Ayush Goyal
190905522 CSE D Roll 62

Compiler Design Lab 3: Construction of Token Generator

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.

Code(along with file names):

“spaces.h”:

```
int space(){
	FILE *fa, *fb;
	int ca, cb;
	fa = fopen("input.c", "r");
	if(fa == NULL){
		printf("Cannot open file!\n");
		exit(0);
	}
	fb = fopen("space_output.c", "w");
	ca = getc(fa);
	while(ca != EOF){
		if(ca == ' ' || ca == '\t'){
			putc(' ',fb);
			while(ca == ' ' || ca == '\t')
				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
                putc(ca, fb);
            ca = getc(fa);
        }
        fclose(fa);
        fclose(fb);
        return 0;
}
```

“preprocess.h”:

```
int process(){
        FILE *fin = fopen("space_output.c", "r");
        FILE *fout = fopen("process_output.c", "w+");
        char c = 0;
        char buffer[100];
        buffer[0] = '\0';
        int i = 0;
        char *includeStr = "include", *defineStr = "define", *mainStr = "main";
        int mainFlag = 0;
        while(c != EOF){
            c = fgetc(fin);
            if(c == '#' && mainFlag == 0){
                while(c!=' '){
                    c = fgetc(fin);
                    buffer[i++] = c;
                }
                buffer[i] = '\0';
                if(strstr(buffer, includeStr)!=NULL || strstr(buffer, defineStr)!=NULL){
                    while(c!='\n'){
                        c = fgetc(fin);
                    }
                }
                else{
                    fputc('#', fout);
                    for(int j = 0; j<i; j++){
                        fputc(buffer[j], fout);
                    }
                    while(c!='\n'){
                        c = fgetc(fin);
                        fputc(c, fout);
                    }
                }
                i = 0;
                buffer[0]= '\0';
            }
            else{
                if(mainFlag == 0){
                    buffer[i++] = c;
```

```
				buffer[i] = '\0';
				if(strstr(buffer, mainStr)!=NULL){
					mainFlag = 1;
				}
			}
			if(c == ' ' || c == '\n'){
				buffer[0] = '\0';
				i = 0;
			}
			fputc(c, fout);
		}
	}
	fclose(fin);
	fclose(fout);
	return 0;
}
```

“getNextToken.c”:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
#include "spaces.h"
#include "preprocess.h"
#define MAX_SIZE 20

char keywords[32][10] = {"auto", "double", "int", "struct",
"break", "else", "long", "switch", "case", "enum", "register",
"typedef", "char", "extern", "return", "union", "const", "float",
"short", "unsigned", "continue", "for", "signed", "void",
"default", "goto", "sizeof", "voltile", "do", "if", "static",
"while"}; // list of keywords

char operators[5] = {'+','-','/','%','*'};
char brackets[6] = {'(', ')', '[', ']', '{', '}'};

//element for hash table created:

typedef struct node{
	char *cur;
	int row,col;
	struct node *next;
} *Node;

Node hashTable[MAX_SIZE]; //hash table created

//to check for keyword
int compare(char buffer[]){
	for(int i=0;i<32;i++){
		if(strcmp(buffer, keywords[i]) == 0){
				return 1;
			}
		}
		return 0;
}

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

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

//creating a hashing function
int hash(int size){
		return (size % MAX_SIZE);
}

//searching in the hash table
int search(char buffer[]){
		int index = hash(strlen(buffer));
		if(hashTable[index] == NULL)
			return 0;
		Node cur = hashTable[index];
		while(cur != NULL){
			if(strcmp(cur->cur, buffer) == 0)
				return 1;
			cur = cur->next;
		}
		return 0;
}

//inserting elements in the hash table
void insert(char buffer[], int row, int col, int type){
		if(type == 1){
			if(search(buffer) == 0){
				printf("<%s,%d,%d>\n", buffer, row, col);
				int index = hash(strlen(buffer));
				Node n = (Node)malloc(sizeof(struct node));
				char *str = (char *)calloc(strlen(buffer) + 1, sizeof(char));
				strcpy(str, buffer);
				n->cur = str;
				n->next = NULL;
```

```
                    n->row = row;
                    n->col = col;
                    if(hashTable[index] == NULL){
                        hashTable[index] = n;
                        return;
                    }
                    Node cur = hashTable[index];
                    while(cur->next != NULL){
                        cur = cur->next;
                    }
                    cur->next = n;
                }
        }
        else{
                printf("<%s,%d,%d>\n", buffer, row, col);
                int index = hash(strlen(buffer));
                Node n = (Node)malloc(sizeof(struct node));
                char *str = (char *)calloc(strlen(buffer) + 1, sizeof(char));
                strcpy(str, buffer);
                n->cur = str;
                n->next = NULL;
                n->row = row;
                n->col = col;
                if(hashTable[index] == NULL){
                    hashTable[index] = n;
                    return;
                }
                Node cur = hashTable[index];
                while(cur->next != NULL){
                    cur = cur->next;
                }
                cur->next = n;
        }
}


int main(){
        //executing space function from spaces.h
        space();
        //executing process function from preprocess.h
        process();

        for(int i=0;i<MAX_SIZE;i++)
                hashTable[i] = NULL;

        FILE *fin = fopen("process_output.c", "r");
        if(fin == NULL){
                printf("Cannot open file!\n");
                return 0;
        }
        char buffer[100], c = 0;
        int i = 0, row = 1, col_global = 1, col;
```

```
while(c != EOF){
        if(isalpha(c) != 0 || c == '_'){
                buffer[i++] = c;
                col = col_global;
                while(isalpha(c) != 0 || c == '_' || isdigit(c) != 0){
                        c = fgetc(fin);
                        col_global++;
                        if (isalpha(c) != 0 || c == '_' || isdigit(c) != 0)
                                buffer[i++] = c;
                }
                buffer[i] = '\0';
                if(compare(buffer) == 1){
                        insert(buffer, row, col, 1); // keyword
                }
                else{
                        insert("id", row, col, 0); // identifier
                }
                i = 0;
                if(c == '\n')
                        row++, col_global = 1;
                buffer[0] = '\0';
        }
        else if(isdigit(c) != 0){
                buffer[i++] = c;
                col = col_global;
                while(isdigit(c) != 0 || c == '.'){
                        c = fgetc(fin);
                        col_global++;
                        if(isdigit(c) != 0 || c == '.')
                                buffer[i++] = c;
                }
                buffer[i] = '\0';
                insert("num", row, col, 0); // numerical constant
                i = 0;
                if(c == '\n')
                        row++, col_global = 1;
                buffer[0] = '\0';
                c = fgetc(fin);
                col_global++;
        }
        else if(c == '\"'){
                col = col_global;
                buffer[i++] = c;
                c = 0;
                while(c != '\"'){
                        c = fgetc(fin);
                        col_global++;
                        buffer[i++] = c;
                }
                buffer[i] = '\0';
                insert(buffer, row, col, 0); // string literals
                buffer[0] = '\0';
```

```
				i = 0;
				c = fgetc(fin);
				col_global++;
			}
			else{
				col = col_global;
				if(c == '='){ // relational and logical operators
					c = fgetc(fin);
					col_global++;
					if(c == '='){
						insert("==", row, col, 1);
					}
					else{
						insert("=", row, col, 1);
						fseek(fin, -1, SEEK_CUR);
						col_global--;
					}
				}
				if(c == '>' || c == '<' || c == '!'){
					c = fgetc(fin);
					col_global++;
					if(c == '='){
						char temp_str[2] = {c, '='};
						insert(temp_str, row, col, 1);
					}
					else{
						char temp_str[1] = {c};
						insert(temp_str, row, col, 1);
						fseek(fin, -1, SEEK_CUR);
						col_global--;
					}
				}
				if(isOperator(c) == 1 || isBracket(c) == 1){ //parentheses
					char temp_string[1] = {c};
					insert(temp_string, row, col, 1);
				}
				if(c == '\n')
					row++, col_global = 1;
				c = fgetc(fin);
				col_global++;
			}
		}
		return 0;
}
```

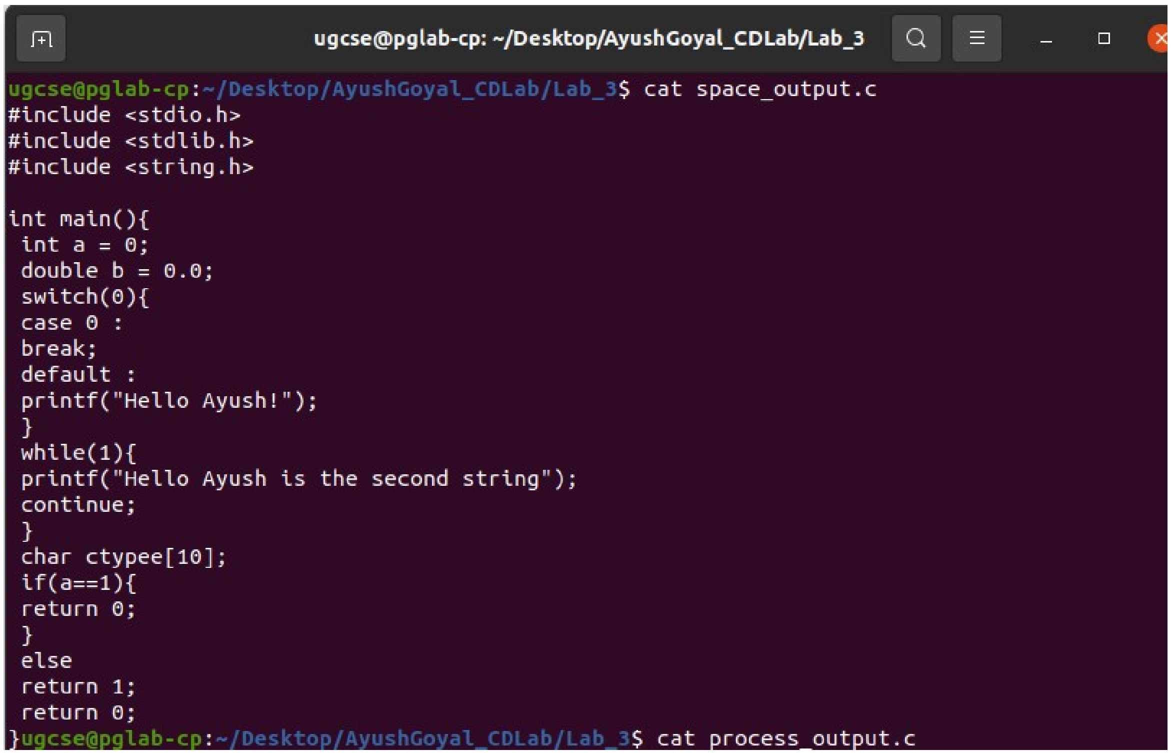
I have made a customized “input.c” file to test the program output which is executed and shown below. The “input.c” file is shown below but the intermediate files “space_output” and “process_output” are shown in the output screenshots:

“input.c”:

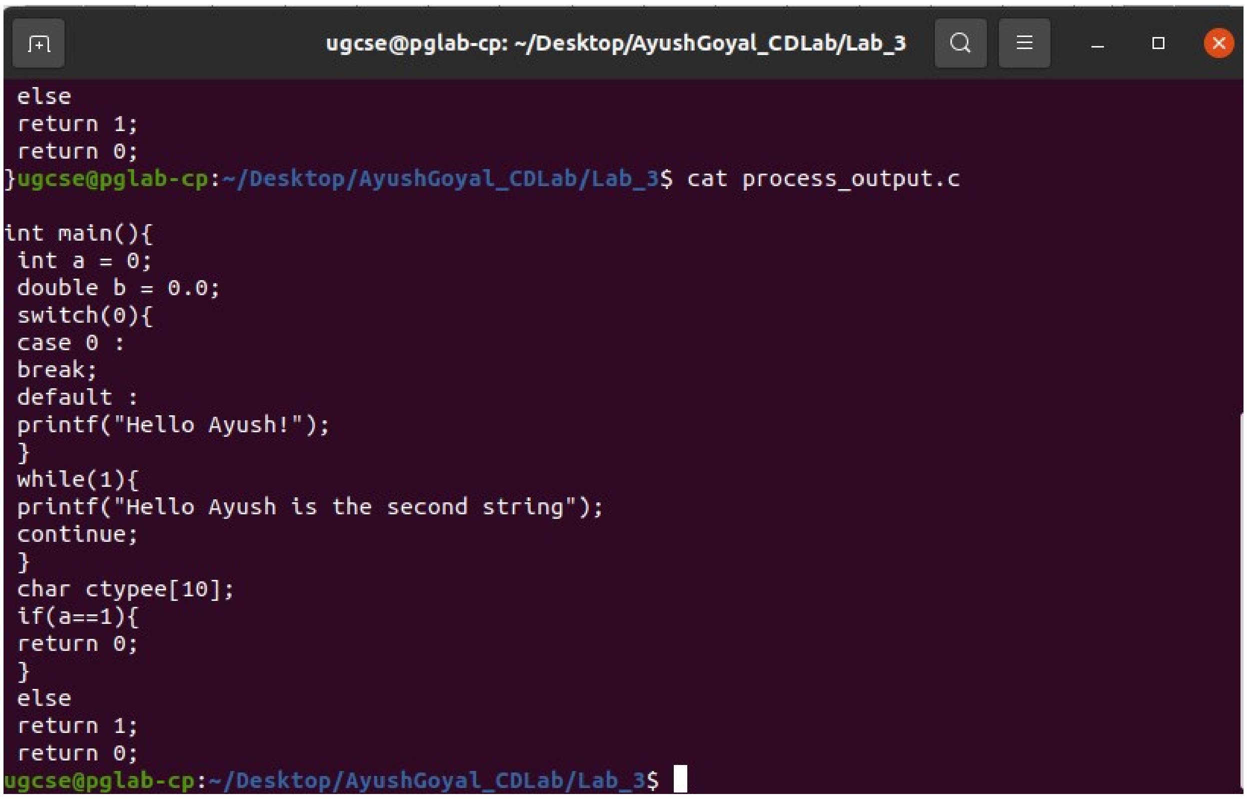
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(){
	int a = 0;
	double b = 0.0;
	switch(0){
		case 0 :
					break;
		default :
					printf("Hello Ayush!");
	}
	while(1){
		printf("Hello Ayush is the second string");
		continue;
	}
	char ctypee[10];
	if(a==1){
		return 0;
	}
	else
		return 1;
	return 0;
}
```

Output:

Intermediate files are as shown below:

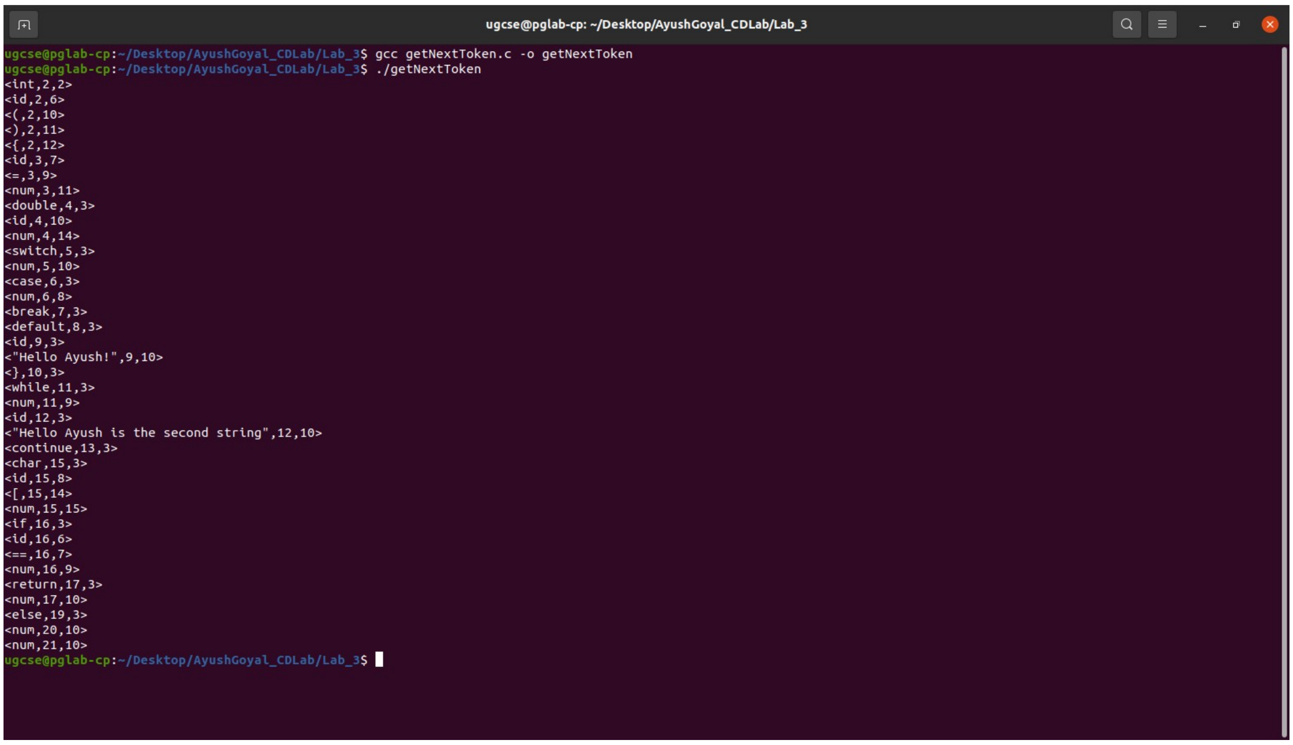


```
ugcse@pglab-cp: ~/Desktop/AyushGoyal_CDLab/Lab_3
ugcse@pglab-cp:~/Desktop/AyushGoyal_CDLab/Lab_3$ cat space_output.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(){
 int a = 0;
 double b = 0.0;
 switch(0){
 case 0 :
 break;
 default :
 printf("Hello Ayush!");
 }
 while(1){
 printf("Hello Ayush is the second string");
 continue;
 }
 char ctypee[10];
 if(a==1){
 return 0;
 }
 else
 return 1;
 return 0;
}ugcse@pglab-cp:~/Desktop/AyushGoyal_CDLab/Lab_3$ cat process_output.c
```



```
else
return 1;
return 0;
}ugcse@pglab-cp:~/Desktop/AyushGoyal_CDLab/Lab_3$ cat process_output.c

int main(){
 int a = 0;
 double b = 0.0;
 switch(0){
 case 0 :
 break;
 default :
 printf("Hello Ayush!");
 }
 while(1){
 printf("Hello Ayush is the second string");
 continue;
 }
 char ctypee[10];
 if(a==1){
 return 0;
 }
 else
 return 1;
 return 0;
ugcse@pglab-cp:~/Desktop/AyushGoyal_CDLab/Lab_3$
```

The Output in which tokens are generated:



```
ugcse@pglab-cp:~/Desktop/AyushGoyal_CDLab/Lab_3$ gcc getNextToken.c -o getNextToken
ugcse@pglab-cp:~/Desktop/AyushGoyal_CDLab/Lab_3$ ./getNextToken
<int,2,2>
<id,2,6>
<(,2,10>
<),2,11>
<{,2,12>
<id,3,7>
<=,3,9>
<num,3,11>
<double,4,3>
<id,4,10>
<num,4,14>
<switch,5,3>
<num,5,10>
<case,6,3>
<num,6,8>
<break,7,3>
<default,8,3>
<id,9,3>
<"Hello Ayush!",9,10>
<},10,3>
<while,11,3>
<num,11,9>
<id,12,3>
<"Hello Ayush is the second string",12,10>
<continue,13,3>
<char,15,3>
<id,15,8>
<[,15,14>
<num,15,15>
<if,16,3>
<id,16,6>
<==,16,7>
<num,16,9>
<return,17,3>
<num,17,10>
<else,19,3>
<num,20,10>
<num,21,10>
ugcse@pglab-cp:~/Desktop/AyushGoyal_CDLab/Lab_3$
```

THE END