Roll No:

Lab Performance Test [No]
Lab Task Q[No]

Q1. Consider the following code snippet:

def X as INT;

FOR_LOOP (Y from 10 to 20.0);

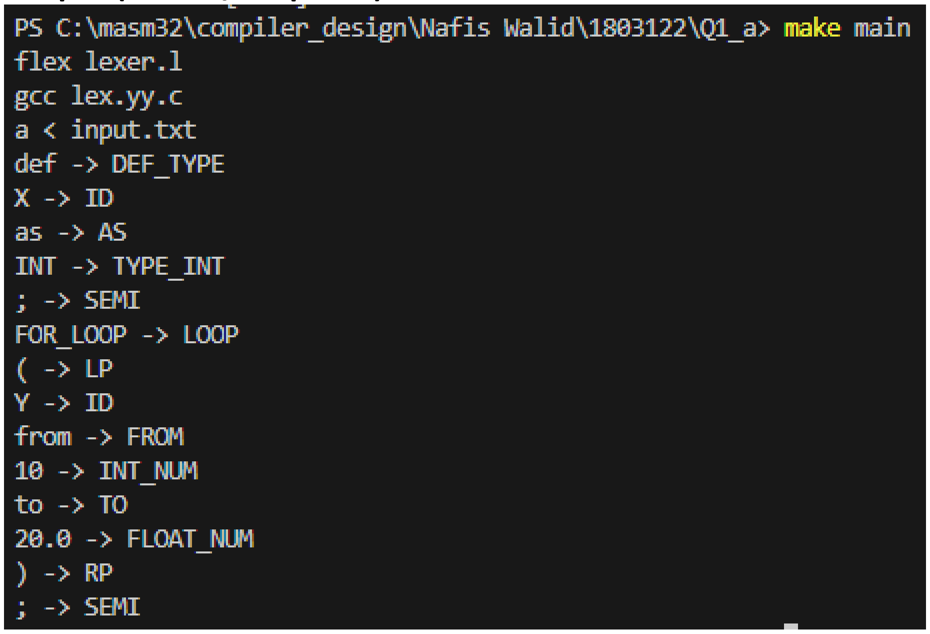
a) Perform Lexical Analysis on the given code snippet.

Solution (Bold your own written code):

```
%option noyywrap

%{

%}
delim [ \t\n]
ws {delim}+
digit     [0-9]

ICONST    [0-9]{digit}*
FCONST    {digit}*"."{digit}+

%%
{ws}        { }
{ICONST}    {printf("%s -> INT_NUM\n", yytext);}
{FCONST}    {printf("%s -> FLOAT_NUM\n", yytext);}
"def"       { printf("%s -> DEF_TYPE\n", yytext); }
"X"     { printf("%s -> ID\n", yytext); }
"as"        { printf("%s -> AS\n", yytext); }
"INT"       { printf("%s -> TYPE_INT\n", yytext); }
"FOR_LOOP"      { printf("%s -> LOOP\n", yytext); }
"("     { printf("%s -> LP\n", yytext); }
")"     { printf("%s -> RP\n", yytext); }
"Y"     { printf("%s -> ID\n", yytext); }
"from"      { printf("%s -> FROM\n", yytext); }
"to"        { printf("%s -> TO\n", yytext); }
";"       { printf("%s -> SEMI\n", yytext); }
%%

int main()
```

```
{
    yylex();
    return 0;
}
```

Output (Screen/SnapShot):



```
PS C:\masm32\compiler_design\Nafis Walid\1803122\Q1_a> make main
flex lexer.l
gcc lex.yy.c
a < input.txt
def -> DEF_TYPE
X -> ID
as -> AS
INT -> TYPE_INT
; -> SEMI
FOR_LOOP -> LOOP
( -> LP
Y -> ID
from -> FROM
10 -> INT_NUM
to -> TO
20.0 -> FLOAT_NUM
) -> RP
; -> SEMI
```

Question: Q2_b
Perform Syntax Analysis on the given code snippet.

def X as INT;

FOR_LOOP (Y from 10 to 20.0);

Solution (Bold your own written code): lexer.l

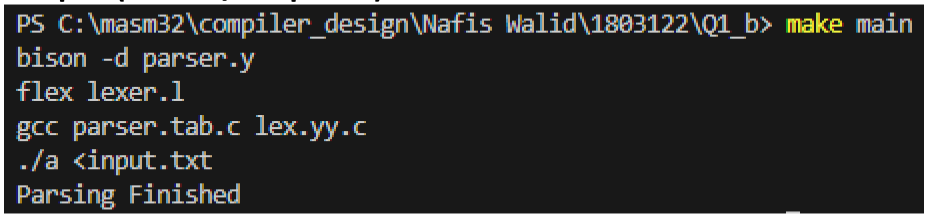
```
%option noyywrap

%{
    #include "parser.tab.h"
%}
delim [ \t\n]
ws {delim}+
digit     [0-9]

ICONST    [0-9]{digit}*
FCONST    {digit}*"."{digit}+

%%
{ws}        { }
{ICONST}    { return INT_NUM;}
{FCONST}    { return FLOAT_NUM; }
"def"       { return DEF; }
"X"     { return ID; }
"as"        { return AS; }
"INT"       { return INT_TYPE; }
"FOR_LOOP"      { return LOOP; }
"("     { return LP; }
")"     { return RP; }
"Y"     { return ID; }
"from"      { return FROM; }
"to"        { return TO; }
";"       { return SEMI; }
%%
```

Solution (Bold your own written code): parser.y

```
%{
#include<stdio.h>
void yyerror(char *s);
int yylex();
%}

%token INT_TYPE DEF AS LOOP FROM TO
%token LP RP SEMI
%token ID INT_NUM FLOAT_NUM

%start stmts
```

```
%%
stmts: stmts stmt
    | stmt
    ;

stmt: loop
    | exp
    ;

loop: LOOP LP ID FROM INT_NUM TO FLOAT_NUM RP SEMI
    ;

exp: DEF ID AS INT_TYPE SEMI
    ;

%%

void yyerror(char *s)
{
    fprintf(stderr, "error: %s", s);
}
int main()
{
    yyparse();
    printf("Parsing Finished\n");
}
```

Output (Screen/SnapShot):



```
PS C:\masm32\compiler_design\Nafis Walid\1803122\Q1_b> make main
bison -d parser.y
flex lexer.l
gcc parser.tab.c lex.yy.c
./a <input.txt
Parsing Finished
```

Q1_c) Perform Semantic Analysis on the given code snippet.

def X as INT;

FOR_LOOP (Y from 10 to 20.0);

Solution (Bold your own written code): symbtab.c

```c
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include "symtab.h"

list_t* head = NULL;

void insert(char* name, int type)
{
    if(search(name)==NULL)
    {
        list_t *temp = (list_t*)malloc(sizeof(list_t));

        strcpy(temp->st_name, name);
        temp->st_type = type;

        printf("inserting %s with type %d\n", temp->st_name, temp-
>st_type);

        temp->next = head;
        head = temp;
    }
    else
    {
        printf("same variable %s is declared more than one\n",
name);
        yyerror();
    }

}

list_t* search(char *name)
{
    list_t *current = head;

    while (current!=NULL)
    {
        if(strcmp(name, current->st_name)!=0)
            current = current->next;
        else
            break;
    }
    return current;
}

int id_check(char *st_name)
```

```
{
    list_t *id = search(st_name);

    if (id==NULL)
        return -1;

    return 1;
}

int get_type(char *st_name)
{
    list_t* id = search(st_name);
    return id->st_type;
}

int type_check(int type1, int type2)
{
    if (type1==INT_TYPE && type2==INT_TYPE)
    {
        return (INT_TYPE);
    }
    else if (type1==INT_TYPE && type2==REAL_TYPE)
    {
        return (REAL_TYPE);
    }
    else if (type1==INT_TYPE && type2==CHAR_TYPE)
    {
        printf("Type INT and Type CHAR are incompatiable\n");
        return (-1);
    }
    else if (type1==CHAR_TYPE && type2==REAL_TYPE)
    {
        printf("Type REAL and Type CHAR are incompatiable\n");
        return (-1);
    }
    else
    {
        printf("Types are incompatiable\n");
        return (-1);
    }

}
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include "symtab.h"
```

```
list_t* head = NULL;

void insert(char* name, int type)
{
    if(search(name)==NULL)
    {
        list_t *temp = (list_t*)malloc(sizeof(list_t));

        strcpy(temp->st_name, name);
        temp->st_type = type;

        printf("inserting %s with type %d\n", temp->st_name, temp-
>st_type);

        temp->next = head;
        head = temp;
    }
    else
    {
        printf("same variable %s is declared more than one\n",
name);
       yyerror();
    }

}

list_t* search(char *name)
{
    list_t *current = head;

    while (current!=NULL)
    {
        if(strcmp(name, current->st_name)!=0)
            current = current->next;
        else
            break;
    }
    return current;
}

int id_check(char *st_name)
{
    list_t *id = search(st_name);

    if (id==NULL)
        return -1;
    return 1;
}

int get_type(char *st_name)
{
    list_t* id = search(st_name);
    return id->st_type;
}

int type_check(int type1, int type2)
{
    if (type1==INT_TYPE && type2==INT_TYPE)
    {
        return (INT_TYPE);
    }
    else if (type1==INT_TYPE && type2==REAL_TYPE)
    {
        return (REAL_TYPE);
    }
    else if (type1==INT_TYPE && type2==CHAR_TYPE)
    {
        printf("Type INT and Type CHAR are incompatiable\n");
        return (-1);
    }
    else if (type1==CHAR_TYPE && type2==REAL_TYPE)
    {
        printf("Type REAL and Type CHAR are incompatiable\n");
      return (-1);
    }
    else
    {
        printf("Types are incompatiable\n");
        return (-1);
    }

}
```

Solution (Bold your own written code):symbtab.h

```
#define INT_TYPE 1
#define REAL_TYPE 2
#define CHAR_TYPE 3

typedef struct list_t
{
    char st_name[40];
    int st_type;
    struct list_t *next;
}list_t;

list_t* search(char *name);
void insert(char* name, int type);
int id_check(char *st_name);
```

Output (Screen/SnapShot):