



Green University of Bangladesh
Department of Computer Science and Engineering (CSE)
Faculty of Sciences and Engineering
Semester: (Spring, Year:2024), B.Sc. in CSE (Day)

Lab Report # 01
Course Title: Compiler Lab

Course Code: CSE 306 Section: 222_D2

Lab Experiment Name: Design a lexical analyzer for a given language.

Student Details

Name		ID
	Md. Moshir Rahman	221902324

Submission Date : 03/03/2024
Course Teacher's Name : Tasnim Tayiba Zannat

Lab Report Status

Marks:
Comments:.....

Signature:.....
Date:.....

1. EXPERIMENT NAME:

- Write a program to recognize identifiers.
- Write a program to recognize constants.
- Write a program to recognize keywords and identifiers.
- Write a program to ignore the comments in the given input source program.

2. OBJECTIVES

- ★ Identify whether a given input is a valid C identifier.
- ★ Recognize if a given input is a constant in C.
- ★ Differentiate between C keywords and identifiers in a given input.
- ★ Removes comments from C source code input

3. INTRODUCTION

Comments are an important part of any programming language. They allow developers to add notes and explanations to their code without affecting the execution. Comments make code more readable and maintainable.

4. ALGORITHM

Task 1:

1. Define a function ``is_identifier`` that takes a string as input.

Check if the first character is an alphabet or underscore

If not, return 0 (false)

b. Iterate through the remaining characters of the string

For each character:

Check if it is alphanumeric or an underscore

If not, return 0 (false)

In the main function:

a. Get the input string from the user

b. Call the ``is_identifier`` function with the input string

c. If the function returns 1 (true), print that the string is an identifier

d. If the function returns 0 (false), print that the string is not an identifier

Task 2:

1. Define a function ``is_constant`` that takes a string as input

2. Inside the function:

a. Check if the first character is a digit

If not, return 0 (false)

b. Iterate through the remaining characters of the string

For each character:

Check if it is a digit

If not, return 0 (false)

c. If all characters are digits, return 1 (true)

3. In the main function:

a. Get the input string from the user

b. Call the `'is_constant'` function with the input string

c. If the function returns 1 (true), print that the string is a constant

d. If the function returns 0 (false), print that the string is not a constant

Task 3:

1. Algorithm:

1. Define a function `'is_keyword'` that takes a string as input

2. Inside the function:

a. Define an array of strings containing all the keywords

b. Iterate through the array of keywords

For each keyword:

Compare the input string with the keyword

If they match, return 1 (true)

c. If no match is found, return 0 (false)

3. Define a function `'is_identifier'` (same as in problem 1)

4. In the main function:

a. Get the input string from the user

b. Call the `'is_keyword'` function with the input string

c. If `'is_keyword'` returns 1 (true), print that the string is a keyword

d. Else, call the `'is_identifier'` function with the input string

e. If `'is_identifier'` returns 1 (true), print that the string is an identifier

f. If both functions return 0 (false), print that the string is neither a keyword nor an identifier.

Task 4:

1. Define a function ``ignore_comments`` that takes two strings as input:

- The input program
- An empty output string

2. Inside the function:

- a. Initialize a flag variable ``in_comment`` to 0 (false)
- b. Iterate through the input program character by character

For each character:

If ``in_comment`` is 1 (true):

Check if the current character is `'*'`` and the next character is `'/'``

If yes, set ``in_comment`` to 0 (false) and skip the next character

Else:

If the current character is `'/'`` and the next character is `'/'``:

Skip all characters until the end of the line (`'\n'``)

Else if the current character is `'/'`` and the next character is `'*'``:

Set ``in_comment`` to 1 (true) and skip the next character

Else:

Append the current character to the output string

- c. Terminate the output string with a null character (`'\0'``)

3. In the main function:

- a. Get the input program from the user
- b. Define an empty output string
- c. Call the ``ignore_comments`` function with the input program and the output string
- d. Print the output string (without comments)

4&5. PROCEDURE & OUTPUT

Implementation:

(A) Recognize identifiers.

```

#include <stdio.h>
#include <string.h>
#include <ctype.h>

int isIdentifier(char str[]) {
    if (!isalpha(str[0]) && str[0] != '_') {
        return 0;
    }

    for (int i = 1; i < strlen(str); i++) {
        if (!isalnum(str[i]) && str[i] != '_') {
            return 0;
        }
    }

    return 1;
}

int main() {
    char input[50];

    printf("Enter an identifier: ");
    scanf("%s", input);

    if (isIdentifier(input)) {
        printf("It is a valid identifier.\n");
    } else {
        printf("It is not a valid identifier.\n");
    }

    return 0;
}

```

Output:

```

Enter an identifier: moshiur
It is a valid identifier.

```

B) Recognize constants:

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int isConstant(char str[]) {
    char *endptr;
    strtod(str, &endptr);
    return (*endptr == '\0');
}

int main() {
    char input[50];

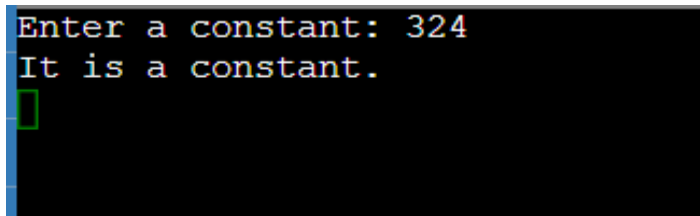
    printf("Enter a constant: ");
    scanf("%s", input);

    if (isConstant(input)) {
        printf("It is a constant.\n");
    } else {
        printf("It is not a constant.\n");
    }

    return 0;
}

```

Output:



```

Enter a constant: 324
It is a constant.

```

C) Recognize keywords and identifiers:

```

#include <stdio.h>
#include <string.h>

int isKeyword(char *str) {
    char keywords[][20] = {
        "auto", "break", "case", "char", "const", "continue", "default", "do",

```

```

    "struct", "switch", "typedef", "union", "unsigned", "void", "volatile", "while"
};

for (int i = 0; i < sizeof(keywords) / sizeof(keywords[0]); i++) {
    if (strcmp(str, keywords[i]) == 0) {
        return 1;
    }
}

return 0;
}

int isIdentifier(char *str) {
    if (!(isalpha(str[0]) || str[0] == '_')) {
        return 0;
    }

    for (int i = 1; i < strlen(str); i++) {
        if (!(isalnum(str[i]) || str[i] == '_')) {
            return 0;
        }
    }

    return 1;
}

int main() {
    char input[50];

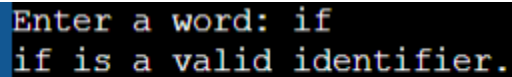
    printf("Enter a word: ");
    scanf("%s", input);

    if (isKeyword(input)) {
        printf("%s is a C keyword.\n", input);
    } else if (isIdentifier(input)) {
        printf("%s is a valid identifier.\n", input);
    } else {
        printf("%s is not a keyword or a valid identifier.\n", input);
    }
}

```

```
    return 0;
}
```

Output:



```
Enter a word: if
if is a valid identifier.
```

D) Ignore the comments in the given input source program:

```
#include <stdio.h>
```

```
int main() {
    char line[1000];
    int inComment = 0;

    printf("Enter source program with comments:\n");

    while (fgets(line, sizeof(line), stdin) != NULL) {
        for (int i = 0; i < sizeof(line) && line[i] != '\0'; i++) {
            if (!inComment) {
                if (line[i] == '/' && line[i + 1] == '/') {
                    break;
                } else if (line[i] == '/' && line[i + 1] == '*') {
                    inComment = 1;
                    i++;
                } else {
                    putchar(line[i]);
                }
            } else {
                if (line[i] == '*' && line[i + 1] == '/') {
                    inComment = 0;
                    i++;
                }
            }
        }
    }
}
```



```
        return 0;
    }
```

Output:

Enter source program with comments:

```
#include <stdio.h>
```

```
int main() {
    printf("Hello, GUB!\n");
}
```

```
#include <stdio.h>
```

```
int main() {
    printf("Hello, GUB!\n");
```

```
}
```

```
#include <stdio.h>
```

```
int main() {
    printf("Hello, GUB!\n");
}
```

7. ANALYSIS AND DISCUSSION

The lab exercises focused on implementing lexical analyzers, which are an essential part of a compiler's frontend. A lexical analyzer is responsible for tokenizing the input source code by recognizing patterns of characters that represent different token types, such as identifiers, keywords, constants, operators, and comments. One of the challenges faced during the implementation was handling edge cases and ensuring that the lexical analyzer was robust enough to handle various input scenarios. For example, properly handling nested multi-line comments, identifiers with special characters, and ensuring that the analyzer did not misinterpret one token type for another.