structure
- Type checking memastikan type compatibility dalam expressions
- Hash table adalah implementasi efisien untuk symbol table
- Type system adalah fondasi untuk semantic analysis

Kata Kunci: *Symbol Table, Scope Management, Type System, Type Checking, Hash Table, Nested Scopes, Type Inference*