**COMPILER DESIGN WEEK 8**

Q1. Design the recursive descent parser to parse array declarations and expression

statements with error reporting. Subset of grammar 7.1 is as follows:

PROGRAM:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include "lexanls.h"

void Program();

void declarations();

void data\_type();

void identifier\_list();

void statement\_list();

int statement();

void assign\_stat();

void expn();

void eprime();

void simple\_exp();

void seprime();

void term();

void tprime();

void factor();

void relop();

void addop();

void mulop();

struct token curr;

FILE \*f1;

void invalid()

{

printf("error");

exit(0);

}

//PROGRAM FUNCTION

void Program()

{

curr=getNextToken(f1);

if(isdtype(curr.lexeme))

{

curr=getNextToken(f1);

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();

statement\_list();

if(strcmp(curr.lexeme,"}")==0)

{

return;

}

else

{

printf("\n missing } at row:%d and col:%d.\n",curr.row,curr.col);

exit(0);

}

}

else

{

printf("\n missing { at row:%d and col:%d.\n",curr.row,curr.col);

exit(0);

}

}

else

{

printf("\n missing ) at row:%d and col:%d.\n",curr.row,curr.col);

exit(0);

}

}

else

{

printf("\n missing ( at row:%d and col:%d.\n",curr.row,curr.col);

exit(0);

}

}

else

{

printf("\n missing main function\n");

exit(0);

}

}

else

{

printf("return datatype missing at row:%d and col:%d\n",curr.row,curr.col);

exit(0);

}

}

//DECLARATION FUNCTION

void declarations()

{

if(isdtype(curr.lexeme))

{

data\_type();

identifier\_list();

if(strcmp(curr.lexeme,";")==0)

{

curr=getNextToken(f1);

declarations();

}

else printf("\n missing ; at row:%d and col:%d.\n",curr.row,curr.col);

}

}

//DATA TYPE FUNCTION

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("\n missing data type at row:%d and col:%d.\n",curr.row,curr.col);

exit(0);

}

}

//IDENTIFIER LIST FUNCTION

void identifier\_list()

{

if(strcmp(curr.type,"IDENTIFIER")==0)

{

curr=getNextToken(f1);

if(strcmp(curr.lexeme,",")==0)

{

curr=getNextToken(f1);

identifier\_list();

}

else if(strcmp(curr.lexeme,"[")==0)

{

curr=getNextToken(f1);

if(strcmp(curr.type,"NUMBER")==0)

{

curr=getNextToken(f1);

if(strcmp(curr.lexeme,"]")==0)

{

curr=getNextToken(f1);

if(strcmp(curr.lexeme,",")==0)

{

curr=getNextToken(f1);

identifier\_list();

}

else

return;

}

else

{

printf("missing ] in row:%d and col:%d\n",curr.row,curr.col);

exit(0);

}

}

else

{

printf("value not integer in row:%d and col:%d\n",curr.row,curr.col);

exit(0);

}

}

else

return;

}

else

{

printf("\n missing identifier at row:%d and col:%d.\n",curr.row,curr.col);

exit(0);

}

}

//STATEMENT LIST FUNCTION

void statement\_list()

{

if(statement()==1)

statement\_list();

}

//STATEMENT FUNCTION

int statement()

{

assign\_stat();

return 1;

}

//ASSIGN STAT FUNCTION

void assign\_stat()

{

if(strcmp(curr.type,"IDENTIFIER")==0)

{

curr=getNextToken(f1);

if(strcmp(curr.lexeme,"=")==0)

{

curr=getNextToken(f1);

expn();

return;

}

else

{

printf("missing = in row:%d and col:%d\n",curr.row,curr.col);

exit(0);

}

}

else

{

printf("\n missing identifier at row:%d and col:%d.\n",curr.row,curr.col);

exit(0);

}

}

//EXPN FUNCTION

void expn()

{

simple\_exp();

eprime();

}

void eprime()

{

if (strcmp(curr.type,"RELATIONALOPERATOR")==0)

{

relop();

simple\_exp();

}

}

void simple\_exp()

{

term();

seprime();

}

void seprime()

{

if (strcmp(curr.lexeme,"+") == 0 || strcmp(curr.lexeme,"-") == 0)

{

addop();

term();

seprime();

}

}

void term()

{

factor();

tprime();

}

void tprime()

{

if (strcmp(curr.lexeme,"\*") == 0 || strcmp(curr.lexeme,"/") == 0 || strcmp(curr.lexeme,"%") == 0)

{

mulop();

factor();

tprime();

}

}

void factor()

{

if (strcmp(curr.type, "IDENTIFIER") == 0)

{

curr = getNextToken(f1);

return;

}

else if (strcmp(curr.type, "NUMBER") == 0)

{

curr = getNextToken(f1);

return;

}

}

void relop()

{

if (strcmp(curr.lexeme, "==") == 0)

{

curr = getNextToken(f1);

return;

}

if (strcmp(curr.lexeme, "!=") == 0)

{

curr = getNextToken(f1);

return;

}

if (strcmp(curr.lexeme, "<=") == 0)

{

curr = getNextToken(f1);

return;

}

if (strcmp(curr.lexeme, ">=") == 0)

{

curr = getNextToken(f1);

return;

}

if (strcmp(curr.lexeme, "<") == 0)

{

curr = getNextToken(f1);

return;

}

if (strcmp(curr.lexeme, ">") == 0)

{

curr = getNextToken(f1);

return;

}

}

void addop()

{

if (strcmp(curr.lexeme, "+") == 0)

{

curr= getNextToken(f1);

return;

}

if (strcmp(curr.lexeme, "-") == 0)

{

curr = getNextToken(f1);

return;

}

}

void mulop()

{

if (strcmp(curr.lexeme, "\*") == 0)

{

curr = getNextToken(f1);

return;

}

if (strcmp(curr.lexeme, "/") == 0)

{

curr = getNextToken(f1);

return;

}

if (strcmp(curr.lexeme, "\*") == 0)

{

curr = getNextToken(f1);

return;

}

}

int main()

{

FILE \*fa, \*fb;

int ca, cb;

fa = fopen("inp\_71.c", "r");

if (fa == NULL)

{

printf("Invalid file\n");

return 0;

}

fb = fopen("out\_71.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("out\_71.c", "r");

if(fa == NULL)

{

printf("Invalid file");

return 0;

}

fb = fopen("temp.c", "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("temp.c", "r");

fb = fopen("out\_71.c", "w");

ca = getc(fa);

while(ca != EOF)

{

putc(ca, fb);

ca = getc(fa);

}

fclose(fa);

fclose(fb);

remove("temp.c");

f1=fopen("out\_71.c","r");

if(f1==NULL)

{

printf("Invalid file\n");

return 0;

}

struct token tkn;

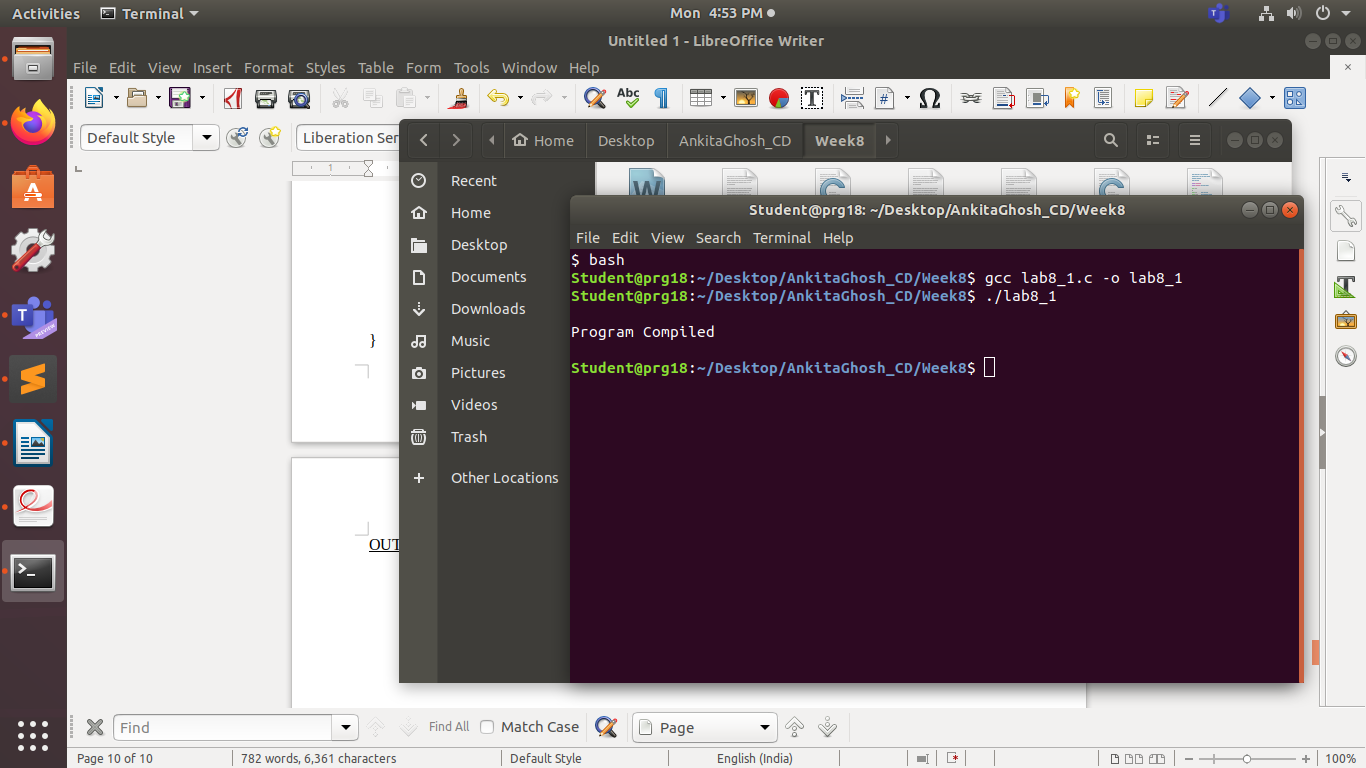
Program();

printf("\nProgram Compiled\n\n");

fclose(f1);

}

OUTPUT:



Input Program:

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

int main()  
{  
int arr[20],a,b,sum  
a=5;  
b=1;  
sum=a+b;  
}