**Roll No: 1803122**

**Lab Performance Test [No]**

**Lab Task Q[No]**

**Q1.**  Consider the following code snippet:

**def X as INT;**

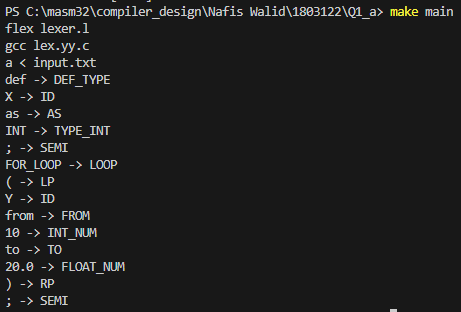
**FOR\_LOOP ( Y from 10 to 20.0);**

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):**

****

**Question: Q2**\_b

Perform Syntax Analysis on the given code snippet.

**def X as INT;**

**FOR\_LOOP ( Y from 10 to 20.0);**

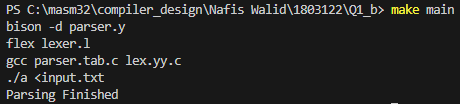
**Solution (Bold your own written code): lexer.l**

|  |
| --- |
| %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):**

****

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**

|  |
| --- |
| #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):**