BITS Pilani, Hyderabad Campus
Department of Computer Science and Information Systems
Second Semester, 2024-25
CS F363 Compiler Construction
Lab-10: Symbol table

1 Introduction

A symbol table is a crucial data structure used in compilers and interpreters. It stores information about identifiers (variables, functions, etc.) used in a program, such as their names, types, scopes, and memory locations. This lab will guide you through building a basic symbol table using these tools.

2 Objectives

- Implement a symbol table using a hash table with chaining for efficient identifier storage and retrieval.
- Extend a Bison parser to handle C-like declarations and assignments while maintaining the symbol table.
- Detect semantic errors such as undeclared variable usage and redefinition errors during parsing.

3 Data Structures for Symbol Table

3.1 Symbol Table Entry

Each entry in the symbol table represents an identifier.

```
typedef struct Symbol {
    char* name;
    char* type;
    struct Symbol* next;
} Symbol;
```

3.2 Hash Table

A hash table is used to store symbols for fast retrieval.

```
#define HASH_TABLE_SIZE 101

typedef struct HashTable {
    Symbol* table[HASH_TABLE_SIZE];
} HashTable;
```

- This defines a hash table to store symbols (identifiers like variables, functions, etc.).
- `HASH_TABLE_SIZE` is set to 101 (a prime number, which helps distribute keys more evenly).
- The hash table is implemented as an array of linked lists, where each index stores a pointer to a Symbol.

3.3 Symbol Table Stack (Linked List of Hash Tables)

```
typedef struct ScopeNode {
    HashTable* hashTable;
    struct ScopeNode* next;
} ScopeNode;

ScopeNode* symbolTableStack = NULL;
int currentScope = 0;
```

- A linked list of hash tables is used to implement scoping.
- Each ScopeNode contains:
  - hashTable: A pointer to a hash table (for the current scope).
  - next: A pointer to the previous scope's hash table.
- symbolTableStack is a stack pointer that keeps track of the current scope.
- currentScope keeps track of the scope level (0 = global, 1 = first function block, etc.).

4 Functions

Some important function definitions are given below (you can change them according to your needs).

```
int hash(char* name) {
    unsigned long hash = 5381;
    int c;
    while ((c = *name++))
        hash = ((hash << 5) + hash) + c;
    return hash % HASH_TABLE_SIZE;
}

void insertSymbol(char* name, char* type) {
    HashTable* currentTable = symbolTableStack->hashTable;
    int index = hash(name);
    Symbol* newSymbol = (Symbol*)malloc(sizeof(Symbol));
    newSymbol->name = strdup(name);
    newSymbol->type = strdup(type);
    newSymbol->next = currentTable->table[index];
    currentTable->table[index] = newSymbol;
}
```

```
void printSymbolTable() {
    if (symbolTableStack == NULL || symbolTableStack->hashTable == NULL) {
        printf("\nSymbol table is empty!\n");
        return;
    }

    printf("\n================ Symbol Table ================\n");
    printf("| %-15s | %-10s |\n", "Name", "Type");
    printf("---------------------------------------------\n");

    HashTable* globalTable = symbolTableStack->hashTable; // Access only the
        top-level table
    for (int i = 0; i < HASH_TABLE_SIZE; i++) {
        Symbol* currentSymbol = globalTable->table[i];
        while (currentSymbol != NULL) {
            printf("| %-15s | %-10s |\n",
                   currentSymbol->name,
                   currentSymbol->type);
            currentSymbol = currentSymbol->next;
        }
    }
    printf("=============================================\n");
}
```

5 Exercises

Task 1 Write a bison code that parses declaration statements as in C language and prints the symbol table after the parsing.

For the sake of simplicity, proceed as follows:

(a) First, consider the declaration statements without assigning a value or expression to a variable. For example:

```
int x, y, z;
float avg, cgpa;
char Grade;
```

The expected output is as follows:

```
================ Symbol Table ================
| Name           | Type     |
---------------------------------------------
| x              | int      |
| y              | int      |
| z              | int      |
| avg            | float    |
| cgpa           | float    |
| Grade          | char     |
=============================================
```

(b) Extend your bison code to handle the assignment statements with constants. In this case, your symbol table should have a column `value` that contains the current value of the variable.

For example:

```
int x, y=10, z;
float avg, cgpa=7.5;
char Grade='B';
```

The expected output is as follows:

```
==================== Symbol Table ====================
| Name           | Type     | Value    |
------------------------------------------------------
| x              | int      | (null)   |
| y              | int      | 10       |
| z              | int      | (null)   |
| avg            | float    | (null)   |
| cgpa           | float    | 7.5      |
| Grade          | char     | 'B'      |
======================================================
```

(c) Further, extend your code to handle the assignment statements with arithmetic expressions. For example:

```
int x=5, y=10, z=x+2*5;
float avg, cgpa=7.5;
char Grade='B';
```

The expected output is as follows:

```
==================== Symbol Table ====================
| Name   | Type   | Value   |
----------------------------------
| x      | int    | 5       |
| y      | int    | 10      |
| z      | int    | 15      |
| avg    | float  | (null)  |
| cgpa   | float  | 7.5     |
| Grade  | char   | 'B'     |
======================================================
```

Task 2 (a) Extend your code to identify if a variable is used before declaration in the input (ignore the code after the error). For example:

```
int x;
x = 10;
float y;
y = x + 5.5;
z = 20;
x=10+y;
```

The expected output is as follows:

```
Error: Variable 'z' used before declaration.

==================== Symbol Table ====================
| Name  | Type   | Value |
---------------------------
| x     | int    | 10    |
| y     | float  | 15.5  |
======================================================
```

(b) Extend your code to identify if a variable is redefined in the input. For example:

```
int x;
x = 10;
float y;
int x=20;
z = 20;
```

The expected output is as follows:

```
Error: Variable 'x' is already declared.
Error: Variable 'z' used before declaration.

==================== Symbol Table ====================
| Name  | Type   | Value  |
----------------------------
| x     | int    | 10     |
| y     | float  | (null) |
======================================================
```