Lab 7

Sahil Saini Salaria
Reg No. 180905048
Roll no. 11C

```
//header File getNextToken
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>

struct token{
	char lexeme[64];
	int row,col;
	char type[20];
};

static int row=1,col=1;
char buff[2048];
const char specialsymbols[]={'?',';',':',','};
const char
*keywords[]={"const","char","int","return","for","while","do","switch","if","else","unsigned","case
","break"};
const char *datatypes[]={"int","char","void","float","bool"};

const char arithmeticsymbols[]={'*'};

int isdatatype(char *w){
	int i;
	for(i=0;i<sizeof(datatypes)/sizeof(char *);i++){
		if(strcmp(w,datatypes[i])==0){
			return 1;
		}
	}
	return 0;
}

int iskeyword(char *str){
	for(int i=0;i<sizeof(keywords)/sizeof(char *);i++){
		if(strcmp(str,keywords[i])==0){
			return 1;
		}
	}
	return 0;
}

int charbelongsto(int c, const char *arr){
	int len;
	if(arr==specialsymbols){
		len=sizeof(specialsymbols)/sizeof(char);
	}else if(arr==arithmeticsymbols){
		len=sizeof(arithmeticsymbols)/sizeof(char);
	}
	for(int i=0;i<len;i++){
		if(c==arr[i]){
			return 1;
		}
	}
	return 0;
}


void filltoken(struct token *tkn, char c, int row, int col, char *type){
	tkn->row=row;
	tkn->col=col;
	strcpy(tkn->type,type);
	tkn->lexeme[0]=c;
	tkn->lexeme[1]='\0';
}

void newline(){
	row++;
	col=1;

}

struct token getnexttoken(FILE *f1){
	int c;
	struct token tkn={
		row=-1
	};
	int gottoken=0;

	while(!gottoken &&(c=fgetc(f1))!=EOF){
		if(charbelongsto(c,specialsymbols)){
			filltoken(&tkn,c,row,col,"specialsymbols");
			gottoken=1;
			col++;
		}

		else if(charbelongsto(c,arithmeticsymbols)){
			filltoken(&tkn,c,row,col,"arithmeticoperator");
			gottoken=1;
			col++;
		}

		else if(c=='('){
			filltoken(&tkn,c,row,col,"leftbracket");
			gottoken=1;
			col++;
		}
		else if(c==')'){
```

```
		filltoken(&tkn,c,row,col,"rightbracket");
		gottoken=1;
		col++;
	}
	else if(c=='{'){
		filltoken(&tkn,c,row,col,"left curly");
		gottoken=1;
		col++;
	}
	else if(c=='}'){
		filltoken(&tkn,c,row,col,"right curly");
		gottoken=1;
		col++;
	}
	else if(c=='+'){
		int d=fgetc(f1);
		if(d!='+'){
			filltoken(&tkn,c,row,col,"arithmeticoperator");
			gottoken=1;
			col++;
			fseek(f1,-1,SEEK_CUR);
		}else{
			filltoken(&tkn,c,row,col,"unary coperator");
			strcpy(tkn.lexeme,"++");
			gottoken=1;
			col+=2;

		}
	}
	else if(c=='+'){
		int d=fgetc(f1);
		if(d!='-'){
			filltoken(&tkn,c,row,col,"arithmeticoperator");
			gottoken=1;
			col++;
			fseek(f1,-1,SEEK_CUR);
		}else{
			filltoken(&tkn,c,row,col,"unary operator");
			strcpy(tkn.lexeme,"--");
			gottoken=1;
			col+=2;

		}
	}
	else if(c=='='){
		int d=fgetc(f1);
		if(d!='-'){
			filltoken(&tkn,c,row,col,"arithmeticoperator");
			gottoken=1;
			col++;
			fseek(f1,-1,SEEK_CUR);
		}else{
				filltoken(&tkn,c,row,col,"relational operator");
				strcpy(tkn.lexeme,"==");
				gottoken=1;
				col+=2;

			}
		}
		else if(isdigit(c)){
			tkn.row=row;
			tkn.col=col;
			tkn.lexeme[0]=c;
			int k=1;
			while((c=fgetc(f1))!=EOF&&isdigit(c)){
				tkn.lexeme[k++]=c;
				col++;
			}
			tkn.lexeme[k]='\0';
			strcpy(tkn.type,"number");
			gottoken=1;
			fseek(f1,-1,SEEK_CUR);
		}

		else if(c=='#'){
			while((c=fgetc(f1))!=EOF&&c!='\n');
			newline();

		}

		else if(c=='\n'){
			newline();
			c=fgetc(f1);
			if(c=='#'){
				while((c=fgetc(f1))!=EOF&&c!='\n');
				newline();

			}

			else if(c!=EOF){
				fseek(f1,-1,SEEK_CUR);
			}
		}

		else if(isspace(c)){
			col++;
		}

		else if(isalpha(c)||c=='_'){
			tkn.row=row;
			tkn.col=col++;
			tkn.lexeme[0]=c;
			int k=1;
			while((c=fgetc(f1))!=EOF && isalnum(c)){
```

```
                tkn.lexeme[k++]=c;
                col++;
        }
        tkn.lexeme[k]='\0';
        if(iskeyword(tkn.lexeme)){
                strcpy(tkn.type,"keyword");
        }else{
                strcpy(tkn.type,"identifier");
        }
        gottoken=1;
        fseek(f1,-1,SEEK_CUR);
}

else if(c=='/'){
        int d=fgetc(f1);
        col++;
        if(d=='/'){
                while((c=fgetc(f1))!=EOF&&c!='\n'){
                        col++;
                }
                if(c=='\n'){
                        newline();
                }
        }else if(d=='*'){
                do{
                        if(d=='\n'){
                                newline();
                        }
                        while((c=fgetc(f1))!=EOF &&c!='*'){
                                col++;
                                if(c=='\n'){
                                        newline();
                                }
                        }
                        col++;
                }
                while((d=fgetc(f1))!=EOF &&d!='/' &&col++);
                        col++;


        }else{
                filltoken(&tkn,c,row,col--,"arithmeticoperator");
                gottoken=1;
                fseek(f1,-1,SEEK_CUR);
        }
}
else if(c=='"'){
        tkn.row=row;
        tkn.col=col;
        strcpy(tkn.type,"String literal");
        int k=1;
        tkn.lexeme[0]='"';
```

```
                while((c=fgetc(f1))!=EOF &&c!='"'){
                        tkn.lexeme[k++]=c;
                        col++;
                }
                tkn.lexeme[k]='"';
                gottoken=1;
        }

        else if(c=='<' || c=='>' || c=='!'){
                filltoken(&tkn,c,row,col,"relational operator");
                col++;
                int d=fgetc(f1);
                if(d=='='){
                        col++;
                        strcat(tkn.lexeme,"=");
                }
                else{
                        if(c=='!'){
                                strcpy(tkn.type,"logical operator");
                        }
                        fseek(f1,-1,SEEK_CUR);
                }
                        gottoken=1;

        }

        else if(c=='&'||c=='|'){
                int d=fgetc(f1);
                if(c==d){
                        tkn.lexeme[0]=tkn.lexeme[1]=c;
                        tkn.lexeme[2]='\0';
                        tkn.row=row;
                        tkn.col=col;
                        col++;
                        gottoken=1;
                        strcpy(tkn.type,"logical operator");
                }else{
                        fseek(f1,-1,SEEK_CUR);
                }
                col++;
        }
        else{
                col++;
        }
    }
        return tkn;


}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "nextToken.h"


void program();
void declarations();
void data_type();
void identifier_list();
void assign_stat();

struct token curr;
FILE *f1;
void invalid(){
	printf("error");
	exit(0);
}
void program()
{

	if(strcmp(curr.lexeme,"main")==0)
	{
		curr=getnexttoken(f1);
		if(strcmp(curr.lexeme,"(")==0)
		{
			curr=getnexttoken(f1);
			if(strcmp(curr.lexeme,")")==0)
			{
				curr=getnexttoken(f1);;
				if(strcmp(curr.lexeme,"{")==0)
				{
					curr=getnexttoken(f1);
					declarations();
					assign_stat();
					if(strcmp(curr.lexeme,"}")==0)
					{
						return;
					}
					else
				{
					printf("\nMissing } at row:%d and col:%d.\n\
n",curr.row,curr.col);
					exit(0);
				}
				}
				else
				{
					printf("\nMissing { at row:%d and col:%d.\n\
n",curr.row,curr.col);
					exit(0);
```

```
                                        }
                              }
                              else
                                        {
                                                  printf("\nMissing ) at row:%d and col:%d.\n\
n",curr.row,curr.col);
                                                  exit(0);
                                        }
                    }
                    else
                                        {
                                                  printf("\nMissing ( at row:%d and col:%d.\n\
n",curr.row,curr.col);
                                                  exit(0);
                                        }
          }
          else
                                        {
                                                  printf("\nMissing main function\n\n");
                                                  exit(0);
                                        }
}

void declarations()
{

          if(isdatatype(curr.lexeme)==0)
          {
                    return;
          }
          data_type();
          identifier_list();
          if(strcmp(curr.lexeme,";")==0)
          {
                    curr=getnexttoken(f1);
                    declarations();
          }
          else {printf("\nMissing ; at row:%d and col:%d.\n\n",curr.row,curr.col); exit(0);}


}

void data_type()
{
          if(strcmp(curr.lexeme,"int")==0)
          {
                    curr=getnexttoken(f1);
                    return;
          }
          else if(strcmp(curr.lexeme,"char")==0)
          {
                    curr=getnexttoken(f1);
```

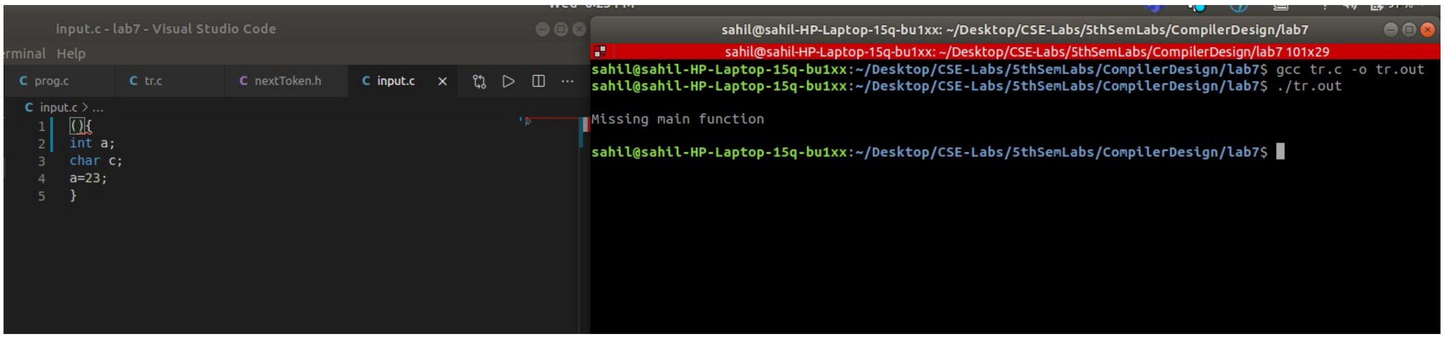
```
                return;
        }
        else
                                {
                                        printf("\nMissing data type at row:%d and col:%d.\n\
n",curr.row,curr.col);
                                        exit(0);
                                }
}

void identifier_list()
{
        if(strcmp(curr.type,"identifier")==0)
        {
                curr=getnexttoken(f1);
                if(strcmp(curr.lexeme,",")==0)
                {
                        curr=getnexttoken(f1);
                        identifier_list();
                }
                else return;
        }

        else
                {
                        printf("\nMissing identifier at row:%d and col:%d.\n\n",curr.row,curr.col);
                        exit(0);
                }
}

void assign_stat()
{
        if(strcmp(curr.type,"identifier")==0)
        {
                curr=getnexttoken(f1);
                if(strcmp(curr.lexeme,"=")==0)
                {
                        curr=getnexttoken(f1);
                        if(strcmp(curr.type,"identifier")==0)
                        {
                                curr=getnexttoken(f1);
                                if(strcmp(curr.lexeme,";")==0)
                                {
                                        curr=getnexttoken(f1);
                                        return;
                                }
                        }

                else if(strcmp(curr.type,"number")==0)
                {
                        curr=getnexttoken(f1);
                                if(strcmp(curr.lexeme,";")==0)
```

```
				{
					curr=getnexttoken(f1);
					return;
				}
				else
				{
					printf("\nMissing ; at row:%d and col:%d.\n\
n",curr.row,curr.col);
					exit(0);
				}

			}
			else
				{
					printf("\nMissing identifier at row:%d and col:%d.\n\
n",curr.row,curr.col);
					exit(0);
				}
			}
			else
				{
					printf("\nMissing = at row:%d and col:%d.\n\
n",curr.row,curr.col);
					exit(0);
				}
	}
	else
				{
					printf("\nMissing identifier at row:%d and col:%d.\n\
n",curr.row,curr.col);
					exit(0);
				}
}

int main()
{
	FILE *fa, *fb;
	int ca, cb;
	fa = fopen("input.c", "r");
	if (fa == NULL){
		// printf("hii");
		printf("Cannot open file \n");
		return 0;
	}
	fb = fopen("output.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 putc(ca,fb);
        ca = getc(fa);
}
fclose(fa);
fclose(fb);
fa = fopen("output.c", "r");
if(fa == NULL){
        printf("Cannot open file");
        return 0;
}
fb = fopen("input", "w");
ca = getc(fa);
while(ca != EOF){
        if(ca == '#'){
                while(ca != '\n'){
                        ca = getc(fa);
                }
        }
        ca = getc(fa);
        if(ca != EOF && ca != '#'){
                putc(ca, fb);
        }
}
fclose(fa);
fclose(fb);

fa = fopen("input.c", "r");
fb = fopen("output.c", "w");
ca = getc(fa);
```

```
int a,b;
        while(ca != EOF){
                putc(ca, fb);
                ca = getc(fa);
        }
        fclose(fa);
        fclose(fb);
        // remove("sample1.c");
        f1=fopen("output.c","r");
        if(f1==NULL)
          {
                printf("Error! File cannot be opened!\n");
                return 0;
          }
        struct token tkn;
        curr=getnexttoken(f1);
        program();
        printf("\nCompiled\n\n");
   fclose(f1);
}
```

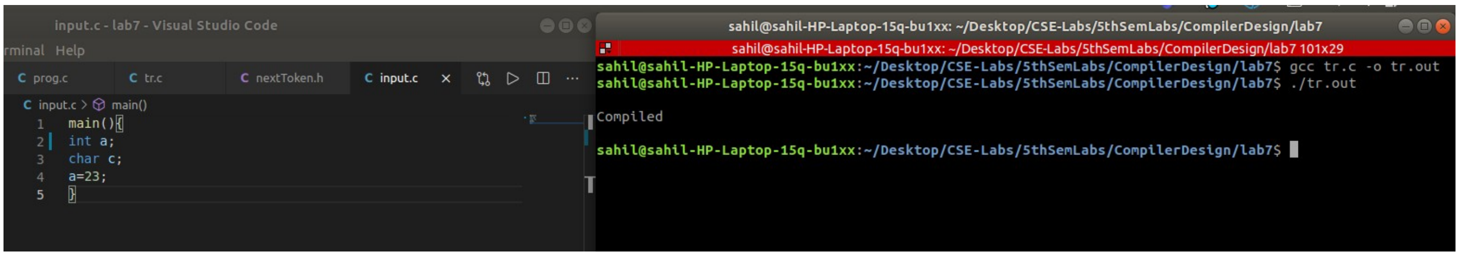
Test Case 1:


input.c - lab7 - Visual Studio Code
rminal Help
prog.c
tr.c
nextToken.h
input.c
input.c > ...
sahil@sahil-HP-Laptop-15q-bu1xx: ~/Desktop/CSE-Labs/5thSemLabs/CompilerDesign/lab7
sahil@sahil-HP-Laptop-15q-bu1xx: ~/Desktop/CSE-Labs/5thSemLabs/CompilerDesign/lab7 101x29

```
(){
int a;
char c;
a=23;
}
```

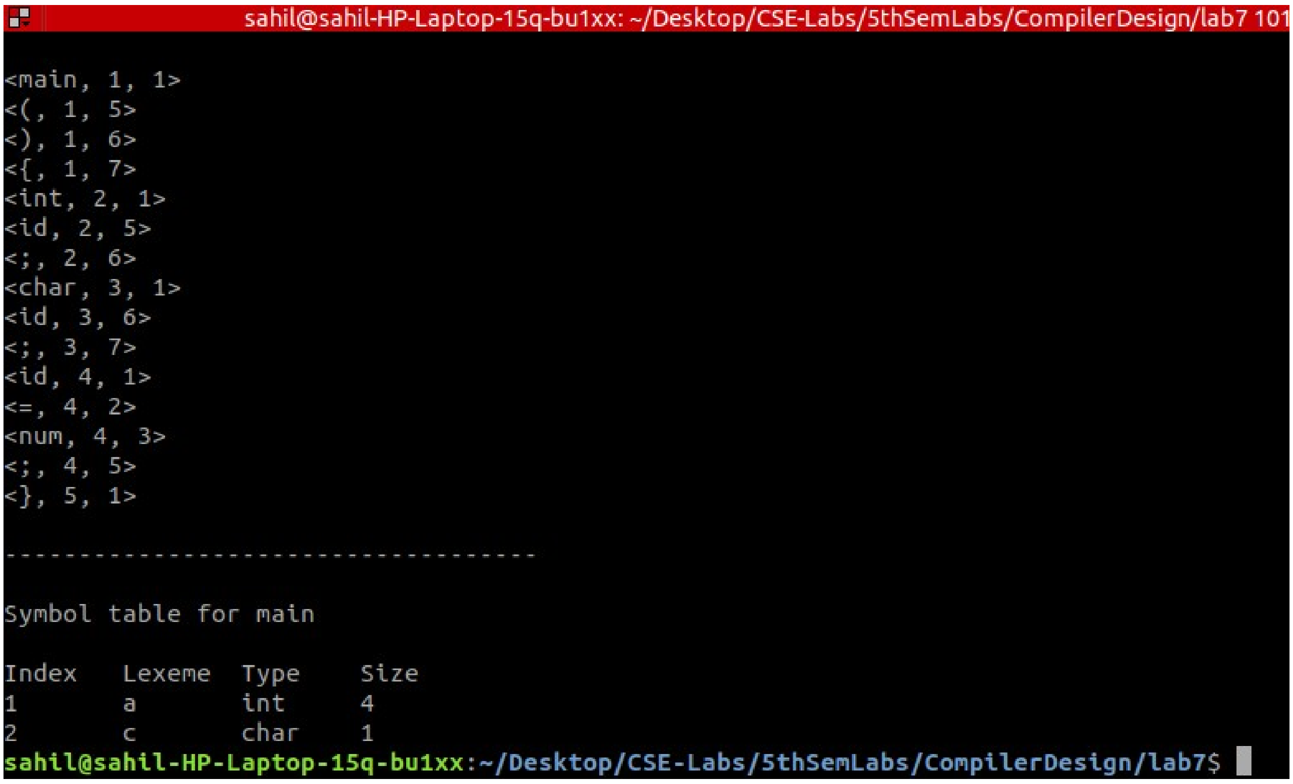
```
sahil@sahil-HP-Laptop-15q-bu1xx:~/Desktop/CSE-Labs/5thSemLabs/CompilerDesign/lab7$ gcc tr.c -o tr.out
sahil@sahil-HP-Laptop-15q-bu1xx:~/Desktop/CSE-Labs/5thSemLabs/CompilerDesign/lab7$ ./tr.out

Missing main function

sahil@sahil-HP-Laptop-15q-bu1xx:~/Desktop/CSE-Labs/5thSemLabs/CompilerDesign/lab7$
```

Test Case 2:


input.c - lab7 - Visual Studio Code
rminal Help
prog.c
tr.c
nextToken.h
input.c
input.c > main()
sahil@sahil-HP-Laptop-15q-bu1xx: ~/Desktop/CSE-Labs/5thSemLabs/CompilerDesign/lab7
sahil@sahil-HP-Laptop-15q-bu1xx: ~/Desktop/CSE-Labs/5thSemLabs/CompilerDesign/lab7 101x29

```
main(){
int a;
char c;
a=23;
}
```

```
sahil@sahil-HP-Laptop-15q-bu1xx:~/Desktop/CSE-Labs/5thSemLabs/CompilerDesign/lab7$ gcc tr.c -o tr.out
sahil@sahil-HP-Laptop-15q-bu1xx:~/Desktop/CSE-Labs/5thSemLabs/CompilerDesign/lab7$ ./tr.out

Compiled

sahil@sahil-HP-Laptop-15q-bu1xx:~/Desktop/CSE-Labs/5thSemLabs/CompilerDesign/lab7$
```


sahil@sahil-HP-Laptop-15q-bu1xx: ~/Desktop/CSE-Labs/5thSemLabs/CompilerDesign/lab7 101

```
<main, 1, 1>
<(, 1, 5>
<), 1, 6>
<{, 1, 7>
<int, 2, 1>
<id, 2, 5>
<;, 2, 6>
<char, 3, 1>
<id, 3, 6>
<;, 3, 7>
<id, 4, 1>
<=, 4, 2>
<num, 4, 3>
<;, 4, 5>
<}, 5, 1>

------------------------------------

Symbol table for main

Index   Lexeme  Type    Size
1       a       int     4
2       c       char    1
sahil@sahil-HP-Laptop-15q-bu1xx:~/Desktop/CSE-Labs/5thSemLabs/CompilerDesign/lab7$ 
```

//////////////////////////////////////END///