CSA1455

Compiler design for Lexical Analysis

Assignment - 3

Name : Sai Lokesh Nalabothu

Reg. NO : 192365023

Branch : CSE - Cyber Security

Date : 22-02-2025

Serial. No : 27

Semantic Analysis and Type checking

Semantic analysis is a crucial phase in the compilation process that ensures a program to the Rules of its Programming language beyond just Syntax. One of its Primary Roles is type checking.

Role of Semantic Analyzer

The Semantic analyzer performs deeper checks on the code after the Syntax has been validated.

* variables and Functions are correctly declared before use.
* Type Rules are followed.
* Scope Rules are enforced.

Type Compatability

* Implicit type conversion
* Explicit type Conversion
* Structural Compatability
* Name Compatability

Example:

```
int x = 5.7;
String s = 10;
```

Error detection

* Type Mismatches
* Undeclared variables
* Multiple declarations
* Incorrect Function Calls

Example:

```
int sum (String a, int b) {
    return a + b;
}
```

1. What is Semantic Analysis and why it is Important?

It is a Phase in the Compilation Process that Ensure the Meaning of a Program is Correct according to the Rules of the Programming Language. It goes beyond Syntax Checking to verify that the code makes logical sense.

Importance

1. Ensures type safety
2. Detects Logical Errors Early
3. validates Scope and Binding Rules
4. Facilitates Code Optimization
5. Prevents Ambiguous Code Execution

Example

```
int x = "Hello";

void func (int a) { }

func (3.14);
if ("true") { }
```

2. How does Semantic analyzer Perform type checking?

The Semantic analyzer performs type checking to ensure that operations in a Program involve compatable data types. It verifiy that variables, Expressions, and Function calls follow the type Rules of the Programming language.

Steps in type checking

1. Symbol table construction.

2. Expression type Checking

3. Assignment type checking

4. Function type checking

5. operator type checking

6. Control flow type checking

7 Implicit and Explicit type checking.

The semantic analyzer performs type checking by using a symbol table, analyzing expressions, checking function calls, validating operators, and enforcing control flow Rules. This prevents errors and ensures type safety before code Execution.

C. Explain the concept of type coercoin with an example.

Type coercoin is the automatic conversion of one data into another by the compiler or Interpeter. It allows operations between different types without Explicit conversion by the programmer.

Types of type coercoin

1. Implicit type coercoin
2. Explicit type coercoin

Type coercoin simplifies coding but can lead to unExpected behaviour. especially in weakly typed languages like Javascript.

4. What type-Related errors can occur in a program, and how are they handled?

Type Related errors occur when operations, assignments or Expressions Involve incompatible data types. These Errors can lead to Compilation failure, Runtime crashes, Incorrect Results.

1. Common type-Related Errors

* type MisMatch Error
* Implicit Conversion Errors
* Division by zero
* Function Argument type MisMatch
* Operator type Error
* Scope and Declaration Errors

2. How type Errors are Handled

Compile time Handling:

* Languages like C, Java, C++ are checked types at Compile-time.

* type Errors cause Compilation failure, Preventing Execution.

Run-time Handling:

* Python and JavaScript check types at Runtime.

* Errors cause a Runtime Exception instead of Preventing Compilation.

Exception Handling:

* Languages like Java, Python, and C++ Provide Exception Handling to catch and manage type Errors.

5. Discuss the Importance of symbol tables in semantic Analysis.

A symbol table is a crucial data structure used in the semantic analysis phase of a compiler, It acts as a lookup table that stores information about variables, Functions, classes. This helps the compiler enforce type checking, scope Resolution and semantic Correctness.

1. Role of symbol tables

* variable and Function declaration tracking
* Scope management and Name Resolution
* Type checking and Compatibility.
* Function overloading & Signature Matching
* Memory Allocation & Code optimization.

2. structure of a symbol table.

Identifier	Type	Scope	MEMORY LOCATION	other Info
x	int	global	0x1001	-
y	float	local	0x2002	-
sum()	int→int	global	0x 3003	Function signature

Conclusion:

The Symbol table is an essential component of Semantic analysis in Compilers. It Ensures proper scope management, type checking, Function validation, and efficient Memory usage.