Compiler Design Lab 3

Dipesh Singh – 190905520

Question 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.

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 *finp = 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(finp);
          if(c == '#' && mainFlag == 0){
               while(c!=' '){
                    c = fgetc(finp);
                    buffer[i++] = c;
               }
               buffer[i] = '\0';
               if( strstr(buffer, includeStr)!=NULL ||
strstr(buffer, defineStr)!=NULL){
                    while(c!='\n'){
                         c = fgetc(finp);
                    }
               }
               else{
                    fputc('#', fout);
                    for(int j = 0; j<i; j++){
                         fputc(buffer[j], fout);
                    }
                    while(c!='\n'){
                         c = fgetc(finp);
                         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(finp);
     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] = {'(', ')', '[', ']', '{', '}'};

typedef struct node
{
 char *cur;
 int row, col;
 struct node *next;
} * Node; // element for hash table

Node hashTable[MAX_SIZE]; // hash table

int compare(char buffer[]) // function to check for keyword
{
 for (int i = 0; i < 32; i++)
 {
          if (strcmp(buffer, 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 hash(int size) // hashing function
{
 return (size) % MAX_SIZE;
}

int search(char buffer[]) // to search in hash table
{
 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;
}

void insert(char buffer[], int row, int col, int type)
{ // insert in hash table
     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()
{
 space(); // from spaces.h
 process(); // from preprocess.h
 for (int i = 0; i < MAX_SIZE; i++)
           hashTable[i] = NULL;
 FILE *finp = fopen("process_output.c", "r");
 if (finp == NULL)
 {
           printf("Cannot Find file, exiting ... ");
           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(finp);
```

```
                    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(finp);
                    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(finp);
               col_global++;
          }
          else if(c == '\"')
          {
               col = col_global;
               buffer[i++] = c;
               c = 0;
               while(c != '\"')
               {
                    c = fgetc(finp);
                    col_global++;
                    buffer[i++] = c;
               }
               buffer[i] = '\0';
               insert(buffer, row, col, 0); // string literals
               buffer[0] = '\0';
               i = 0;
               c = fgetc(finp);
               col_global++;
          }
          else
          {
               col = col_global;
               if(c == '='){ // relational and logical operators
                    c = fgetc(finp);
                    col_global++;
                    if(c == '=')
                    {
                         insert("==", row, col, 1);
                    }
                    else
                    {
                         insert("=", row, col, 1);
                         fseek(finp, -1, SEEK_CUR);
                         col_global--;
                    }
               }
               if(c == '>' || c == '<' || c == '!')
               {
                    c = fgetc(finp);
                    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(finp, -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(finp);
               col_global++;
          }

 }
 return 0;
}
```

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 world");
     }
     while(1){
          printf("hello world this is the second string");
          continue;
     }
     char ctypee[10];
     if(a==1){return 0;}
     else return 1;
     return 0;
}
```

space_output.c :

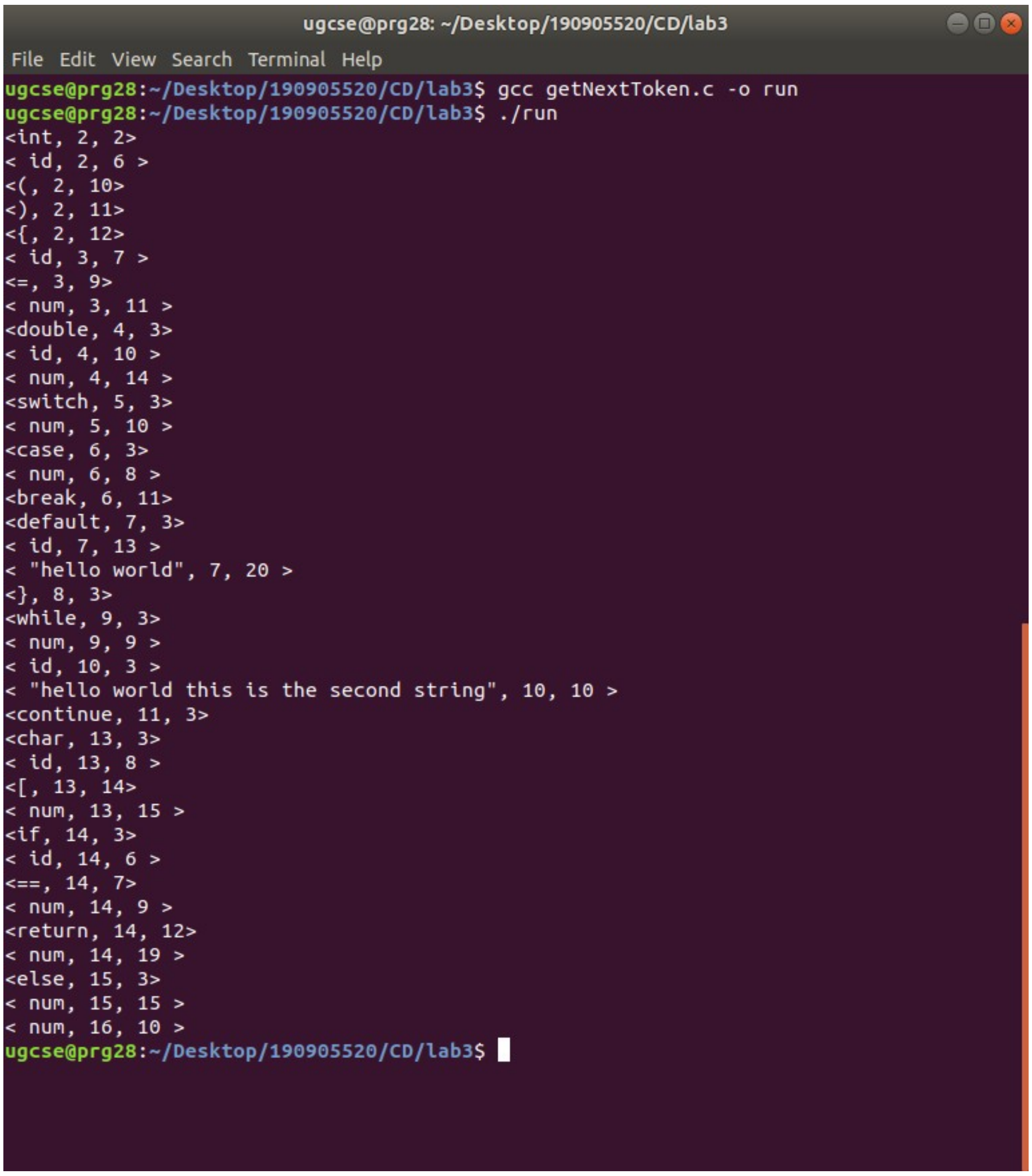
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

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

preprocess_output.c :

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

Output :


ugcse@prg28: ~/Desktop/190905520/CD/lab3
File Edit View Search Terminal Help

```
ugcse@prg28:~/Desktop/190905520/CD/lab3$ gcc getNextToken.c -o run
ugcse@prg28:~/Desktop/190905520/CD/lab3$ ./run
<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, 6, 11>
<default, 7, 3>
< id, 7, 13 >
< "hello world", 7, 20 >
<}, 8, 3>
<while, 9, 3>
< num, 9, 9 >
< id, 10, 3 >
< "hello world this is the second string", 10, 10 >
<continue, 11, 3>
<char, 13, 3>
< id, 13, 8 >
<[, 13, 14>
< num, 13, 15 >
<if, 14, 3>
< id, 14, 6 >
<==, 14, 7>
< num, 14, 9 >
<return, 14, 12>
< num, 14, 19 >
<else, 15, 3>
< num, 15, 15 >
< num, 16, 10 >
ugcse@prg28:~/Desktop/190905520/CD/lab3$
```