EXPERIMENT-4

Program:

Write a c program to identifiers, keywords, white spaces and comments (Lexical analysis) in a program.

Theory:

To remove identifiers, keywords, white spaces, and comments during lexical analysis, you can follow these general steps:

1. **Define Token Types:**
    - Identify and define the different types of tokens in your programming language, such as keywords, identifiers, operators, literals, and comments.
2. **Regular Expressions:**
    - Use regular expressions to define patterns for each type of token. For example, you might use a regular expression to match keywords, identifiers, or comments.
3. **Lexical Rules:**
    - Establish lexical rules that specify how to recognize and categorize tokens based on the defined patterns. These rules serve as the foundation for the lexical analyzer.
4. **Tokenization:**
    - Implement a lexical analyzer that scans the source code and recognizes tokens based on the lexical rules. This process involves identifying keywords, identifiers, operators, literals, and comments.
5. **Filtering Out Unwanted Tokens:**
    - Once you've identified the different types of tokens, filter out the ones you want to remove. This may include discarding identifiers, keywords, white spaces, and comments.
6. **Output:**
    - Provide the filtered list of tokens as output, excluding the specified elements. This processed list can then be passed on to the next stage of the compiler for further analysis and processing.
7. **Handling White Spaces:**
    - If white spaces need to be removed, simply skip over them during the tokenization process or filter them out in the token list.
8. **Handling Comments:**

- If comments need to be removed, you can choose to ignore or skip them during the tokenization process. Regular expressions can help in recognizing and ignoring comment blocks.

Remember that the exact implementation details may vary based on the programming language and the tools you are using for lexical analysis. Many compiler construction tools, such as Lex and Flex, provide convenient ways to define token patterns and rules for lexical analysis.

Pseudo code:

```
procedure tokenize(source_code):
    initialize an empty list for tokens
    current_position = 0
    while current_position < length(source_code):
        token, new_position = match_token(source_code, current_position)
        if token is not null:
            tokens.append(token)
        current_position = new_position
    filtered_tokens = filter_unwanted_tokens(tokens)
    return filtered_tokens
procedure match_token(source_code, current_position):
    keyword_pattern = /\b(if|else|while|for)\b/
    identifier_pattern = /[a-zA-Z_][a-zA-Z0-9_]*/
    comment_pattern = /(\/\/.*|\/\*[\s\S]*?\*\/)/
    master_pattern = `${keyword_pattern}|${identifier_pattern}|${comment_pattern}`
    match_result = match(master_pattern, source_code, current_position)
    if match_result is not null:
        matched_token = match_result.group()
        new_position = current_position + length(matched_token)
        return matched_token, new_position
    // No match found
    return null, current_position + 1
procedure filter_unwanted_tokens(tokens):
    // Filter out unwanted tokens (identifiers, keywords, comments, etc.)
```

```
    filtered_tokens = []
    for token in tokens:
        if not is_unwanted_token(token):
            filtered_tokens.append(token)
    return filtered_tokens
function is_unwanted_token(token):
    // Implement logic to determine if the token is unwanted (e.g., identifier, keyword, comment)
    return is_identifier(token) or is_keyword(token) or is_comment(token)
function is_identifier(token):
    // Implement logic to check if the token is an identifier
    return match(/[a-zA-Z_][a-zA-Z0-9_]*/, token)
function is_keyword(token):
    // Implement logic to check if the token is a keyword
    return token in {'if', 'else', 'while', 'for'}
function is_comment(token):
    // Implement logic to check if the token is a comment
    return match(/\/\/.*|\/\*[\s\S]*?\*\//, token)
```

Input:

```
#include <stdio.h>
#include <stdbool.h>
#include <ctype.h>
bool is_whitespace(char c)
{    return c == ' ' || c == '\t' || c == '\n';
}bool is_operator(char c)
{    return c == '+' || c == '-' || c == '*' || c == '/' || c == '%' || c == '=' || c == '<' || c == '>';
}bool is_valid_identifier_char(char c)
{    return isalnum(c) || c == '_';
}void remove_identifiers(FILE *input_file, FILE *output_file) {
    int c;
    bool in_identifier = false;
```

```
    while ((c = fgetc(input_file)) != EOF)    {
        if (is_valid_identifier_char(c))       {
            in_identifier = true;
        } else        {
            if (in_identifier)            {
                fputc(' ', output_file);
                in_identifier = false;           }
            fputc(c, output_file);       }    }}
int main()
{    FILE *input_file, *output_file;
    char input_filename[100], output_filename[100];
    printf("Enter the input file name: ");
    scanf("%s", input_filename);
    input_file = fopen(input_filename, "r");
    if (input_file == NULL)
    {       printf("Error in opening input file.\n");
        return 1;
    }    printf("Enter the output file name: ");
    scanf("%s", output_filename);
    output_file = fopen(output_filename, "w");
    if (output_file == NULL)
    {       printf("Error opening output file.\n");
        fclose(input_file);
        return 1;
    }    remove_identifiers(input_file, output_file);
    fclose(input_file);
    fclose(output_file);
    printf("Identifiers removed successfully.\n");
    return 0;}
```

Output:

```
Enter the input file name: Original.txt
Enter the output file name: File.txt
Error opening output file.
```

main.c | Original.txt | File.txt

```c
#include<stdio.h>
int main()
{
    printf("Hello Asmitha");
}
```

main.c | Original.txt | File.txt

```c
# < . >
   ()
{
     ("    ");
}
```

```
Enter the input file name: Original.txt
Enter the output file name: File.txt
Identifiers removed successfully.


...Program finished with exit code 0
Press ENTER to exit console.
```

Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science and Engineering Amity University, Noida (U.P)			
Programme	B.Tech CSE	Course Name	Complier Construction
Course Code	CSE304	Semester	6
Student Name		Enrollment No.	
Marking Criteria			
Criteria	Total Marks	Marks Obtained	Comments
Concept	2		
Implementation	2		
Performance	2		
Total	6		