190905104
Lab 4

1)
// Using getNextToken( ) implemented in Lab No 3,design a Lexical Analyser to
// implement local and global symbol table to store tokens for identifiers using
// array of structure.

LexAnalyser.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "clean.h"
#include "preprocesses.h"
#define MAX_SIZE 30

char keywords[33][10] = {"auto", "double", "int", "struct", "break", "else",
"long", "switch", "case", "enum", "register", "typedef", "char", "extern",
"return", "union", "const", "float", "short", "default", "unsigned", "continue",
"for", "signed", "void", "goto", "sizeof", "voltile", "do", "if", "static", "while"
};

char dataTypes[][10] = {
        "double",
        "int",
        "float",
        "char"
};

char operators[5]={'-','%','+','*','/'};

char brackets[6] = {'(',')','{','}','[',']'};

char specialSymbols[12] = {
        '*',
        '<',
        '>',
        ';',
        '.',
        ',',
        '~',
        '`',
        ':',
        '^',
        '&',
        '!'
};
```

```
enum TYPE{
	IDENTIFIER,
	STRING_LITERAL,
	KEYWORD,
	NUMERIC_CONSTANT,
	BRACKET,
	OPERATOR,
	SPECIAL_SYMBOL,
	RELATIONAL_OPERATOR,
	CHARACTER_CONSTANT
};

char types[][30] = {
	"IDENTIFIER",
	"STRING_LITERAL",
	"KEYWORD",
	"NUMERIC_CONSTANT",
	"BRACKET",
	"OPERATOR",
	"SPECIAL_SYMBOL",
	"RELATIONAL_OPERATOR",
	"CHARACTER_CONSTANT"
};

typedef struct node{
	char *cur;
	int row,col;
	struct node *next;
	enum TYPE type;
}*node;

typedef struct symbol{
	char *name;
	char *dataType;
	struct symbol *next;
	unsigned int size;
}*symbol;

node hashTable[MAX_SIZE];
symbol symTable[MAX_SIZE];

void init(){
	for(int i = 0;i<MAX_SIZE;i++){
		hashTable[i] = NULL;
	}
}

int isKeyword(char buf[]){
	for(int i =0;i<32;i++){
		if(strcmp(buf,keywords[i]) == 0)
			return 1;
	}
```

```
	return 0;
}

int isDataType(char buf[]){
	for(int i = 0;i<4;i++){
		if(strcmp(buf,dataTypes[i]) == 0)
			return 1;
	}
	return 0;
}

int isOperator(char c){
	for(int i = 0;i<5;i++){
		if(operators[i] == c)
			return 1;
	}
	return 0;
}

int isSpecialChar(char c){
	for(int i = 0;i<12;i++){
		if(specialSymbols[i] == c)
			return 1;
	}
	return 0;
}

int isBracket(char c){
	for(int i = 0;i<6;i++){
		if(brackets[i] == c){
			return 1;
		}
	}
	return 0;
}

int hashing(char* s){
	int sum = 0;
	for(int i = 0; i < strlen(s); i++){
		sum += (int)s[i];
	}
	return (sum)%MAX_SIZE;
}

void displaySymbolTable(){
	printf("\tName\t|\tType\t|\tSize\t\n\n");

	for(int i = 0;i<MAX_SIZE;i++){
		if(symTable[i] == NULL)
			continue;
		else{
			symbol cur = symTable[i];
```

```
                while(cur){
                        printf("%10s\t|%10s\t|%10d\t\n",cur->name,cur->dataType,cur->size);
                        cur = cur->next;
                }
        }
    }
}

int searchSymTable(char id[]){
    int index = hashing(id);

    if(symTable[index] == NULL)
        return -1;
    symbol cur = symTable[index];
    int i = 0;
    while(cur != NULL){
        if(strcmp(id,cur->name) == 0){
            return i;
        }
        cur = cur->next;
        i++;
    }
    return -1;
}

int searchHashTable(char buf[]){
    int index = hashing(buf);
    if(hashTable[index] == NULL)
        return 0;
    node cur= hashTable[index];
    while(cur != NULL){
        if(strcmp(cur->cur,buf) == 0)
            return 1;
        cur = cur->next;
    }
    return 0 ;
}

void insertSym(char id[],char dataType[]){
    if(searchSymTable(id)==-1){
        symbol s = (symbol)malloc(sizeof(struct symbol));
        char *str = (char*)calloc(strlen(id)+1,sizeof(char));
        strcpy(str,id);
        s->name = str;
        s->next = NULL;
        char *type = (char*)calloc(strlen(dataType)+1,sizeof(char));
        strcpy(type,dataType);
        s->dataType = type;
        if(strcmp(dataType,"int") == 0)
            s->size = 4;
        else if(strcmp(dataType,"double") == 0)
            s->size = 8;
```

```
		else if(strcmp(dataType,"char") == 0)
			s->size = 1;
		else if(strcmp(dataType,"function") == 0)
			s->size = 0;
		else
			s->size = 4;

		int index = hashing(id);
		if(symTable[index] == NULL){
			symTable[index] = s;
			return;
		}
		symbol cur = symTable[index];
		while(cur->next !=NULL)
			cur = cur->next;
		cur ->next = s;
		return;
	}
}

void insertHash(char buf[],int row,int col,enum TYPE type){
	if(type == IDENTIFIER || searchHashTable(buf) == 0){
		printf("<%s | %d | %d | %s >\n",buf,row,col,types[type]);
		int index = hashing(buf);
		node n = (node)malloc(sizeof(struct node));
		char *str = (char *)calloc(strlen(buf) +1,sizeof(char));
		strcpy(str,buf);
		n->cur = str;
		n->next = NULL;
		n->row =row;
		n->col = col;
		n->type = type;
		if(hashTable[index] == NULL){
			hashTable[index] = n;
			return;
		}
		node cur = hashTable[index];
		while(cur->next != NULL){
			cur = cur->next;
		}
		cur->next = n;
		return;
	}
}

int main(){
	removeSpaceComment();
	int row = removePreprocess();
	init();


	enum TYPE type;
```

```
FILE *fin = fopen("pre_out.c","r");

if(!fin){
	printf("can't open file\n");
	exit(0);
}

char buf[100],dataTypeBuf[100],c = 0;
int i =0,globalCol=1,col,tempRow = row;


while(c != EOF){
	if(isalpha(c) != 0||c == '_'){
		buf[i++] = c;
		col = globalCol;
		while(isalpha(c) != 0 || c =='_' || isdigit(c) != 0){
			c = fgetc(fin);
			globalCol++;
			if(isalpha(c)!= 0 || c=='_'||isdigit(c) != 0)
				buf[i++] = c;
		}
		buf[i] = '\0';
		if(isDataType(buf) == 1){
			insertHash(buf,row,col-1,KEYWORD);
			strcpy(dataTypeBuf,buf);
		}
		else if(isKeyword(buf) == 1){
			insertHash(buf,row,col-1,KEYWORD);
		}
		else{
			insertHash(buf,row,col-1,IDENTIFIER);
			if(c == '(')
				insertSym(buf,"function");
			else
				insertSym(buf,dataTypeBuf);
			dataTypeBuf[0] = '\0';
		}
		i = 0;
		if(c == '\n')
			row++,globalCol=1;
		buf[0] = '\0';
	}
	else if(isdigit(c) != 0){
		buf[i++] = c;
		col = globalCol;
		while(isdigit(c) != 0 || c=='.'){
			c = fgetc(fin);
			globalCol++;
			if(isdigit(c) != 0||c == '.')
				buf[i++] = c;
		}
```

```
		buf[i] = '\0';
		insertHash(buf,row,col-1,NUMERIC_CONSTANT);
		i = 0;
		if(c == '\n')
			row++,globalCol=1;
		buf[0] = '\0';
	}
	else if(c == '\"'){
		col = globalCol;
		buf[i++] = c;
		c = 0;
		while(c != '\"'){
			c = fgetc(fin);
			globalCol++;
			buf[i++] = c;
		}
		buf[i] = '\0';
		insertHash(buf,row,col-1,STRING_LITERAL);
		buf[0] = '\0';
		i = 0;
		c = fgetc(fin);
		globalCol++;
	}
	else if(c == '\''){
		col = globalCol;
		buf[i++] = c;
		c = fgetc(fin);
		globalCol++;
		buf[i++] = c;
		if(c=='\\'){
			c = fgetc(fin);
			globalCol++;
			buf[i++] =c ;
		}
		c = fgetc(fin);
		globalCol++;
		buf[i++] = c;
		buf[i] = '\0';
		insertHash(buf,row,col-1,CHARACTER_CONSTANT);
		buf[0] = '\0';
		i = 0;
		c = fgetc(fin);
		globalCol++;
	}
	else{
		col = globalCol;
		if(c == '='){
			c = fgetc(fin);
			globalCol++;
			if(c == '='){
				insertHash("==",row,col-1,RELATIONAL_OPERATOR);
			}
```

```
		else{
			insertHash("=",row,col-1,RELATIONAL_OPERATOR);
			fseek(fin,-1,SEEK_CUR);
			globalCol--;
		}
	}
	else if(c == '<' || c == '>' || c == '!'){
		char temp = c;
		c = fgetc(fin);
		globalCol++;
		if(c == '='){
			char tempStr[3] = {temp,'=','\0'};
			insertHash(tempStr,row,col-1,RELATIONAL_OPERATOR);

		}
		else{
			char tempStr[2] = {temp,'\0'};
			insertHash(tempStr,row,col-1,RELATIONAL_OPERATOR);
			fseek(fin,-1,SEEK_CUR);
			globalCol--;
		}
	}
	else if(isBracket(c) == 1){
		char tempStr[2] = {c,'\0'};
		insertHash(tempStr,row,col-1,BRACKET);
	}
	else if(isSpecialChar(c) == 1){
		char tempStr[2] = {c,'\0'};
		insertHash(tempStr,row,col-1,SPECIAL_SYMBOL);
	}
	else if(isOperator(c) == 1){
		char temp = c;
		c = fgetc(fin);
		globalCol++;
		if(c == '=' || (temp == '+' && c == '+')||(temp == '-' &&  c == '-'))
		{
			char tempStr[3] = {temp,c,'\0'};
			insertHash(tempStr,row,col-1,OPERATOR);
		}
		else{
			char tempStr[2] = {temp,'\0'};
			insertHash(tempStr,row,col-1,OPERATOR);
			fseek(fin,-1,SEEK_CUR);
			globalCol--;
		}
	}
	else if(c=='\n'){
		row++;
		globalCol =1;
	}
	c = fgetc(fin);
	globalCol++;
```

```
                }
        }
        printf("\nSymbol Table:\n\n");
        displaySymbolTable();
        return 0;
}
```

clean.h

```
// Program to remove spaces and comments
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

int removeSpaceComment(){
        FILE *fin,*fout;
        int ca,cb;
        fin = fopen("input_program.c","r");
        if(!fin){
                printf("cannot open file\n");
                exit(0);
        }
        fout = fopen("space_out.c","w");
        ca = fgetc(fin);
        while(ca!=EOF){
                if(ca == ' ' || ca == '\t'){
                        putc(' ',fout);
                        while(ca == ' '||ca == '\t')
                                ca = fgetc(fin);
                }
                if(ca == '/'){
                        cb = fgetc(fin);
                        if(cb == '/'){
                                while(ca != '\n')
                                        ca = fgetc(fin);
                        }
                        else if(cb == '*'){
                                do{
                                        while(ca != '*')
                                                ca = fgetc(fin);
                                        ca = fgetc(fin);
                                }while(ca != '/');
                        }
                        else{
                                putc(ca,fout);
                                putc(cb,fout);
                        }
                }
                else
                        putc(ca,fout);
```

```
            ca = fgetc(fin);
        }
        fclose(fin);
        fclose(fout);
        return 0;
}
```

preprocesses.h

```
// Program to remove preprocessing directive. Function also returns the row number after removing the directives

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

const char *directives[] = {"#define","#include"};

int isDirective(const char *str){
        for(int i = 0;i<sizeof(directives)/sizeof(char*);i++){
            if(strncmp(str,directives[i],strlen(directives[i])) == 0){
                return 1;
            }

        }
        return 0;
}

int removePreprocess(){
        FILE *fa,*fb;
        char buff[2048];
        char* filename;
        int row = 1;
        // printf("enter filename to open:\n");
        // scanf("%s",filename);
    filename = "space_out.c";
        fa = fopen(filename,"r");

        fb = fopen("pre_out.c","w");
        if(!fa||!fb){
            printf("cannot open file\n");
            exit(0);
        }
        while(fgets(buff,2048,fa)){
            if(!isDirective(buff))
                fputs(buff,fb);
            else{
                row++;
                fputs("\n", fb);
            }
        }
```

```
        fclose(fa);
        fclose(fb);

}
```

input_program.c

```
#include <stdio.h>
int add(int a, int b){
   return a+b;
}

int main(){
   int a = 1;
   double buf;
   if(a % 2 == 1){  // Check if odd
      printf("Odd\n");
   }
   else{  // Else it is even
      printf("Even\n");
   }
   return 0;
}
```

```
ugcse@prg28:~/190905104_CD/lab4$ ./lexanal2
<int | 2 | 1 | KEYWORD >
<add | 2 | 5 | IDENTIFIER >
<( | 2 | 8 | BRACKET >
<a | 2 | 13 | IDENTIFIER >
<, | 2 | 14 | SPECIAL_SYMBOL >
<b | 2 | 20 | IDENTIFIER >
<) | 2 | 21 | BRACKET >
<{ | 2 | 22 | BRACKET >
<return | 3 | 2 | KEYWORD >
<a | 3 | 9 | IDENTIFIER >
<+ | 3 | 10 | OPERATOR >
<b | 3 | 11 | IDENTIFIER >
<; | 3 | 12 | SPECIAL_SYMBOL >
<} | 4 | 1 | BRACKET >
<main | 6 | 5 | IDENTIFIER >
<a | 7 | 6 | IDENTIFIER >
<= | 7 | 8 | RELATIONAL_OPERATOR >
<1 | 7 | 10 | NUMERIC_CONSTANT >
<double | 8 | 2 | KEYWORD >
<buf | 8 | 9 | IDENTIFIER >
<if | 9 | 2 | KEYWORD >
<a | 9 | 5 | IDENTIFIER >
<% | 9 | 7 | OPERATOR >
<2 | 9 | 9 | NUMERIC_CONSTANT >
<== | 9 | 11 | RELATIONAL_OPERATOR >
<printf | 9 | 19 | IDENTIFIER >
<"Odd\n" | 9 | 26 | STRING_LITERAL >
<else | 11 | 2 | KEYWORD >
<printf | 11 | 9 | IDENTIFIER >
<"Even\n" | 11 | 16 | STRING_LITERAL >
<0 | 13 | 9 | NUMERIC_CONSTANT >

Symbol Table:

        Name    |        Type     |        Size

      main      |  function       |          0
         a      |       int       |          4
         b      |       int       |          4
       buf      |    double       |          8
       add      |  function       |          0
    printf      |  function       |          0
ugcse@prg28:~/190905104_CD/lab4$
```