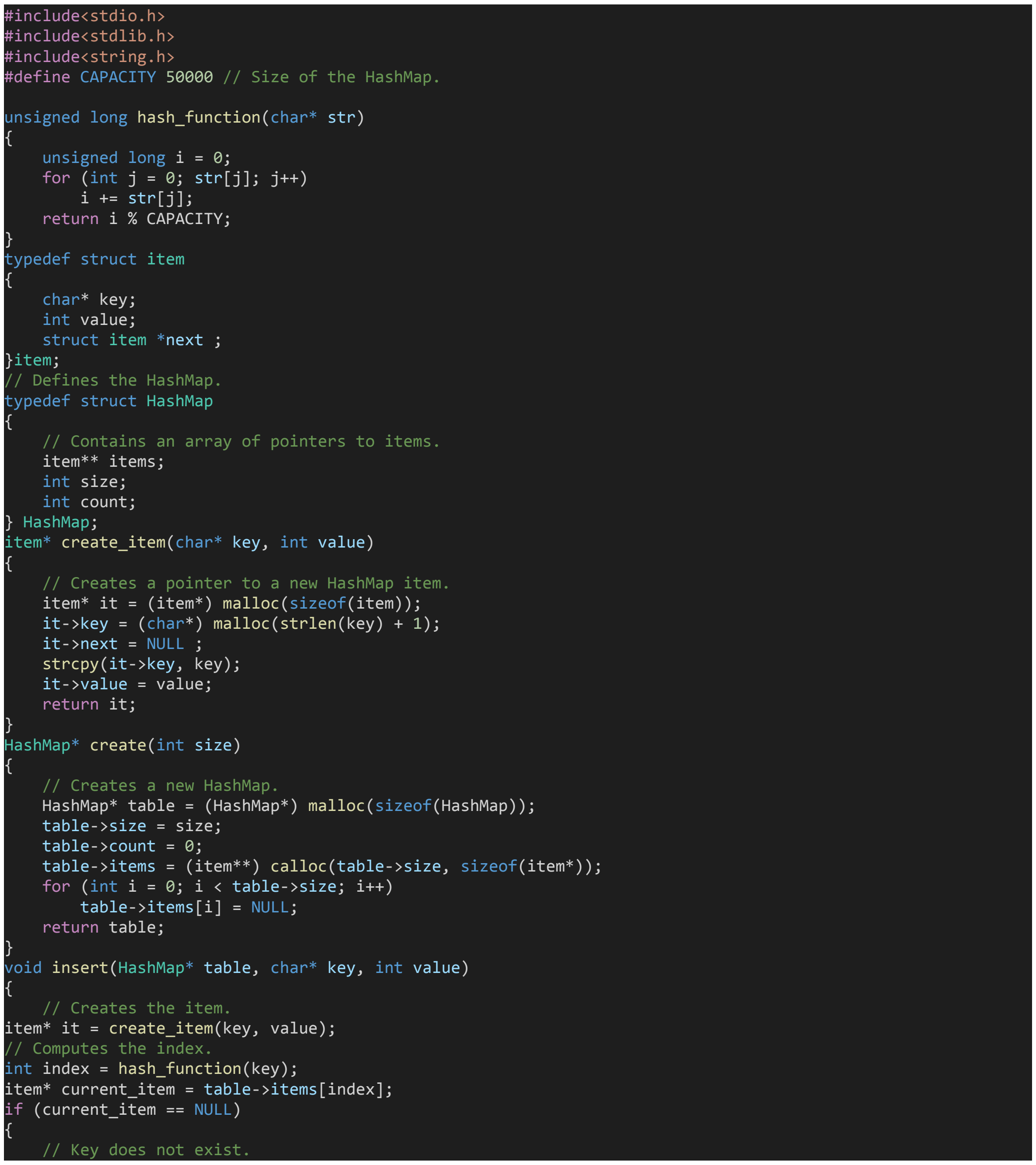
Compiler Design Lab Assignment 2

Name : Santanu Mandal
Roll No : 002010501102

1) **Write a program that will accept a 'C' code as input, perform lexical analysis and output a stream of tokens with lexemes, tokens and the position of each token in the 'C' code.**

Header file :



```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define CAPACITY 50000 // Size of the HashMap.

unsigned long hash_function(char* str)
{
    unsigned long i = 0;
    for (int j = 0; str[j]; j++)
        i += str[j];
    return i % CAPACITY;
}
typedef struct item
{
    char* key;
    int value;
    struct item *next ;
}item;
// Defines the HashMap.
typedef struct HashMap
{
    // Contains an array of pointers to items.
    item** items;
    int size;
    int count;
} HashMap;
item* create_item(char* key, int value)
{
    // Creates a pointer to a new HashMap item.
    item* it = (item*) malloc(sizeof(item));
    it->key = (char*) malloc(strlen(key) + 1);
    it->next = NULL ;
    strcpy(it->key, key);
    it->value = value;
    return it;
}
HashMap* create(int size)
{
    // Creates a new HashMap.
    HashMap* table = (HashMap*) malloc(sizeof(HashMap));
    table->size = size;
    table->count = 0;
    table->items = (item**) calloc(table->size, sizeof(item*));
    for (int i = 0; i < table->size; i++)
        table->items[i] = NULL;
    return table;
}
void insert(HashMap* table, char* key, int value)
{
    // Creates the item.
item* it = create_item(key, value);
// Computes the index.
int index = hash_function(key);
item* current_item = table->items[index];
if (current_item == NULL)
{
    // Key does not exist.
    if (table->count == table->size)
    {
        // HashMap is full.
        //printf("Insert Error: Hash Table is full\n");
        return;
    }
    // Insert directly.
    table->items[index] = it;
    table->count++;
}
else {
    item *cur = current_item ;
    while(cur->next){
        cur = cur->next ;
    }
    cur->next = it ;
}
}
int search(HashMap *h,char *str){
    int idx = hash_function(str) ;
    item* it = h->items[idx];
    while(it){
        if(strcmp(it->key,str)==0) return it->value ;
        it = it->next ;
    }
}
```

Lex file :

```
%{
    #include "data_structure.h"
    #include<string.h>
    int char_no = 1;
    int line_no = 1;

    int isCommentOn = 0;
    int itr = 1;
    int printflag = 0 ;
    HashMap *hp;
    void endcomment(){
        if (!isCommentOn){
            fprintf(yyout, "arith: *\n");
            fprintf(yyout, "arith: /\n");
        } else {
            isCommentOn = 0;
        }
        line_no += 2;
    }
    void space(char * s, int len, char mode){
        if (mode == 'W' || mode == 'C'){
            for (int i = 0; i < len; i++){
                if (s[i] == '\n') {
                    char_no = 1;
                    line_no += 1;
                } else {
                    char_no += 1;
                }
            }
        }
    }
    void job(char * s, int len){
        int val = search(hp, s);
        if (val == 0) {
            fprintf (yyout, "\t\tNew Id encountered\n");
            insert(hp,s,itr);
            itr++;
        } else {
            fprintf (yyout, "\t\tId encountered: %d\n", val);
        }
    }
%}
%%
[ \n\t]+                        { space (yytext, strlen(yytext), 'W'); }
"//"[^\n]*                      { fprintf(yyout, "REMOVING THE SINGLELINE COMMENT"); line_no ++; char_no =
1;}
"/*"[^"*/"]*                    { fprintf(yyout, "REMOVING THE MULTILINE COMMENT\n"); isCommentOn = 1; space
(yytext, strlen(yytext), 'C');}
"*/"                            { endcomment(); }
'.'                           { fprintf(yyout, "Character: %s\t\tpos(%d, %d)\n", yytext, line_no, char_no);
char_no += 3;}
```

```
#include                {fprintf(yyout, "Pre processor directive: %s\t\tpos(%d, %d)\n", yytext, line_no,
char_no); char_no += strlen(yytext);}
[A-Za-z]*.h             {fprintf(yyout, "Header file: %s\t\tpos(%d, %d)\n", yytext, line_no, char_no);
char_no += strlen(yytext);}
\"([A-Za-z0-9]+\s)*\" {fprintf(yyout, "String: %s\t\tpos(%d, %d)\n", yytext, line_no, char_no); char_no +=
strlen(yytext); job(yytext, strlen(yytext)); }
auto|double|int|struct|break|else|long|switch|case|enum|register|typedef|char|extern|return|union|continue|
for|signed|void|do|if|static|while|default|goto|sizeof|volatile|const|float|short|unsigned|printf
{ if(strcmp(yytext,"printf")==0)printflag = 1 ; fprintf(yyout, "keyword encountered: %s\t\tpos(%d, %d)\n",
yytext, line_no, char_no); char_no += strlen(yytext); }
[A-Za-z][A-Za-z0-9]*    { if(!printflag && strcmp(yytext,"main")!=0){fprintf(yyout, "identifier
encountered: %s\t\tpos(%d, %d)\n", yytext, line_no, char_no); char_no += strlen(yytext); job(yytext,
strlen(yytext));} }
">="|"<="|"=="              { fprintf(yyout, "rel operator: %s\t\tpos(%d, %d)\n", yytext, line_no,
char_no); char_no += strlen(yytext);}
">"|"<"                     { fprintf(yyout, "rel operator : %s\t\tpos(%d, %d)\n", yytext, line_no,
char_no); char_no += strlen(yytext); }
"++"|"--"                   { fprintf(yyout, "increment/decrement operator: %s\t\tpos(%d, %d)\n", yytext,
line_no, char_no); char_no += strlen(yytext); }
"+="|"-="|"*="|"/="         { fprintf(yyout, "arith assignment operator: %s\t\tpos(%d, %d)\n", yytext,
line_no, char_no); char_no += strlen(yytext); }
"&="|"|="|"<<="|">>="       { fprintf(yyout, "bitwise assignment operator: %s\t\tpos(%d, %d)\n", yytext,
line_no, char_no); char_no += strlen(yytext); }
"&&="|"||="                 { fprintf(yyout, "and/or assignment: %s\t\tpos(%d, %d)\n", yytext, line_no,
char_no); char_no += strlen(yytext); }
"="                         { fprintf(yyout, "assignment: %s\t\tpos(%d, %d)\n", yytext, line_no, char_no);
char_no += strlen(yytext); }
"+"|"-"|"*"|"/"             { fprintf(yyout, "arithmetic operator: %s\t\tpos(%d, %d)\n", yytext, line_no,
char_no); char_no += strlen(yytext); }
"&"|"|"|"<<"|">>"           { fprintf(yyout, "bitwise operator: %s\t\tpos(%d, %d)\n", yytext, line_no,
char_no); char_no += strlen(yytext); }
"&&"|"||"                   { fprintf(yyout, "logical operator: %s\t\tpos(%d, %d)\n", yytext, line_no,
char_no); char_no += strlen(yytext); }
[;,(){}]                    { if(strcmp(yytext,";")==0 && printflag) printflag = 0 ; fprintf(yyout,
"special character: %s\t\tpos(%d, %d)\n", yytext, line_no, char_no); char_no +=strlen(yytext);  }
[0-9]+                                                  { fprintf(yyout, "Integer: %s\t\tpos(%d,
%d)\n", yytext, line_no, char_no); char_no += strlen(yytext);}
[0-9]+\.[0-9]+                                          { fprintf(yyout, "Float: %s\t\tpos(%d, %d)\n",
yytext, line_no, char_no); char_no += strlen(yytext);}
[0-9]+E[?\-0-9][0-9]*                                   { fprintf(yyout, "Exponential: %s\t\tpos(%d,
%d)\n", yytext, line_no, char_no); char_no += strlen(yytext);}
[?\-0-9][0-9]*?.[0-9]+E[?\-0-9][0-9]*                   { fprintf(yyout, "Exponential: %s\t\tpos(%d,
%d)\n", yytext, line_no, char_no); char_no += strlen(yytext);}
[0-9][A-Za-z0-9]*[A-Za-z][A-Za-z0-9]*                   { if(printflag==0) fprintf(yyout, "Error
Encountered: %s\t\tpos(%d, %d)\n", yytext, line_no, char_no); char_no += strlen(yytext);}
.                                                       { if(printflag==0) fprintf(yyout, "INVALID text
encountered: %s\t\tpos(%d, %d)\n", yytext, line_no, char_no); char_no += strlen(yytext); }
%%
int yywrap(){}
int main() {
    hp = create(50000);
    extern FILE *yyin, *yyout;
    yyin = fopen("test.c", "r");
    yyout = fopen("Output.txt", "w");
    yylex();
```

```
    if (isCommentOn){
        fprintf(yyout, "Error: Comment not ended\n");
    }
    fprintf(yyout, "\n\nSymbol Table\n\nid\t\tvalue\n------------\n");
    for (int i = 0; i < 50000; i++){
        item * temp = hp->items[i];
        while (temp){
            fprintf(yyout, "id%d\t\t%s\n", temp->value, temp->key);
            temp = temp->next;
        }
    }
    return 0;
}
```

Input C File :-

```
#include<stdio.h>

int main(){
    int a = 0 , b = 0 , c = 5 ;
    printf("Hello World !!!") ;
    return 0 ;
}
```

Output file with lexems :-

```
Pre processor directive: #include       pos(1, 1)
rel operator 1digit: <      pos(1, 9)
Header file: stdio.h        pos(1, 10)
rel operator 1digit: >      pos(1, 17)
keyword encountered: int        pos(3, 1)
special character: (        pos(3, 5)
special character: )        pos(3, 6)
special character: {        pos(3, 7)
keyword encountered: int        pos(4, 5)
identifier encountered: a       pos(4, 9)
        New Id encountered
assignment: =       pos(4, 11)
Integer: 0      pos(4, 13)
special character: ,        pos(4, 15)
identifier encountered: b       pos(4, 17)
        New Id encountered
assignment: =       pos(4, 19)
Integer: 0      pos(4, 21)
special character: ,        pos(4, 23)
identifier encountered: c       pos(4, 25)
        New Id encountered
assignment: =       pos(4, 27)
Integer: 5      pos(4, 29)
special character: ;        pos(4, 31)
keyword encountered: printf     pos(5, 5)
special character: (        pos(5, 11)
special character: )        pos(5, 19)
special character: ;        pos(5, 21)
keyword encountered: return     pos(6, 5)
Integer: 0      pos(6, 12)
special character: ;        pos(6, 14)
special character: }        pos(7, 1)


Symbol Table
id      value
------------
id1     a
id2     b
id3     c
```

a) Output any lexical error in the 'C' code.

Input c file

```
#include<stdio.h>

int main(){
    int 6b = 0 ;
    return 0 ;
}
```

Output file with lexems :-

```
Pre processor directive: #include       pos(1, 1)
rel operator 1digit: <      pos(1, 9)
Header file: stdio.h        pos(1, 10)
rel operator 1digit: >      pos(1, 17)
keyword encountered: int        pos(3, 1)
special character: (        pos(3, 5)
special character: )        pos(3, 6)
special character: {        pos(3, 7)
keyword encountered: int        pos(4, 5)
Error Encountered: 6b       pos(4, 9)                # Error is encountered in this line
assignment: =       pos(4, 13)
Integer: 0      pos(4, 15)
special character: ;        pos(4, 17)
keyword encountered: return     pos(5, 5)
Integer: 0      pos(5, 12)
special character: ;        pos(5, 14)
special character: }        pos(6, 1)
```

```
Symbol Table
id      value
------------
```

b) A token may appear several times in the code. For example, an identifier 'sum' may appear in the declaration statement, in multiple assignment statements like "sum = a+b;" or "result = sum + z;" or in conditional statements.The tokens need to be stored in a symbol table. You need to store the token using an efficient data structure, such that insertion and searching from the data structure become efficient.Which data structure have you used in your code? What are the time complexities of insertion and search for a token?

Input c file

```
#include<stdio.h>

int main(){
    int s ,a = 5  ;
    s = a ;
    return 0 ;
}
```

Output text file

```
Pre processor directive: #include       pos(1, 1)
rel operator : <     pos(1, 9)
Header file: stdio.h        pos(1, 10)
rel operator : >     pos(1, 17)
keyword encountered: int        pos(3, 1)
special character: (        pos(3, 5)
special character: )        pos(3, 6)
special character: {        pos(3, 7)
keyword encountered: int        pos(4, 5)
identifier encountered: sum     pos(4, 9)
        New Id encountered
special character: ,        pos(4, 13)
identifier encountered: a       pos(4, 14)
        New Id encountered
assignment: =       pos(4, 16)
Integer: 5      pos(4, 18)
special character: ,        pos(4, 20)
identifier encountered: b       pos(4, 22)
        New Id encountered
assignment: =       pos(4, 24)
Integer: 10     pos(4, 26)
special character: ;        pos(4, 29)
identifier encountered: s     pos(5, 5)
        Id encountered: 1
assignment: =       pos(5, 9)
identifier encountered: a       pos(5, 11)
        Id encountered: 2
arithmetic operator: +      pos(5, 12)
special character: ;        pos(5, 15)
keyword encountered: return     pos(6, 5)
Integer: 0      pos(6, 12)
special character: ;        pos(6, 14)
special character: }        pos(7, 1)
```

Symbol Table

```
Symbol Table

id      value
------------
id2     a
id1     s
```