190905104
Parth Shukla
Lab3

1) Design a lexical analyzer which contains getNextToken( ) for a simple C program to create a structure of token each time and return, which includes row number, column number and token type. The tokens to be identified are arithmetic operators, relational operators, logical operators, special symbols, keywords, numerical constants, string literals and identifiers. Also, getNextToken() should ignore all the tokens when encountered inside single line or multiline comment or inside string literal. Preprocessor directive should also be stripped.

// input_program.c

```
/* Input program
for lexical analyser*/
#include <stdio.h>

int main(){
int a = 1;
if(a % 2 == 1){
   printf("Odd\n");
}
else{
   printf("Even\n");
}
return 0;
}
```

// getNextToken.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "clean_code.h"
#include "remove_comments.h"
#include "preprocess.h"

#define SIZE 100

char *keywords[] = {"printf", "int", "float", "return", "break", "continue", "if", "else", "for",
"while"};
char operators[5] = {'+','-','/','%','*'};
char brackets[6] = {'(',')','{','}','[',']'};

struct token{
         char *token_name;
         int index;
   int row, col;
         char *type;
};
```

```
struct token* hashtable[SIZE];
int size = 0;

int hashkey(int key){
   return key % SIZE;
}

int isKeyword(char buf[]){
         for(int i = 0; i< 10; i++){
                  if(strcmp(buf,keywords[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 isBracket(char c){
         for(int i = 0;i<6;i++){
                  if(brackets[i] == c) return 1;
         }
         return 0;
}

int searchHash(char buf[]){
         int index = hashkey(strlen(buf));
         if(hashtable[index] == NULL)
                  return 0;

   for(int i = 0; i < size; i++){
      struct token* cur = hashtable[i];
      if(cur == NULL) continue;
      if(strcmp(cur->token_name, buf) == 0)
         return 1;
   }
         return 0;
}

void insert(char buf[],int row,int col,int type){
   if(type == 1){  // Keyword
      if(searchHash(buf) == 0){
         printf("<%s,%d,%d>\n",buf,row,col);
         int index = hashkey(strlen(buf));
         struct token* temp = (struct token*)malloc(sizeof(struct token));
                           char *str = (char *)calloc(strlen(buf)+1,sizeof(char));
```

```
            strcpy(str, buf);
            temp->token_name = str;
            temp->row = row;
            temp->col = col;
            temp->index = index;
            temp->type = "KEYWORD";
        }
    }
    else{
        if(searchHash(buf) == 0){
            printf("<%s,%d,%d>\n",buf,row,col);
            int index = hashkey(strlen(buf));
            struct token* temp = (struct token*)malloc(sizeof(struct token));
                        char *str = (char *)calloc(strlen(buf)+1,sizeof(char));
            strcpy(str, buf);
            temp->token_name = str;
            temp->row = row;
            temp->col = col;
            temp->index = index;
            temp->type = "IDENTIFIER";
        }
    }

}

int main(){
    /*
    Flow: input_program.c -> cleanOut.c -> remCommentsOut.c -> pre_out.c
    */
    clean_code();
    remCom();
    preprocess();
        for(int i = 0;i<SIZE;i++)
                hashtable[i] = NULL;
        FILE *fin = fopen("pre_out.c","r");
        if(!fin){
                printf("can't find file");
                return 0;
        }
        char buf[100],c=0;
        int i = 0,row = 1,globalcols=1,col;
        while(c!=EOF){
                //keywords & identifiers
                if(isalpha(c) != 0 || c == '_'){
                        buf[i++] = c;
                        col = globalcols;
                        while(isalpha(c) != 0|| c=='_'||isdigit(c)!=0){
                                c = fgetc(fin);
                                globalcols++;
                                if(isalpha(c)!=0 || c == '_'||isdigit(c) != 0)
                                        buf[i++] =c;
                        }
                        buf[i] = '\0';
                        if(isKeyword(buf) == 1){
                                insert(buf,row,col,1);
                        }else{
                                insert("id",row,col,0);
                        }
                        i = 0;
                        if(c == '\n') row++,globalcols = 1;
                        buf[0] = '\0';
                }
                //numbers
                else if(isdigit(c) != 0){
                        buf[i++] = c;
                        col = globalcols;
                        while(isdigit(c)!=0 || c == '.'){
                                c = fgetc(fin);
                                globalcols++;
                                if(isdigit(c) !=0 || c == '.')
                                        buf[i++] = c;
                        }
                        buf[i] = '\0';
                        insert("num",row,col,0);
                        i = 0;
                        if(c == '\n') row++,globalcols = 1;
                        buf[0] = '\0';
                        c = fgetc(fin);
                        globalcols++;
                }
                //string literals
                else if(c == '\"'){
                        col = globalcols;
                        buf[i++] = c;
                        c = 0;
                        while(c != '\"'){
                                c = fgetc(fin);
                                globalcols++;
                                buf[i++] = c;
                        }
                        buf[i] = '\0';
                        insert(buf,row,col,0);
                        buf[0] = '\0';
                        i = 0;
                        c = fgetc(fin);
                        globalcols++;
                }
                else{
                        col = globalcols;
                        if(c == '='){
                                c = fgetc(fin);
                                globalcols++;
                                if(c == '=')
                                        insert("==",row,col,1);
```

```
					else{
						insert("=",row,col,1);
						fseek(fin,-1,SEEK_CUR);
						globalcols--;
					}
				}
				//relational operators
				if(c == '>' || c == '<' || c == '!'){
					c = fgetc(fin);
					globalcols++;
					if(c == '='){
						char temp[2] = {c,'='};
						insert(temp,row,col,1);
					}
					else{
						char temp[1] = {c};
						insert(temp,row,col,1);
						fseek(fin,-1,SEEK_CUR);
						globalcols--;
					}
				}
				//brackets & arithmetic operators
				if(isOperator(c) == 1 || isBracket(c) == 1){
					char temp[1] = {c};
					insert(temp,row,col,1);
				}
				if(c == '\n') row++,globalcols=1;
				c = fgetc(fin);
				globalcols++;
			}
	}
	printf("\n");
	return 0;
}
```

// clean_code.c

```
// Header file to remove tabs and spaces
#include <stdio.h>
#include <stdlib.h>


void clean_code(){
	FILE *fa,*fb;
   int ca, cb;
   char* fname;
   // printf("Enter file name\n");
   // scanf("%s", fname);
   fname = "input_program.c";
	fa = fopen(fname,"r");
	if(fa == NULL){
```

```
                        printf("Cannot open file \n");
                        exit(0);
                }
    fb = fopen("cleanOut.c", "w");

    ca = getc(fa);
    while(ca != EOF){
        if(ca == ' '){
            putc(' ', fb);
            while(ca == ' '){
                ca = getc(fa);
            }
            putc(ca, fb);
        }
        else{
            putc(ca, fb);
        }
        ca = getc(fa);
    }
    fclose(fa);
    fclose(fb);
}
```

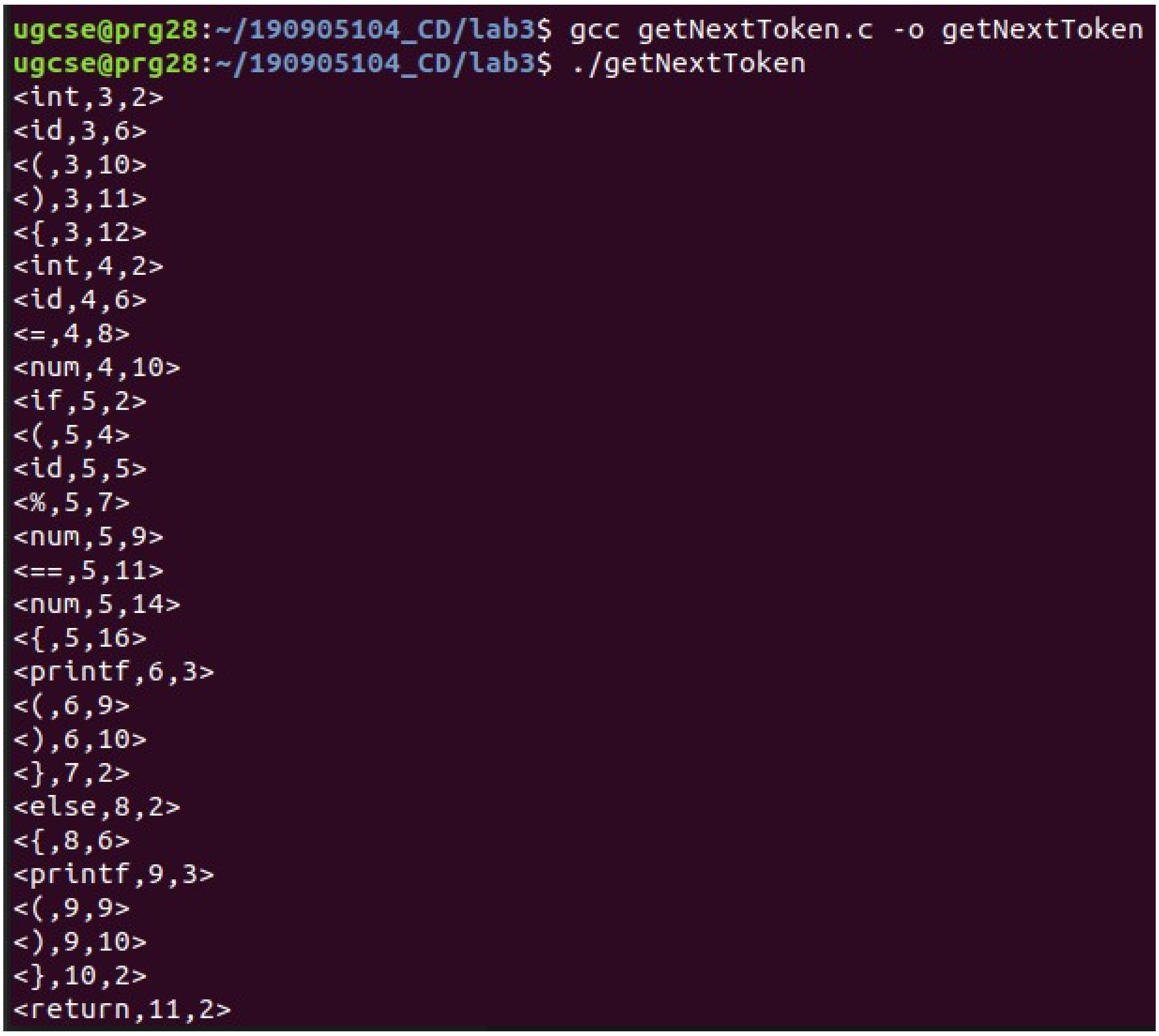
// remove_comments.h

```
// Program to remove single and multiline comments from a given C file
// Also, removing string literals

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

int remCom(){
            FILE *fa,*fb;
            int ca,cb;
    char* fname;
    // printf("Enter file name\n");
    // scanf("%s", fname);
    fname = "cleanOut.c";
            fa = fopen(fname,"r");
            if(fa == NULL){
                        printf("Cannot open file \n");
                        exit(0);
            }
            fb = fopen("remCommentsOut.c","w");
            ca = getc(fa);
            while(ca != EOF){
                        if(ca == ' '){
                                    putc(ca,fb);
                                    while(ca == ' ') ca = getc(fa);
                        }
                        if(ca == '/'){
                                    cb = getc(fa);
                        }
                                    if(cb == '/'){
                                        while(ca != '\n')
                                            ca = getc(fa);
                                    }
                                    else if(cb == '*'){
                                        do{
                                            while(ca !='*')
                                                ca = getc(fa);
                                            ca = getc(fa);
                                        }while(ca !='/');
                                    }
                                    else{
                                        putc(ca,fb);
                                        putc(cb,fb);
                                    }
                        }
                   else if(ca == '"'){
            ca = getc(fa);
            while(ca != '"'){
              ca = getc(fa);
            }
                   }
                        else putc(ca,fb);
                        ca = getc(fa);
            }
            fclose(fa);
            fclose(fb);
            return 0;
}
```

// preprocess.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 100

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 preprocess(){
    FILE *fa,*fb;
    char buff[2048];
```

```
	char* filename;
	// printf("enter filename to open:\n");
	// scanf("%s",filename);
    filename = "remCommentsOut.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);
	}
	fclose(fa);
	fclose(fb);

}
```



```
ugcse@prg28:~/190905104_CD/lab3$ gcc getNextToken.c -o getNextToken
ugcse@prg28:~/190905104_CD/lab3$ ./getNextToken
<int,3,2>
<id,3,6>
<(,3,10>
<),3,11>
<{,3,12>
<int,4,2>
<id,4,6>
<=,4,8>
<num,4,10>
<if,5,2>
<(,5,4>
<id,5,5>
<%,5,7>
<num,5,9>
<==,5,11>
<num,5,14>
<{,5,16>
<printf,6,3>
<(,6,9>
<),6,10>
<},7,2>
<else,8,2>
<{,8,6>
<printf,9,3>
<(,9,9>
<),9,10>
<},10,2>
<return,11,2>
```