Vellore Institute Of Technology

Compiler design lab

Assessment – 2

L45+L46

Professor: Sureshkumar WI

Harish Thangaraj

21BRS1033

I affirm that all the work presented against this assignment is done by me with limited Internet reference and can be tested on the logic and concepts if necessary .

Experiment – 1.1: To identify tokens from a simple expression stored in the form of a linear array.

Algorithm:

1. Import the necessary header files.
2. Predefine max tokens and max length of each token as macros.
3. Define a function to classify tokens taking token array as the argument.
    a. Validate each index for keywords, constants, operators, identifiers and return respective types.
4. Define a function to identify tokens from the given statement and store them in an array
5. Under the main function, initialize arrays for statement, tokens, and types.
6. Take input for statement from the user.
7. Call the above functions to classify, identify and return tokens as output.

Source code:

Volumes > Harish > Winter 2023-2024 > Compiler design > Lab > Lab 2 > C tokenss.c > main()

```c

/*
Name : Harish Thangaraj
Reg_no: 21BRS1033
Slot: L45+L46
*/


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

#define MAX_TOKENS 100
#define MAX_TOKEN_LENGTH 50

// Function to classify the type of token
const char* classify_token(const char *token) {
    // Check if the token is a keyword
    const char *keywords[] = {"auto", "break", "case", "char", "const", "continue", "default", "do", "double", "else", "enum", "extern", "float", "for", "goto", "if", "int", "long",
    for (int i = 0; i < sizeof(keywords) / sizeof(keywords[0]); i++) {
        if (strcmp(token, keywords[i]) == 0) {
            return "Keyword";
        }
    }

    // Check if the token is an identifier
    if (isalpha(token[0]) || token[0] == '_') {
        return "Identifier";
    }

    // Check if the token is a constant
    int valid_constant = 1;
    for (int i = 0; i < strlen(token); i++) {
        if (!isdigit(token[i])) {
            valid_constant = 0;
            break;
        }
    }
    if (valid_constant) {
        return "Constant";
    }

    // Check if the token is an operator or special symbol
    const char *operators[] = {"+", "-", "*", "/", "=", "==", "!=", "<", ">", "<=", ">=", "&&", "||", "!", "&", "|", "^", "~", "<<", ">>", "++", "--", "+=", "-=", "*=", "/=", "%=",
    for (int i = 0; i < sizeof(operators) / sizeof(operators[0]); i++) {
        if (strcmp(token, operators[i]) == 0) {
            return "Operator/Special Symbol";
        }
    }

    // If none of the above conditions match, classify the token as "Unknown"
    return "Unknown";
```

Fig. 1.1.1

```c
// Function to identify tokens from a statement and store them in an array
void identify_tokens(const char *statement, char tokens[][MAX_TOKEN_LENGTH], char types[][20], int *num_tokens) {
    char *token;
    char copy[strlen(statement) + 1]; // Create a copy of the statement to avoid modifying the original string
    strcpy(copy, statement);

    // Tokenize the statement using strtok
    token = strtok(copy, " ");
    while (token != NULL) {
        // Copy each token into the tokens array
        strcpy(tokens[*num_tokens], token);
        // Classify the token
        strcpy(types[*num_tokens], classify_token(token));
        (*num_tokens)++;

        // Get the next token
        token = strtok(NULL, " ");
    }
}

int main() {
    char statement[MAX_TOKEN_LENGTH * MAX_TOKENS];
    char tokens[MAX_TOKENS][MAX_TOKEN_LENGTH];
    char types[MAX_TOKENS][20];
    int num_tokens = 0;

    // Input a statement
    printf("Enter a statement: ");
    fgets(statement, sizeof(statement), stdin);

    // Remove newline character from input
    if (statement[strlen(statement) - 1] == '\n') {
        statement[strlen(statement) - 1] = '\0';
    }

    // Identify tokens and store them in the tokens array
    identify_tokens(statement, tokens, types, &num_tokens);

    // Print the tokens and their types
    printf("Tokens and their types:\n");
    for (int i = 0; i < num_tokens; i++) {
        printf("%s: %s\n", tokens[i], types[i]);
    }

    return 0;
}
```

Fig. 1.1.2

Output :

```
(base) casarulez@Harishs-MacBook-Pro output % ./"tokenss"
Enter a statement: Hello my name is 123
Tokens and their types:
Hello: Identifier
my: Identifier
name: Identifier
is: Identifier
123: Constant
```

Fig. 1.1.3

Experiment – 1.2: To identify tokens from a simple program written in a text file.

Algorithm:

1. Import the necessary header files.
2. Predefine max tokens and max length of each token as macros.
3. Define a function to classify tokens taking token array as the argument.
    a. Validate each index for keywords, constants, operators, identifiers and return respective types.
4. Define a function to identify tokens from the given statement and store them in an array
5. Under the main function, initialize arrays for statement, tokens, types and filename.
6. Take input for filename from user.
7. Read each line and process the tokens in the file and also eliminated the new line characters.
8. Call the above functions to classify, identify and return tokens as output.

Source code:

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#define MAX_TOKENS 100
#define MAX_TOKEN_LENGTH 50

// Function to classify the type of token
const char* classify_token(const char *token) {
    // Check if the token is a keyword
    const char *keywords[] = {"auto", "break", "case", "char", "const", "continue", "default", "do", "double", "else", "enum", "extern", "float", "for", "goto", "if", "int", "long",
    for (int i = 0; i < sizeof(keywords) / sizeof(keywords[0]); i++) {
        if (strcmp(token, keywords[i]) == 0) {
            return "Keyword";
        }
    }

    // Check if the token is an identifier
    if (isalpha(token[0]) || token[0] == '_') {
        return "Identifier";
    }

    // Check if the token is a constant (for simplicity, we'll consider integers as constants)
    int valid_constant = 1;
    for (int i = 0; i < strlen(token); i++) {
        if (!isdigit(token[i])) {
            valid_constant = 0;
            break;
        }
    }
    if (valid_constant) {
        return "Constant";
    }

    // Check if the token is an operator or special symbol
    const char *operators[] = {"+", "-", "*", "/", "=", "==", "!=", "<", ">", "<=", ">=", "&&", "||", "!", "&", "|", "^", "~", "<<", ">>", "++", "--", "+=", "-=", "*=", "/=", "%=",
    for (int i = 0; i < sizeof(operators) / sizeof(operators[0]); i++) {
        if (strcmp(token, operators[i]) == 0) {
            return "Operator/Special Symbol";
        }
    }

    // If none of the above conditions match, classify the token as "Unknown"
    return "Unknown";
}

// Function to identify tokens from a statement and store them in an array
void identify_tokens(const char *statement, char tokens[][MAX_TOKEN_LENGTH], char types[][20], int *num_tokens) {
    char *token;
    char copy[strlen(statement) + 1]; // Create a copy of the statement to avoid modifying the original string
    strcpy(copy, statement);
```

Fig. 1.2.1

```c
    // Tokenize the statement using strtok
    token = strtok(copy, " ");
    while (token != NULL) {
        // Copy each token into the tokens array
        strcpy(tokens[*num_tokens], token);
        // Classify the token
        strcpy(types[*num_tokens], classify_token(token));
        (*num_tokens)++;

        // Get the next token
        token = strtok(NULL, " ");
    }

int main() {
    char filename[100];
    char statement[MAX_TOKEN_LENGTH * MAX_TOKENS];
    char tokens[MAX_TOKENS][MAX_TOKEN_LENGTH];
    char types[MAX_TOKENS][20];
    int num_tokens = 0;

    // Input filename
    printf("Enter the filename: ");
    scanf("%s", filename);

    // Open the file
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }

    // Read each line from the file and process it
    while (fgets(statement, sizeof(statement), file)) {
        // Remove newline character from input
        if (statement[strlen(statement) - 1] == '\n') {
            statement[strlen(statement) - 1] = '\0';
        }

        // Identify tokens and store them in the tokens array
        identify_tokens(statement, tokens, types, &num_tokens);
    }

    // Close the file
    fclose(file);

    // Print the tokens and their types
    printf("Tokens and their types:\n");
    for (int i = 0; i < num_tokens; i++) {
        printf("%s: %s\n", tokens[i], types[i]);
    }

    return 0;
```

Fig. 1.2.2

Output:

```
(base) casarulez@Harishs-MacBook-Pro output % ./"tokenstf"
Enter the filename: /Users/casarulez/Documents/code.txt
Tokens and their types:
int: Keyword
a=2: Identifier
int: Keyword
b=3: Identifier
int: Keyword
c=a+b: Identifier
```

Fig. 1.2.3

Code.txt:

```
int a=2
int b=3
int c=a+b
```

Fig. 1.2.4

Experiment -1.3: To identify tokens from a simple program written in a text file and write the token output in another text file.

Algorithm:

1. Import the necessary header files.
2. Predefine max tokens and max length of each token as macros.
3. Define a function to classify tokens taking token array as the argument.
    a. Validate each index for keywords, constants, operators, identifiers and return respective types.
4. Define a function to identify tokens from the given statement and store them in an array
5. Under the main function, initialize arrays for statement, tokens, types and filename.
6. Take input for filename from user.
7. Read each line and process the tokens in the file and also eliminated the new line characters.
8. Call the above functions to classify, identify and write the tokens in a text file as output.

Source code:

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#define MAX_TOKENS 100
#define MAX_TOKEN_LENGTH 50

// Function to classify the type of token
const char* classify_token(const char *token) {
    // implementation...
}

// Function to identify tokens from a statement and store them in an array
void identify_tokens(const char *statement, char tokens[][MAX_TOKEN_LENGTH], char types[][20], int *num_tokens) {
    // implementation...
}

int main() {
    char filename[100];
    char statement[MAX_TOKEN_LENGTH * MAX_TOKENS];
    char tokens[MAX_TOKENS][MAX_TOKEN_LENGTH];
    char types[MAX_TOKENS][20];
    int num_tokens = 0;

    // Input filename
    printf("Enter the filename: ");
    scanf("%s", filename);

    // Open the input file
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }

    // Open the output file
    FILE *output_file = fopen("output.txt", "w");
    if (output_file == NULL) {
        printf("Error creating output file.\n");
        return 1;
    }

    // Read each line from the file and process it
    while (fgets(statement, sizeof(statement), file)) {
        // Remove newline character from input
        if (statement[strlen(statement) - 1] == '\n') {
            statement[strlen(statement) - 1] = '\0';
        }

        // Identify tokens and store them in the tokens array
        identify_tokens(statement, tokens, types, &num_tokens);
    }
```

Fig. 1.3.1

```
    // Close the input file
    fclose(file);

    // Write the tokens and their types to the output file
    fprintf(output_file, "Tokens and their types:\n");
    for (int i = 0; i < num_tokens; i++) {
        fprintf(output_file, "%s: %s\n", tokens[i], types[i]);
    }

    // Close the output file
    fclose(output_file);

    printf("Tokens output has been written to output.txt\n");

    return 0;
}
```

Fig. 1.3.2

Output:

```
(base) casarulez@Harishs-MacBook-Pro output % ./"tokenswtf"
Enter the filename: /Users/casarulez/Documents/file.txt
Tokens output has been written to output.txt
```

Fig. 1.3.3

Code.txt:

```
int a=2
int b=3
int c=a+b
```

Fig. 1.3.4

Output.txt:

```
int: Keyword
a=2: Identifier
int: Keyword
b=3: Identifier
Int: Keyword
c=a+b Identifier
```

Fig. 1.3.5

Experiment – 2: To construct a lexical analyzer using LEX tool.

Lex tool used: **Flex**

Lexer.l :

lexer.l

```
%{
#include <stdio.h>
%}

%%
[0-9]+          { printf("NUMBER: %s\n", yytext); }
[a-zA-Z]+       { printf("WORD: %s\n", yytext); }
[ \t\n]         { /* ignore whitespace */ }
.               { printf("UNKNOWN: %s\n", yytext); }
%%

int yywrap() {
    return 1;  // Indicate that there are no more input streams
}

int main() {
    yylex();
    return 0;
}

```

Fig. 2.1

Compiler and running the lexer:

```
(base) casarulez@Harishs-MacBook-Pro ~ % cd /Users/casarulez/Documents
(base) casarulez@Harishs-MacBook-Pro Documents % flex lexel.l
flex: can't open lexel.l
(base) casarulez@Harishs-MacBook-Pro Documents % flex lexer.l
(base) casarulez@Harishs-MacBook-Pro Documents % gcc lex.yy.c -o lexer -ll
ld: warning: object file (/Library/Developer/CommandLineTools/SDKs/MacOSX14.2.sd
k/usr/lib/libl.a[arm64][3](libyywrap.o)) was built for newer 'macOS' version (14
.2) than being linked (14.0)
(base) casarulez@Harishs-MacBook-Pro Documents % ./lexer
```

Output:

```
(base) casarulez@Harishs-MacBook-Pro Documents % ./lexer

hello 123
WORD: hello
NUMBER: 123
```

----------**********----------