****

**Programming Assignment 1**

**Implementation of Lexical Analyser and Symbol Table**

Name: Jayannthan P T

Dept: CSE ‘A’

Roll No.: 205001049

C Programming Language: Lexical construct

**Source code:**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <stdbool.h>

#include <fcntl.h>

#define MAX 10000

typedef struct SYMBOL\_TABLE

{

    char id[10];

    char type[5];

    int bytes;

    int address;

    char value[15];

} table;

table tbl[15];

int entry = 0;

char preID[10];

char val[10];

char key[10];

void addEntry(char string[])

{

    strcpy(tbl[entry].id, string);

    strcpy(tbl[entry].type, key);

    if (strcmp(key, "int") == 0)

    {

        tbl[entry].bytes = 2;

    }

    else

    {

        tbl[entry].bytes = 4;

    }

    if (entry == 0)

    {

        tbl[entry].address = 1000;

    }

    else

    {

        tbl[entry].address = tbl[entry - 1].address + tbl[entry - 1].bytes;

    }

    strcpy(tbl[entry].value, val);

    entry++;

}

int find(char string[])

{

    for (int i = 0; i < entry; i++)

    {

        if (strcmp(tbl[i].id, string) == 0)

        {

            return 1;

        }

    }

    return 0;

}

bool isDelim(char ch)

{

    if (ch == ' ' || ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == ',' || ch == ';' || ch == '>' || ch == '<' || ch == '=' || ch == '(' || ch == ')' || ch == '[' || ch == ']' || ch == '{' || ch == '}' || ch == '!' || ch == '&' || ch == '|')

    {

        return true;

    }

    return false;

}

bool isSplChar(char ch)

{

    if (ch == '{' || ch == '}' || ch == ';' || ch == ',' || ch == '(' || ch == ')')

    {

        return true;

    }

    return false;

}

bool isBracket(char ch)

{

    if (ch == ')')

    {

        return true;

    }

    return false;

}

bool isPre(char ch)

{

    if (ch == '>')

    {

        return true;

    }

    return false;

}

bool isLogicOp(char ch)

{

    if (ch == '!' || ch == '&' || ch == '|' || ch == '=')

    {

        return true;

    }

    return false;

}

bool isRelOp(char ch)

{

    if (ch == '>' || ch == '<' || ch == '=')

    {

        return true;

    }

    return false;

}

bool isArithOp(char ch)

{

    if (ch == '+' || ch == '-' || ch == '\*' || ch == '/')

    {

        return true;

    }

    return false;

}

bool isAssnOp(char ch)

{

    if (ch == '=')

    {

        return true;

    }

    return false;

}

bool valIden(char \*string)

{

    if (string[0] == '0' || string[0] == '1' || string[0] == '2' || string[0] == '3' || string[0] == '4' || string[0] == '5' || string[0] == '6' || string[0] == '7' || string[0] == '8' || string[0] == '9' || isDelim(string[0]) == true)

    {

        return false;

    }

    return true;

}

bool isKey(char \*string)

{

    if (!strcmp(string, "if") || !strcmp(string, "else") || !strcmp(string, "while") || !strcmp(string, "do") || !strcmp(string, "break") || !strcmp(string, "continue") || !strcmp(string, "int") || !strcmp(string, "double") || !strcmp(string, "float") || !strcmp(string, "return") || !strcmp(string, "char") || !strcmp(string, "case") || !strcmp(string, "char") || !strcmp(string, "sizeof") || !strcmp(string, "long") || !strcmp(string, "short") || !strcmp(string, "typedef") || !strcmp(string, "switch") || !strcmp(string, "unsigned") || !strcmp(string, "void") || !strcmp(string, "static") || !strcmp(string, "struct") || !strcmp(string, "goto"))

    {

        return true;

    }

    return false;

}

bool isFunc(char \*string)

{

    if (!strcmp(string, "printf") || !strcmp(string, "scanf") || !strcmp(string, "getch") || !strcmp(string, "clrscr") || !strcmp(string, "main"))

    {

        return true;

    }

    return false;

}

bool isInt(char \*string)

{

    int i, len = strlen(string);

    if (len == 0)

    {

        return false;

    }

    for (i = 0; i < len; i++)

    {

        if (string[i] != '0' && string[i] != '1' && string[i] != '2' && string[i] != '3' && string[i] != '4' && string[i] != '5' && string[i] != '6' && string[i] != '7' && string[i] != '8' && string[i] != '9' || (string[i] == '-' && i > 0))

        {

            return false;

        }

    }

    return true;

}

bool isRealNo(char \*string)

{

    int i, len = strlen(string);

    bool hasDecimal = false;

    if (len == 0)

    {

        return false;

    }

    for (i = 0; i < len; i++)

    {

        if (string[i] != '0' && string[i] != '1' && string[i] != '2' && string[i] != '3' && string[i] != '4' && string[i] != '5' && string[i] != '6' && string[i] != '7' && string[i] != '8' && string[i] != '9' && string[i] != '.' || (string[i] == '-' && i > 0))

        {

            return false;

        }

        if (string[i] == '.')

        {

            hasDecimal = true;

        }

    }

    return (hasDecimal);

}

char \*SubStrGen(char \*string, int left, int right)

{

    int i;

    char \*subString = (char \*)malloc(sizeof(char) \* (right - left + 2));

    for (i = left; i <= right; i++)

    {

        subString[i - left] = string[i];

    }

    subString[right - left + 1] = '\0';

    return (subString);

}

void parser(char \*string)

{

    int left = 0, right = 0;

    int len = strlen(string);

    while (right <= len && left <= right)

    {

        if (string[right] == '\n')

        {

            right++;

            left = right;

            continue;

        }

        if (isDelim(string[right]) == false)

            right++;

        if (isDelim(string[right]) == true && left == right)

        {

            if (isSplChar(string[right]) == true)

            {

                printf("%c   -->   is a special character\n", string[right]);

            }

            else if (!isAssnOp(string[right]) && isLogicOp(string[right]) == true)

            {

                if (isLogicOp(string[right]) == true && isLogicOp(string[right + 1]) == true)

                {

                    printf("%c%c  -->   is a logical operator\n", string[right], string[right + 1]);

                    right++;

                }

                else if (isLogicOp(string[right]) == true)

                {

                    printf("%c   -->   is a logical operator\n", string[right]);

                }

            }

            else if (!isAssnOp(string[right]) && isRelOp(string[right]) == true)

            {

                if (isRelOp(string[right]) == true && isRelOp(string[right + 1]) == true)

                {

                    printf("%c   -->   is a relational operator\n", string[right], string[right + 1]);

                    right++;

                }

                else if (isRelOp(string[right]) == true)

                {

                    printf("%c   -->   is a relational operator\n", string[right]);

                }

            }

            else if (isArithOp(string[right]))

            {

                printf("%c   -->   is an arithmetic operator\n", string[right]);

            }

            else if (isAssnOp(string[right]))

            {

                printf("%c   -->   is an assignment operator\n", string[right]);

            }

            right++;

            left = right;

        }

        else if (isDelim(string[right]) == true && left != right || (right == len && left != right))

        {

            char \*subString = SubStrGen(string, left, right - 1);

            if (isKey(subString) == true)

            {

                strcpy(key, subString);

                printf("%s   -->   is a keyword\n", subString);

            }

            else if (strcmp(subString, "#include") == 0)

            {

                right++;

                while (isPre(string[right]) == false)

                    right++;

                right = right + 1;

                char \*subString = SubStrGen(string, left, right - 1);

                printf("%s   -->   is a preprocessor directive\n", subString);

            }

            else if (isFunc(subString) == true)

            {

                right++;

                while (isBracket(string[right]) == false)

                {

                    right++;

                }

                right = right + 1;

                char \*subString = SubStrGen(string, left, right - 1);

                printf("%s   -->   is a function call\n", subString);

            }

            else if (isInt(subString) == true)

            {

                printf("%s   -->   is an integer\n", subString);

                strcpy(val, subString);

                addEntry(preID);

            }

            else if (isRealNo(subString) == true)

            {

                printf("%s   -->   is a real number\n", subString);

                strcpy(val, subString);

                addEntry(preID);

            }

            else if (valIden(subString) == true && isDelim(string[right - 1]) == false)

            {

                printf("%s   -->   is a valid identifier\n", subString);

                strcpy(preID, subString);

            }

            else if (valIden(subString) == false && isDelim(string[right - 1]) == false)

            {

                printf("%s   -->   IS NOT A valid identifier\n", subString);

            }

            left = right;

        }

    }

    return;

}

int main()

{

    int fd = open("test.c", O\_RDONLY);

    if (fd == -1)

    {

        printf("File not found...");

        return 0;

    }

    char buffer[MAX - 1];

    int size = read(fd, buffer, MAX);

    buffer[size] = '\0';

    printf("x----------x\n  Tokens    \nx----------x\n");

    parser(buffer);

    printf("\n\n\nx---------------x\n  Symbol Table    \nx---------------x\n");

    for (int i = 0; i < entry; i++)

    {

        printf("%s %s %d %d %s\n", tbl[i].id, tbl[i].type, tbl[i].bytes, tbl[i].address, tbl[i].value);

    }

    return (0);

}

**Input Code:**

#include <stdio.h>

main()

{

    int a = 10, b = 20;

    if (a > b)

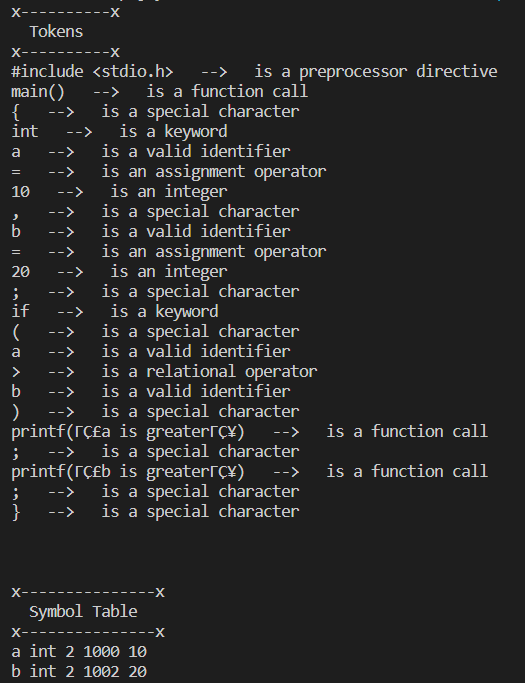
        printf(“a is greater”);

    else

        printf(“b is greater”);

}

**Output:**

****

**Learning Outcome:**

* Understood the working of lexical analyser for debugging of programs.
* Understood the role of lexical analyser in running a program