Open Ended

Aim- Design a Mini Compiler for a language in c.

Tool Used- Vs code

Theory- Designing a mini-compiler involves several components, including lexical analysis, syntax analysis, semantic analysis, code generation, and optimization.

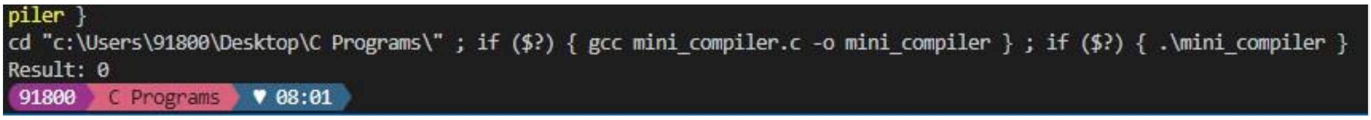
1. **Lexical Analysis (Scanner)**: The first step is to break the input source code into tokens.
2. **Syntax Analysis (Parser)**: This step involves building a parse tree or an abstract syntax tree (AST) from the tokens.
3. **Semantic Analysis**: After parsing, the compiler performs semantic analysis to check for semantic errors that cannot be captured by the grammar alone.
4. **Intermediate Code Generation**: At this stage, we can generate an intermediate representation (IR) of the code. This IR is usually closer to machine code but still abstract enough to allow for optimization.
5. **Optimization**: We can apply various optimization techniques to the IR to improve the efficiency of the generated code.
6. **Code Generation**: Finally, the compiler translates the optimized IR into target machine code.

Code-

```
      #include <stdio.h>
#include <stdlib.h>
     #include <ctype.h>
// Token types
typedef enum {
   TOK_INT,
   TOK_FLOAT,
   TOK_PLUS,
   TOK_MINUS,
   TOK_MUL,
   TOK_DIV,
   TOK_LPAREN,
   TOK_RPAREN,
   TOK_EOF
} TokenType;
// Token structure
typedef struct {
   TokenType type;
   union {
      int intval;
```

```
        float floatval;
    } value;
} Token;
// Function declarations
void next_token();
void match(TokenType expected);
int expr();
int term();
int factor();
// Global variables
Token current_token;
int main() {
    next_token(); // Read the first token
    int result = expr();
    printf("Result: %d\n", result);
    return 0;
}
// Lexical analyzer
void next_token() {
    char c = getchar();
    if (c == '+') current_token.type = TOK_PLUS;
    else if (c == '-') current_token.type = TOK_MINUS;
    else if (c == '*') current_token.type = TOK_MUL;
    else if (c == '/') current_token.type = TOK_DIV;
    else if (c == '(') current_token.type = TOK_LPAREN;
    else if (c == ')') current_token.type = TOK_RPAREN;
    else if (c == EOF) current_token.type = TOK_EOF;
    else if (isdigit(c)) {
        ungetc(c, stdin);
        scanf("%d", &current_token.value.intval);
        current_token.type = TOK_INT;
    }
    else if (c == '.') {
        ungetc(c, stdin);
        scanf("%f", &current_token.value.floatval);
        current_token.type = TOK_FLOAT;
    }
    else {
        // Handle error
    }
}
// Parser functions
int expr() {
    int result = term();
    while (current_token.type == TOK_PLUS || current_token.type == TOK_MINUS) {
        if (current_token.type == TOK_PLUS) {
            match(TOK_PLUS);
            result += term();
        } else if (current_token.type == TOK_MINUS) {
            match(TOK_MINUS);
            result -= term();
        }
    }
    return result;
}
int term() {
    int result = factor();
    while (current_token.type == TOK_MUL || current_token.type == TOK_DIV) {
        if (current_token.type == TOK_MUL) {
            match(TOK_MUL);
            result *= factor();
        } else if (current_token.type == TOK_DIV) {
            match(TOK_DIV);
            result /= factor();
        }
    }
    return result;
}
int factor() {
    int result;
    if (current_token.type == TOK_INT) {
        result = current_token.value.intval;
        match(TOK_INT);
    } else if (current_token.type == TOK_FLOAT) {
        result = current_token.value.floatval;
        match(TOK_FLOAT);
    } else if (current_token.type == TOK_LPAREN) {
        match(TOK_LPAREN);
        result = expr();
        match(TOK_RPAREN);
    } else {
        // Handle error
    }
    return result;
}
// Helper function to match expected token
void match(TokenType expected) {
    if (current_token.type == expected)
        next_token();
    else {
        // Handle error
        fprintf(stderr, "Syntax error: expected %d, found %d\n", expected, current_token.type);
        exit(1);
         }
}
```

Output-



```
piler }
cd "c:\Users\91800\Desktop\C Programs\" ; if ($?) { gcc mini_compiler.c -o mini_compiler } ; if ($?) { .\mini_compiler }
Result: 0
91800  C Programs  ♥ 08:01
```

Programme	B.Tech CSE	Course Name	Compiler Construction
Course code	CSE304	Semester	6
Student name	Abha Ghildiyal	Enrollment Number	A2305221533

Marking Criteria

Criteria	Total Marks	Marks Obtained	Comments
Concept (A)	2		
Implementation (B)	2		
Performance (C)	2		
Total	6 (To be scaled down to 1.5)		