Bahir Dar University

BiT Faculty of Computing

Departments of Software Engineering

# Principles of Compiler Design on Syntax Analysis
Individual Assignment

By---> Lidia Endalamaw

ID NO---> 1506829

SUBMITTED to - Mr.Wendmu

Jan 2026

# 63. Syntax-Directed Translation Schemes (SDTS)

A **Syntax-Directed Translation Scheme (SDTS)** is a formal method used in compiler design to define how the source program is translated into an intermediate or target representation based on its **syntactic structure**. It extends a **context-free grammar** by embedding **semantic actions** within grammar productions. These semantic actions specify what computations or translations should be performed when a particular syntactic construct is recognized by the parser.

In SDTS, the **structure of the grammar** determines the order in which semantic actions are executed. As the parser analyzes the input according to the grammar rules, the associated semantic actions are triggered, allowing the compiler to perform tasks such as expression evaluation, type checking, symbol table construction, and intermediate code generation.

## Attributes in Syntax-Directed Translation

Syntax-directed translation relies on the concept of **attributes**, which are values associated with grammar symbols. Attributes carry semantic information and are evaluated using **attribute grammar rules**.

There are two main types of attributes:

➢ **Synthesized Attribute**

These attributes are computed from the attributes of child nodes in the parse tree.

Information flows **bottom-up** in the parse tree.

Commonly used in expression evaluation and code generation.

➢ **Inherited Attributes**

These attributes are passed from parent nodes or siblings to a node.

Information flows **top-down or sideways** in the parse tree.

Useful for passing contextual information such as variable types or scopes.

## ● Semantic Actions

Semantic actions are fragments of code (written in a programming language like C or C++) that are placed within grammar productions. They are executed at specific points during parsing.

➢ Example:

E → E1 + T { E.val = E1.val + T.val }

In this example:

E.val, E1.val, and T.val are attributes.

The semantic action computes the value of the expression.

## Types of Syntax-Directed Translation Schemes

- **Postfix Translation Schemes**

  Semantic actions are placed at the end of productions.

  Suitable for **bottom-up parsing**.

- **Infix Translation Schemes**

  Semantic actions are placed between grammar symbols.

  Useful when intermediate results are needed during parsing.

## ● Applications of SDTS

Syntax-directed translation schemes are widely used in different phases of a compiler, including:

Semantic analysis

Construction of syntax trees

Type checking

Intermediate code generation

Symbol table management

## ● Advantages of Syntax-Directed Translation Schemes

Provides a clear connection between syntax and semantics

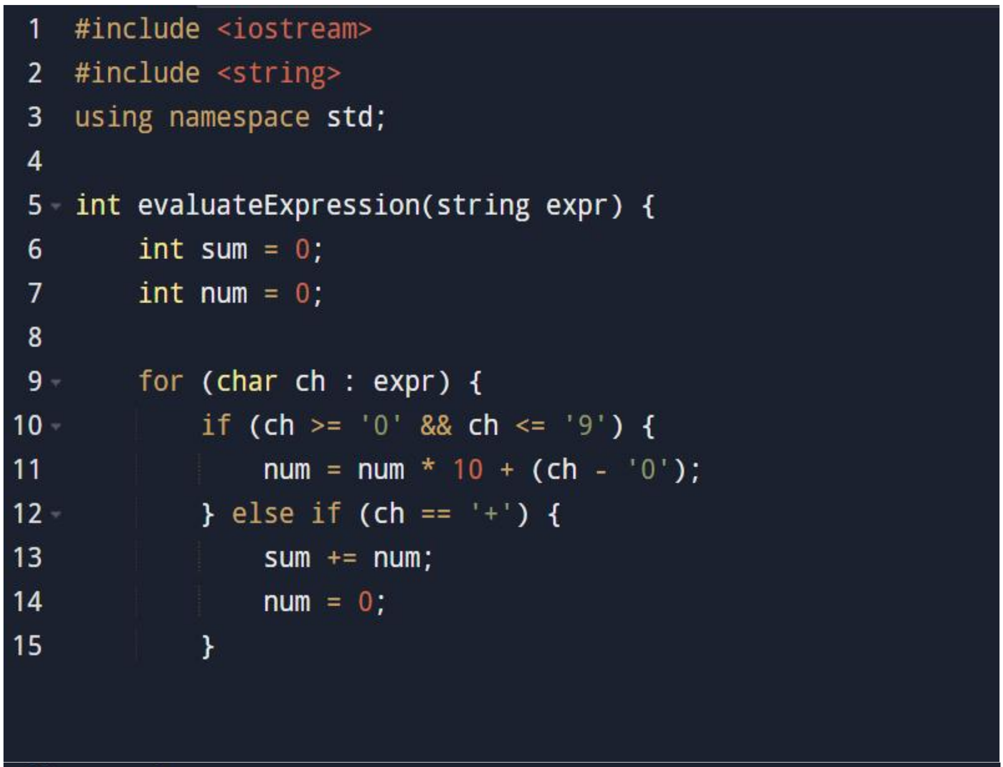
Makes compiler design systematic and organized

Simplifies the implementation of semantic analysis

Helps in automatic generation of translators

## 2. (C++) Write a C++ Function to Evaluate Arithmetic Expressions Containing Only +

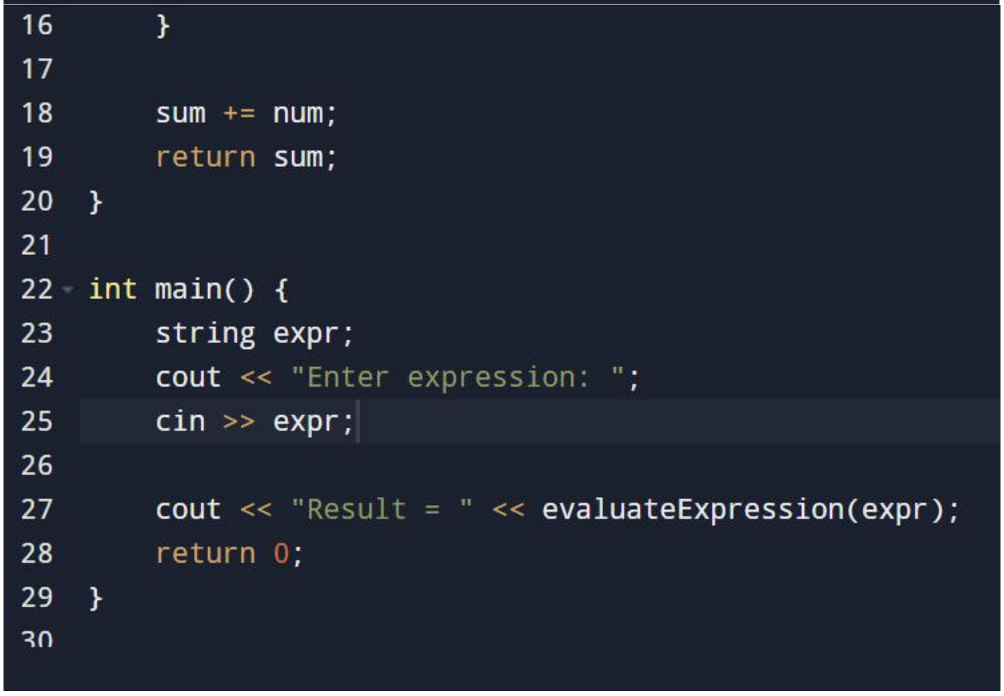
This function evaluates expressions like:
"10+20+30"



```cpp
#include <iostream>
#include <string>
using namespace std;

int evaluateExpression(string expr) {
    int sum = 0;
    int num = 0;

    for (char ch : expr) {
        if (ch >= '0' && ch <= '9') {
            num = num * 10 + (ch - '0');
        } else if (ch == '+') {
            sum += num;
            num = 0;
        }
```

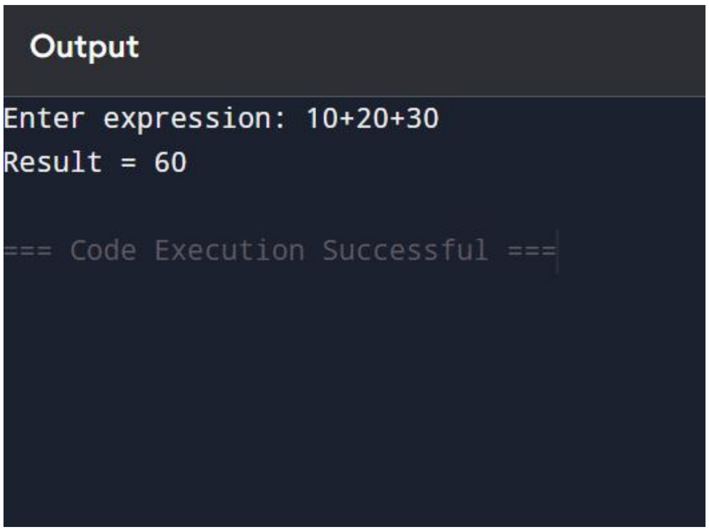


```cpp
    }

    sum += num;
    return sum;
}

int main() {
    string expr;
    cout << "Enter expression: ";
    cin >> expr;

    cout << "Result = " << evaluateExpression(expr);
    return 0;
}
```



```
Enter expression: 10+20+30
Result = 60

=== Code Execution Successful ===
```

## 3. (Problem-Solving) Draw the Parse Tree for "aaabb"

### Given Grammar:

**S → AB**
**A → aA | ε**
**B → bB | ε**

### String:

aaabb

### Derivation:

A generates aaa

B generates bb

### Parse Tree:

```
      S
     / \
    A    B
   /      \
  aA       bB
  |        |
  aA       bB
  |        |
  aA        ε
  |
  ε
```

## Leaves (left to right):

a a a b b
This matches the string **"aaabb"**