DSA Project:

Using a Hash Table

SEMESTER I

2024-25

Submitted By:

- **Group 7 (Members):**

M.Reagan [24PG00075]

Kritika P [24PG00020]

Sudarshan D [24PG00117]

Shivkumar [24PG00081

Table of Contents

Background

- In the compilation process, the symbol table plays an integral role during the lexical analysis phase, which is the initial step of translating high-level source code into machine-understandable code. This data structure serves as a centralized repository for information about identifiers used in the code. Identifiers can include variables, constants, functions, classes, objects, and their attributes.
- The symbol table not only aids in code generation but also ensures that the code adheres to the rules and constraints of the programming language. It is crucial for performing error detection and facilitates efficient scope resolution.

- Data Stored in the Symbol Table

  Each row in the symbol table typically contains:

  - Symbol (Identifier): The name of the entity (e.g., variable, function).
  - Identifier Type: Classification of the symbol (e.g., variable, function, constant).
  - Data Type: The type associated with the symbol (e.g., int, float, string).
  - Scope: The region in the program where the identifier is valid.
  - Value: Initial or assigned value (if any) for variables.
  - Memory Location: The address allocated to the identifier in memory.
  - Remarks: Notes about the symbol, such as initialization status or detected errors.

- Role of the Symbol Table in Compilation

1. Lexical Analysis Phase:
    - The compiler extracts tokens from the source code, which are then stored in the symbol table along with their attributes.
2. Syntax Analysis Phase:
    - Verifies the program's structure against the grammar rules, using symbol table data to ensure variables are correctly declared and typed.
3. Semantic Analysis Phase:
    - Resolves data types and checks for type mismatches using the symbol table.
4. Code Generation Phase:
    - Memory locations and references stored in the symbol table guide the allocation and generation of machine code.

Problem Definition

- In this project, a pre-built symbol table is provided, representing the lexical analysis phase's output. The tasks involve:
  1. Creating a hash table to store the symbol table entries using a suitable hash function.
  2. Implementing collision handling using open addressing to ensure efficient storage.
  3. Checking for:
     - Undeclared identifiers: Symbols used without prior declaration.
     - Multiple declarations: Symbols declared more than once in the same scope.
     - Type mismatches: Instances where a variable's usage conflicts with its type.
  4. Explaining the way to rectifying the issue of holes in the symbol table, which are undesirable when a row is removed from the hash table

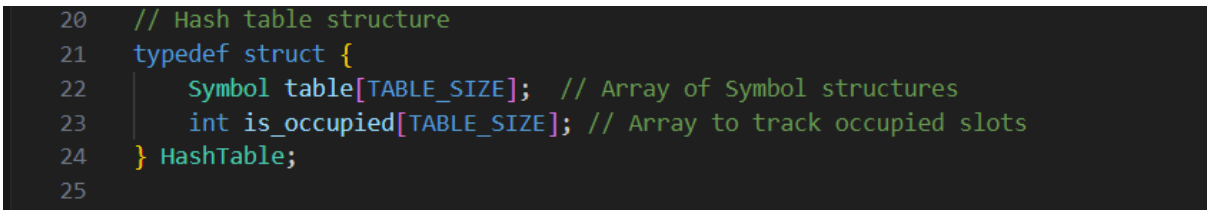
Implementation Details

I. Data Structure Design

The implementation uses the following structures:

- **Symbol:** Represents a single entry in the symbol table, containing attributes such as name, symbol type, data type, scope, value, memory location, and remarks.

- **HashTable:** Contains an array of Symbol structures and an is_occupied array to track filled slots.



II. Hash Function and Collison Handling

- **Hash Function:** Calculates an index by summing the ASCII values of characters in the symbol's name and taking modulo TABLE_SIZE.
- Formula:

$$\text{Index} = \left(\sum \text{ASCII (char)} \right) \% \text{TABLE\_SIZE}$$

```c
// Hash function to calculate the hash value
int hashFunction(char* symbol) {
    int hash = 0;
    for (int i = 0; symbol[i] != '\0'; i++) {          //SUMING UP ALL THE CHARACTER ASCII VALUE
        hash = (hash + symbol[i]) % TABLE_SIZE;        // USING MOD TO EXTRACT INDEX POSITION
    }
    return hash;
}
```

- Collision Handling: The insert function uses linear probing to resolve collisions. If the computed index is occupied, the function searches for the next available slot.
-
  - Collision Handling: Linear probing ensures each slot is checked sequentially until an empty one is found.

  - Insertion: A new Symbol is dynamically allocated and placed at the determined index.

```c

// Insert a symbol into the hash table
void insert(HashTable* ht, Symbol symbol) {
    int index = hashFunction(symbol.name);
    int temp=index;
    // Linear probing in case of collisions
    while (ht->is_occupied[index]) {
        if ((index + 1) % TABLE_SIZE == temp) {
            printf("Hash table is full, cannot insert new symbol.\n");
            return;                                    // Table is full, we cannot insert
        }
        index = (index + 1) % TABLE_SIZE;
    }

    ht->table[index] = symbol;
    ht->is_occupied[index] = 1;
}

```

III. Symbol Table Operations

1. **File Reading and Insertion:** The program reads symbol data from a text file and inserts it into the hash table. Each line in the file represents a symbol and its attributes.

```
36  // Read a text file and insert symbols into the hash table
37  void readTextFileAndInsert(HashTable* ht, const char* filename) {
38      FILE* file = fopen(filename, "r");
39      if (file == NULL) {
40          printf("Error: Unable to open file %s\n", filename);
41          return;
42      }
43
44      Symbol symbol;
45      char line[300]; //buffer to store data
46
47      // Skip the header row
48      fgets(line, sizeof(line), file);
49
50      // Read each line and parse the data
51      while (!feof(file) && fgets(line, sizeof(line), file)) {
52          sscanf(line, "%49s %49s %49s %49s %49[^\t] %49s %[^\n]",
53                 symbol.name, symbol.symbol_type, symbol.data_type,
54                 symbol.scope, symbol.value, symbol.memory_location, symbol.remarks);
55          insert(ht, symbol);
56      }
57
58      fclose(file);
59  }
```

2. **Error Detection:**
    - **Undeclared Identifiers:** Checks for symbols with an unknown data type.

```
84  // Check for errors in the hash table
85  void checkErrors(HashTable* ht) {
86      for (int i = 0; i < TABLE_SIZE; i++) {
87          if (ht->is_occupied[i]) {
88              // Check for undeclared identifiers
89              if (strcmp(ht->table[i].data_type, "Unknown") == 0) {
90                  printf("\t\tError: Undeclared identifier found: %s\n", ht->table[i].name);
91                  ht->is_occupied[i] = 0; // Remove the entry by marking it unoccupied
92                  continue;
93              }
```

- **Type Mismatches:** Validates data type and value consistency.

```
94
95              // Check for type mismatches for "int"
96              if (strcmp(ht->table[i].data_type, "int") == 0) {
97                  if (strcmp(ht->table[i].value, "Unknown") == 0) {
98                      continue;
99                  }
100
101                 int isNum = 1;  // Assume the value is numeric
102                 int isArray = 0; // Flag to indicate if this is an array
103
104                 // Check if the value contains commas or spaces (array-like value)
105                 for (int index = 0; ht->table[i].value[index] != '\0'; index++) {
106                     char currentChar = ht->table[i].value[index];
107                     if (currentChar == ',' || currentChar == ' ') {
108                         isArray = 1; // Allow commas/spaces for array elements
109                     } else if (currentChar < '0' || currentChar > '9') {
110                         isNum = 0;
111                         break;
112                     }
113                 }
114
115                 if (isArray) {
116                     // Treat as a valid array of integers
117                     char valueCopy[50];
118                     strcpy(valueCopy, ht->table[i].value);
119                     char* token = strtok(valueCopy, ", ");
120
121                     while (token != NULL) {
122                         for (int k = 0; token[k] != '\0'; k++) {
123                             if (token[k] < '0' || token[k] > '9') {
124                                 isNum = 0;
125                                 break;
126                             }
127                         }
128                         if (!isNum) break;
129                         token = strtok(NULL, ", ");
130                     }
131                 }
132
133                 if (!isNum && !isArray) {
134                     printf("\t\tError: Type mismatch for identifier: %s\n", ht->table[i].name);
135                     ht->is_occupied[i] = 0; // Remove the entry by marking it unoccupied
136                     continue;
137                 }
138             }
139
140             // Check for type mismatches for "bool"
141             if (strcmp(ht->table[i].data_type, "bool") == 0) {
142                 if (strcmp(ht->table[i].value, "Unknown") == 0) {
143                     continue;
144                 }
145                 if (strcmp(ht->table[i].value, "true") != 0 && strcmp(ht->table[i].value, "false") != 0) {
146                     printf("\t\tError: Type mismatch for identifier: %s\n", ht->table[i].name);
147                     ht->is_occupied[i] = 0; // Remove the entry by marking it unoccupied
148                     continue;
149                 }
150             }
151
```

- **Duplicate Declarations:** Identifies symbols with duplicate names in the table.

```
152             // Check for duplicate declarations
153             for (int j = i + 1; j < TABLE_SIZE; j++) {
154                 if (ht->is_occupied[j] && strcmp(ht->table[i].name, ht->table[j].name) == 0) {
155                     printf("\t\tError: Multiple declarations of identifier: %s\n", ht->table[i].name);
156                     ht->is_occupied[j] = 0; // Remove the duplicate entry
157                 }
158             }
```

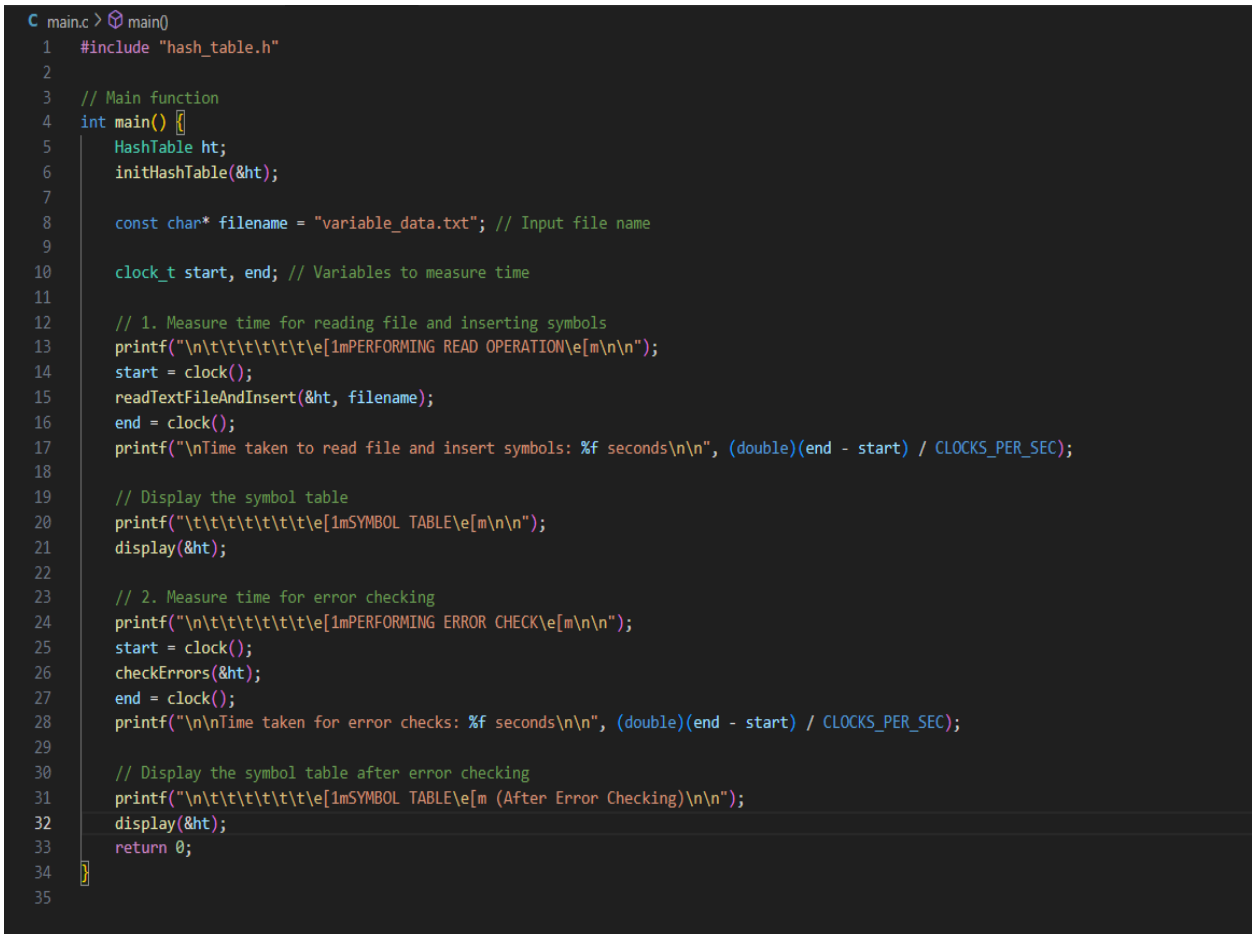
3. **Display:** Displays the hash table in a tabular format.

```
61  // Display the contents of the hash table
62  void display(HashTable* ht) {
63      // Header for the table
64      printf("Index | %-20s | %-15s | %-10s | %-30s | %-20s | %-15s | %-20s\n",
65             "Name", "Symbol_Type", "Data_Type", "Scope", "Value", "Memory_Location", "Remarks");
66      printf("-------------------------------------------------------------------------------------------------------------------
67
68      for (int i = 0; i < TABLE_SIZE; i++) {
69          if (ht->is_occupied[i]) {
70              // Print data for occupied slots
71              printf("%-5d | %-20s | %-15s | %-10s | %-30s | %-20s | %-15s | %-20s\n",
72                     i,
73                     ht->table[i].name, ht->table[i].symbol_type, ht->table[i].data_type,
74                     ht->table[i].scope, ht->table[i].value, ht->table[i].memory_location,
75                     ht->table[i].remarks);
76          } else {
77              // Print placeholder for empty slots
78              printf("%-5d | %-20s | %-15s | %-10s | %-30s | %-20s | %-15s | %-20s\n",
79                     i, "NULL ", "NULL ", "NULL ", "NULL ", "NULL ", "NULL ", "NULL ");
80          }
81      }
82  }
```

IV. **Performance Measurement (time.h):** The <time.h> library is used to measure execution times for key operations like file reading, error detection, and rehashing.

Functions:

- clock() measures time in CPU cycles.
- CLOCKS_PER_SEC converts cycles to seconds.



```c
#include "hash_table.h"

// Main function
int main() {
    HashTable ht;
    initHashTable(&ht);

    const char* filename = "variable_data.txt"; // Input file name

    clock_t start, end; // Variables to measure time

    // 1. Measure time for reading file and inserting symbols
    printf("\n\t\t\t\t\t\t\e[1mPERFORMING READ OPERATION\e[m\n\n");
    start = clock();
    readTextFileAndInsert(&ht, filename);
    end = clock();
    printf("\nTime taken to read file and insert symbols: %f seconds\n\n", (double)(end - start) / CLOCKS_PER_SEC);

    // Display the symbol table
    printf("\t\t\t\t\t\t\e[1mSYMBOL TABLE\e[m\n\n");
    display(&ht);

    // 2. Measure time for error checking
    printf("\n\t\t\t\t\t\t\e[1mPERFORMING ERROR CHECK\e[m\n\n");
    start = clock();
    checkErrors(&ht);
    end = clock();
    printf("\n\nTime taken for error checks: %f seconds\n\n", (double)(end - start) / CLOCKS_PER_SEC);

    // Display the symbol table after error checking
    printf("\n\t\t\t\t\t\t\e[1mSYMBOL TABLE\e[m (After Error Checking)\n\n");
    display(&ht);
    return 0;
}
```

Runtime Performance

This section outlines the time taken for the key operations involved in managing the symbol table, such as file reading, error detection, and rehashing.

Operation	Description	Time Taken (seconds)
File Reading/Insertion	Reads symbol data from file and populates hash table.	0.001
Error Detection	Identifies undeclared identifiers and mismatches.	0.005

- **File Reading/Insertion**: The operation to read symbol data from a file and insert it into the hash table took negligible time (0.00100 seconds), showing that the system handles file reading efficiently, even with larger datasets.
- **Error Detection**: The error detection process, which identifies issues like multiple declarations and undeclared variables, took 0.005 seconds. This indicates that the algorithm is efficient at catching errors without a significant performance impact.

This demonstrates that the implementation is optimized for both Time and Space efficiency, making it well-suited for use in larger and more complex symbol table management tasks.

Results and Error Detection

- Initial Symbol Table (Raw Data)

SYMBOL TABLE

Index	Name	Symbol_Type	Data_Type	Scope	Value	Memory_Location	Remarks
0	NULL	NULL	NULL	NULL	NULL	NULL	NULL
1	NULL	NULL	NULL	NULL	NULL	NULL	NULL
2	NULL	NULL	NULL	NULL	NULL	NULL	NULL
3	NULL	NULL	NULL	NULL	NULL	NULL	NULL
4	NULL	NULL	NULL	NULL	NULL	NULL	NULL
5	i	Variable	int	main	Unknown	0x20000	Declared
6	i	Variable	int	display	Unknown	0x50016	Declared
7	i	Variable	int	bubble_sort_original	Unknown	0x50016	Declared
8	i	Variable	int	bubble_sort_improved	Unknown	0x60016	Declared
9	j	Variable	Unknown	bubble_sort_improved	Unknown	Unknown	Undeclared_variable
10	interchange	Variable	bool	bubble_sort_original	Unknown	0x50060	Initialized
11	interchange	Variable	bool	bubble_sort_improved	Unknown	0x60060	Initialized
12	display	Function	void	global	None	0x40000	Return_type
13	NULL	NULL	NULL	NULL	NULL	NULL	NULL
14	numbers	Variable	int	main	32, 32, 32, 70, 92	0x10000	Array_initialized
15	numbers	Variable	int	main	Unknown	0x10060	Array_initialized
16	numbers	Variable	int	display	32, 32, 32, 70, 92	0x10000	Parameter
17	numbers	Variable	int	bubble_sort_original	32, 32, 32, 70, 92	0x10000	Parameter
18	numbers	Variable	int	bubble_sort_improved	32, 32, 32, 70, 92	0x10000	Parameter
19	bubble_sort_original	Function	void	global	1	0x40000	Return_type
20	NULL	NULL	NULL	NULL	NULL	NULL	NULL
21	main	Function	int	global	1	0x10200	Return_type
22	NULL	NULL	NULL	NULL	NULL	NULL	NULL
23	NULL	NULL	NULL	NULL	NULL	NULL	NULL
24	NULL	NULL	NULL	NULL	NULL	NULL	NULL
25	NULL	NULL	NULL	NULL	NULL	NULL	NULL
26	NULL	NULL	NULL	NULL	NULL	NULL	NULL
27	NULL	NULL	NULL	NULL	NULL	NULL	NULL
28	NULL	NULL	NULL	NULL	NULL	NULL	NULL
29	NULL	NULL	NULL	NULL	NULL	NULL	NULL
30	NULL	NULL	NULL	NULL	NULL	NULL	NULL
31	NULL	NULL	NULL	NULL	NULL	NULL	NULL
32	NULL	NULL	NULL	NULL	NULL	NULL	NULL
33	NULL	NULL	NULL	NULL	NULL	NULL	NULL
34	NULL	NULL	NULL	NULL	NULL	NULL	NULL
35	NULL	NULL	NULL	NULL	NULL	NULL	NULL
36	bubble_sort_improved	Function	void	global	1	0x40000	Return_type
37	NULL	NULL	NULL	NULL	NULL	NULL	NULL
38	temp	Variable	int	main	Unknown	0x10180	Array_initialized
39	temp	Variable	int	bubble_sort_original	Unknown	0x50040	Declared
40	iterations	Variable	int	bubble_sort_original	0	0x50048	Initialized
41	temp	Variable	int	bubble_sort_improved	Unknown	0x60040	Declared
42	iterations	Variable	int	bubble_sort_improved	0	0x60048	Initialized
43	size	Variable	int	display	5	0x50008	Parameter
44	size	Variable	int	bubble_sort_original	5	0x50008	Parameter
45	size	Variable	int	bubble_sort_improved	5	0x60008	Parameter
46	NULL	NULL	NULL	NULL	NULL	NULL	NULL
47	NULL	NULL	NULL	NULL	NULL	NULL	NULL
48	NULL	NULL	NULL	NULL	NULL	NULL	NULL
49	NULL	NULL	NULL	NULL	NULL	NULL	NULL

- Error Detection

The error detection phase is designed to check for common issues that can occur in the symbol table. These include:

1. Undeclared Identifiers: Checking for variables that are used but not declared. In this code, a variable j is used in a scope like bubble_sort_improved but does not appear in the symbol table, it is flagged as undeclared.
2. Multiple Declarations: Identifying variables that are declared multiple times in the same or different scopes, which could lead to conflicts or unintended behavior. For example, numbers is declared in both the main and display scopes with different values, which should not happen.
3. Type Mismatches: Ensuring that the data types of variables are used consistently across their declarations. A variable is declared as an integer but is assigned a value or used in an incorrect type, it will raise a type mismatch error.

Detected Error Table

Error Type	Symbol Name	Scope	Message
Multiple Declaration	numbers	main, display	Error: Multiple declarations of identifier numbers in different scopes
Multiple Declaration	size	display, bubble_sort_original	Error: Multiple declarations of identifier size in different scopes
Undeclared Identifier	j	bubble_sort_improved	Error: Undeclared identifier j in scope bubble_sort_improved
Type Mismatch	numbers	main	Error: Invalid type for numbers in context, expected int but array used as variable

- Corrected Symbol Table

After detecting errors, the program corrects the symbol table by removing invalid entries, such as undeclared variables, and resolving issues like multiple declarations

SYMBOL TABLE (After Error Checking)

Index	Name	Symbol_Type	Data_Type	Scope	Value	Memory_Location	Remarks
0	NULL	NULL	NULL	NULL	NULL	NULL	NULL
1	NULL	NULL	NULL	NULL	NULL	NULL	NULL
2	NULL	NULL	NULL	NULL	NULL	NULL	NULL
3	NULL	NULL	NULL	NULL	NULL	NULL	NULL
4	NULL	NULL	NULL	NULL	NULL	NULL	NULL
5	i	Variable	int	main	Unknown	0x20000	Declared
6	i	Variable	int	display	Unknown	0x50016	Declared
7	i	Variable	int	bubble_sort_original	Unknown	0x50016	Declared
8	i	Variable	int	bubble_sort_improved	Unknown	0x60016	Declared
9	NULL	NULL	NULL	NULL	NULL	NULL	NULL
10	interchange	Variable	bool	bubble_sort_original	Unknown	0x50060	Initialized
11	interchange	Variable	bool	bubble_sort_improved	Unknown	0x60060	Initialized
12	display	Function	void	global	None	0x40000	Return_type
13	NULL	NULL	NULL	NULL	NULL	NULL	NULL
14	numbers	Variable	int	main	32, 32, 32, 70, 92	0x10000	Array_initialized
15	NULL	NULL	NULL	NULL	NULL	NULL	NULL
16	NULL	NULL	NULL	NULL	NULL	NULL	NULL
17	NULL	NULL	NULL	NULL	NULL	NULL	NULL
18	NULL	NULL	NULL	NULL	NULL	NULL	NULL
19	bubble_sort_original	Function	void	global	1	0x40000	Return_type
20	NULL	NULL	NULL	NULL	NULL	NULL	NULL
21	main	Function	int	global	1	0x10200	Return_type
22	NULL	NULL	NULL	NULL	NULL	NULL	NULL
23	NULL	NULL	NULL	NULL	NULL	NULL	NULL
24	NULL	NULL	NULL	NULL	NULL	NULL	NULL
25	NULL	NULL	NULL	NULL	NULL	NULL	NULL
26	NULL	NULL	NULL	NULL	NULL	NULL	NULL
27	NULL	NULL	NULL	NULL	NULL	NULL	NULL
28	NULL	NULL	NULL	NULL	NULL	NULL	NULL
29	NULL	NULL	NULL	NULL	NULL	NULL	NULL
30	NULL	NULL	NULL	NULL	NULL	NULL	NULL
31	NULL	NULL	NULL	NULL	NULL	NULL	NULL
32	NULL	NULL	NULL	NULL	NULL	NULL	NULL
33	NULL	NULL	NULL	NULL	NULL	NULL	NULL
34	NULL	NULL	NULL	NULL	NULL	NULL	NULL
35	NULL	NULL	NULL	NULL	NULL	NULL	NULL
36	bubble_sort_improved	Function	void	global	1	0x40000	Return_type
37	NULL	NULL	NULL	NULL	NULL	NULL	NULL
38	temp	Variable	int	main	Unknown	0x10180	Array_initialized
39	temp	Variable	int	bubble_sort_original	Unknown	0x50040	Declared
40	iterations	Variable	int	bubble_sort_original	0	0x50048	Initialized
41	temp	Variable	int	bubble_sort_improved	Unknown	0x60040	Declared
42	NULL	NULL	NULL	NULL	NULL	NULL	NULL
43	size	Variable	int	display	5	0x50008	Parameter
44	NULL	NULL	NULL	NULL	NULL	NULL	NULL
45	NULL	NULL	NULL	NULL	NULL	NULL	NULL
46	NULL	NULL	NULL	NULL	NULL	NULL	NULL
47	NULL	NULL	NULL	NULL	NULL	NULL	NULL
48	NULL	NULL	NULL	NULL	NULL	NULL	NULL
49	NULL	NULL	NULL	NULL	NULL	NULL	NULL

- Why Holes in a Hash Table Are Undesirable

In a hash table that uses open addressing for collision resolution, removing an entry creates a hole (empty slot).

This is problematic because open addressing relies on continuous probing to locate elements efficiently. A hole disrupts this probing sequence, potentially causing incorrect lookups and insertion errors.

Problems Caused by Holes in the Symbol Table:

- Search Failures: A hole may cause premature search termination, leading to incorrect "not found" results.
- Insertion Issues: New elements may fill the hole incorrectly, disrupting the probing order and making lookups unreliable.
- Symbol Table Integrity: Missing identifiers or type mismatches may occur, leading to compilation errors.

To ensure that deletions do not disrupt the symbol table, we can use lazy deletion, rehashing, or sentinel values to manage empty slots properly.

1. Lazy Deletion (Marker-Based Approach)

   Mark an entry as "deleted" with a special flag, allowing searches to continue and future insertions to reuse the slot without disrupting the probing sequence.

2. Rehashing After Deletion

   After removing an entry, reinsert the following elements into their correct positions, ensuring table integrity. This method is computationally expensive but guarantees a consistent probing sequence.

3. Using a Sentinel Value Instead of Null Slots

   Instead of setting the slot to NULL, use a special value like "DELETED" to indicate that an entry was removed.

   This ensures that lookups do not stop at the hole while still allowing new insertions to use the slot.

Conclusion

The implementation of the symbol table highlights the importance of efficient data structures, such as hash tables, for managing symbols during the compilation process. Key takeaways include:

- Effective Error Detection and Correction: Errors such as undeclared identifiers, type mismatches, and multiple declarations were identified and handled.
- Performance: The operations performed, including error detection and file reading, are optimized for runtime performance.

In future improvements, techniques like dynamic resizing and better hash functions could further enhance the performance of the symbol table.

Annexure

Annexure 1: Input file

The content of the input file variable_data.txt, which is used by the program to load symbols into the hash table. The file contains information about various variables, their types, scopes, etc. Below is a breakdown of the first few rows from the file:

Symbol	Symbol Type	Data Type	Scope	Value	Memory Location	Remarks
numbers	Variable	int	main	32, 32, 32, 70, 92	0x10000	Array_initialized
numbers	Variable	int	main	Unknown	0x10060	Array_initialized
temp	Variable	int	main	Unknown	0x10180	Array_initialized

Explanation of the Rows

1. Row 1 (numbers):
    - The symbol numbers is declared as a variable of type int in the main function.
    - It is initialized as an array with the values: 32, 32, 32, 70, 92.
    - The memory location for this array is 0x10000, and it is marked as "Array_initialized."
2. Row 2 (numbers):
    - This is a subsequent declaration of the same numbers variable, but with an unknown value at memory location 0x10060.
    - It is still an array but its specific value is not provided here. It is marked as "Array_initialized."
3. Row 3 (temp):
    - The symbol temp is another variable of type INT declared in the main function.
    - The value of temp is unknown, and it is initialized as an array at memory location 0x10180.
    - It is marked as "Array_initialized.

Annexure 2: Insert and Read Output

The following output shows read and insert operation from the file and the time taken by the operation.

```
                                    PERFORMING READ OPERATION

Time taken to read file and insert symbols: 0.001000 seconds

                                                SYMBOL TABLE
```

Index	Name	Symbol_Type	Data_Type	Scope	Value	Memory_Location	Remarks
0	NULL	NULL	NULL	NULL	NULL	NULL	NULL
1	NULL	NULL	NULL	NULL	NULL	NULL	NULL
2	NULL	NULL	NULL	NULL	NULL	NULL	NULL
3	NULL	NULL	NULL	NULL	NULL	NULL	NULL
4	NULL	NULL	NULL	NULL	NULL	NULL	NULL
5	i	Variable	int	main	Unknown	0x20000	Declared
6	i	Variable	int	display	Unknown	0x50016	Declared
7	i	Variable	int	bubble_sort_original	Unknown	0x50016	Declared
8	i	Variable	int	bubble_sort_improved	Unknown	0x60016	Declared
9	j	Variable	Unknown	bubble_sort_improved	Unknown	Unknown	Undeclared_variable
10	interchange	Variable	bool	bubble_sort_original	Unknown	0x50060	Initialized
11	interchange	Variable	bool	bubble_sort_improved	Unknown	0x60060	Initialized
12	display	Function	void	global	None	0x40000	Return_type
13	NULL	NULL	NULL	NULL	NULL	NULL	NULL
14	numbers	Variable	int	main	32, 32, 32, 70, 92	0x10000	Array_initialized
15	numbers	Variable	int	main	Unknown	0x10060	Array_initialized
16	numbers	Variable	int	display	32, 32, 32, 70, 92	0x10000	Parameter
17	numbers	Variable	int	bubble_sort_original	32, 32, 32, 70, 92	0x10000	Parameter
18	numbers	Variable	int	bubble_sort_improved	32, 32, 32, 70, 92	0x10000	Parameter
19	bubble_sort_original	Function	void	global	1	0x40000	Return_type
20	NULL	NULL	NULL	NULL	NULL	NULL	NULL
21	main	Function	int	global	1	0x10200	Return_type
22	NULL	NULL	NULL	NULL	NULL	NULL	NULL
23	NULL	NULL	NULL	NULL	NULL	NULL	NULL
24	NULL	NULL	NULL	NULL	NULL	NULL	NULL
25	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Annexure 3 : Error Detection Output

The following output shows errors detected during the error check phase and time taken for detecting errors.

```
                                    PERFORMING ERROR CHECK

        Error: Multiple declarations of identifier: numbers
        Error: Multiple declarations of identifier: numbers
        Error: Multiple declarations of identifier: numbers
        Error: Multiple declarations of identifier: numbers
        Error: Multiple declarations of identifier: size
        Error: Multiple declarations of identifier: size
        Error: Multiple declarations of identifier: iterations
        Error: Undeclared identifier found: j


Time taken for error checks: 0.005000 seconds
```

References

1. https://www.wscubetech.com/resources/dsa/hash-table
2. https://www.geeksforgeeks.org/basics-file-handling-c/
3. https://www.tutorialspoint.com/data_structures_algorithms/hash_table_program_in_c.htm
4. https://www.geeksforgeeks.org/program-to-implement-hash-table-using-open-addressing/
5. https://github.com/jamesroutley/write-a-hash-table/blob/master/02-hash-table/README.md
6. ChatGPT, OpenAI. (2025). *Assistance in report creation and coding queries*.