NAME\_RAFIUR RAHMAN MASHRAFI  
ID\_2221971042

CSE425.9

Task 1 : Attribute Grammar for Simple Assignment Statements

Part A – Simple Grammar

BNF Structure:

<assign> → <var> = <expr>

<expr> → <var> + <var>

<var> → A | B | C

Attribute Grammar:

Syntax Rule 1: <assign> → <var> = <expr>

Semantic Rules:

* <var>.expected\_type = any
* <expr>.expected\_type = any
* <assign>.type = <var>.actual\_type

Predicate:

* None required

Syntax Rule 2: <expr> → <var>1 + <var>2

Semantic Rules:

* <var>1.expected\_type = any
* <var>2.expected\_type = any
* <expr>.actual\_type = <var>1.actual\_type

Predicate:

* <var>1.actual\_type == <var>2.actual\_type
* Error message if false: "Type mismatch in expression: operands must have the same type"

Syntax Rule 3: <var> → A

Semantic Rules:

* <var>.actual\_type = lookup(A)

Syntax Rule 4: <var> → B

Semantic Rules:

* <var>.actual\_type = lookup(B)

Syntax Rule 5: <var> → C

Semantic Rules:

* <var>.actual\_type = lookup(C)

Part B — Attribute Grammar

<assign> → <id> = <expr>

<id> → A | B | C

<expr> → <id> + <expr>

| <id> \* <expr>

| ( <expr> )

| <id>

Attribute Grammar:

Syntax Rule 1: <assign> → <id> = <expr>

Semantic Rules:

* <id>.expected\_type = any
* <expr>.expected\_type = any
* <assign>.type = <id>.actual\_type

Predicate:

* None required

Syntax Rule 2: <expr> → <id> + <expr>₁

Semantic Rules:

* <id>.expected\_type = any
* <expr>₁.expected\_type = <id>.actual\_type
* <expr>.actual\_type = <id>.actual\_type

Predicate:

* <id>.actual\_type == <expr>₁.actual\_type
* Error message if false: "Type mismatch: operands of + must have the same type"

Syntax Rule 3: <expr> → <id> \* <expr>₁

Semantic Rules:

* <id>.expected\_type = any
* <expr>1.expected\_type = <id>.actual\_type
* <expr>.actual\_type = <id>.actual\_type

Predicate:

* <id>.actual\_type == <expr>1.actual\_type
* Error message if false: "Type mismatch: operands of \* must have the same type"

Syntax Rule 4: <expr> → ( <expr>1 )

Semantic Rules:

* <expr>1.expected\_type = <expr>.expected\_type
* <expr>.actual\_type = <expr>1.actual\_type

Predicate:

* None required

Syntax Rule 5: <expr> → <id>

Semantic Rules:

* <id>.expected\_type = <expr>.expected\_type
* <expr>.actual\_type = <id>.actual\_type

Predicate:

* None required

Syntax Rule 6: <id> → A

Semantic Rules:

* <id>.actual\_type = lookup(A)

Syntax Rule 7: <id> → B

Semantic Rules:

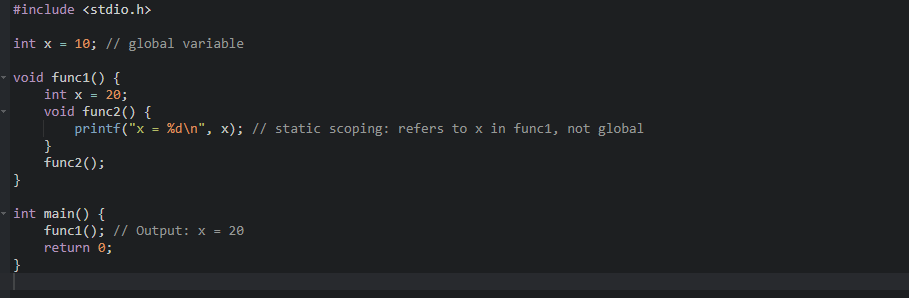
* <id>.actual\_type = lookup(B)

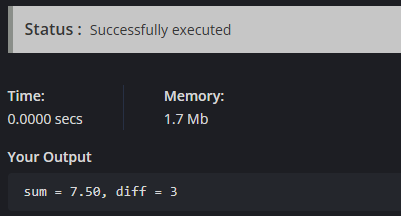
Syntax Rule 8: <id> → C

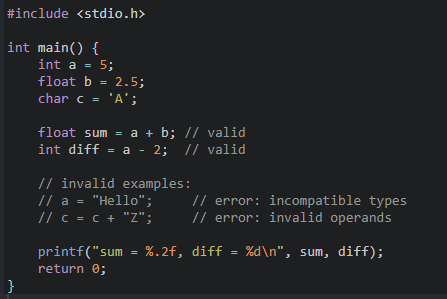
Semantic Rules:

* <id>.actual\_type = lookup(C)

Task 2: Programming Problems

1. Static Scoping in C
2. B. Compile-Time Type Checking in C





Both .c given to in Zip

1. Control Structures in C

Control structures in C are programming constructs that allow you to control the flow of program execution. They determine the order in which statements are executed based on conditions, loops, or jumps. C provides three main categories of control structures:

1. **Sequential Control**: Normal top-to-bottom execution
2. **Selection Control**: Decision-making structures (if, switch)
3. **Iteration Control**: Looping structures (for, while, do-while)
4. **Jump Control**: Unconditional transfers (break, continue, goto, return)

#### Six Flow Control Statements with Syntax

1. if-else Statement

// Syntax:

if (condition) {

// statements executed if condition is true

} else {

// statements executed if condition is false

}

1. switch Statement

// Syntax:

switch (expression) {

case constant1:

// statements

break;

case constant2:

// statements

break;

default:

// default statements

1. for Loop

// Syntax:

for (initialization; condition; increment) {

// statements to be repeated

}

1. while Loop

// Syntax:

while (condition) {

// statements to be repeated

}

1. do-while Loop

// Syntax:

do {

// statements to be executed at least once

} while (condition);

1. break Statement

// Syntax:

break;