Roll No: 1803067

Lab final

Lab Task Q1a

Question:

Q1. Consider following Version1 of Code Snippet:

SUB main ()

suru

INT --> p

INT --> q

p soman 8

q soman p jog 12 - 1

print <-- q

sesh

a. Design Lexical Analysis and Syntax Analysis part of compiler based on the version1 of code snippet.

b. Design Intermediate Code Generation and Code Generation part of compiler based on the version1 of code snippet.

Solution (Bold your own written code):

.l file

```
%option noyywrap

%{
    #define UNDEF_TYPE 0
    #define INT_TYPE 1
    #define REAL_TYPE 2
    #define CHAR_TYPE 3
    #define SINGLE_TYPE 4
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
```

```
    #include "parser.tab.h"

    int lineno = 1; // initialize to 1
    void yyerror();
%}

alpha     [a-zA-Z]
digit     [0-9]
alnum     {alpha}|{digit}
print     [ -~]

ID        {alpha}{alnum}*
ICONST    [0-9]{digit}*
%%

"//".*        { }

"INT"       {yylval.int_val=INT_TYPE; return INT; }
"SUB"           { return RETTYPE; }
"suru"          { return SURU ;}
"sesh"          { return SESH ;}
"-->"           {  return CIN ;}
"<--"           { return COUT; }


"jog"       { return ADDOP; }
"-"       { return SUBOP; }
"*"       { return MULOP; }
"/"       { return DIVOP; }
"=="      { return EQUOP; }
">"       { return GT; }
"<"       { return LT; }

"("       { return LPAREN; }
```

```
")"        { return RPAREN; }
"{"        { return LBRACE; }
"}"        { return RBRACE; }
";"        { return SEMI; }
"soman"       { return ASSIGN; }
"print"       { return PRINT; }
"scan"       { return SCAN; }

{ID}        {strcpy(yylval.str_val, yytext); return ID;}
{ICONST}    {yylval.int_val=atoi(yytext); return ICONST;}

"\n"        { lineno += 1; }
[ \t\r\f]+

.       { yyerror("Unrecognized character"); }
```

.y file

```
%{
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include "symtab.c"
    #include "codeGen.c"
    void yyerror();
    extern int lineno;
    extern int yylex();
%}

%union
{
    char str_val[100];
    int int_val;
}
```

```
%token PRINT SCAN
%token ADDOP SUBOP MULOP DIVOP EQUOP LT GT
%token LPAREN RPAREN LBRACE RBRACE SEMI ASSIGN
%token<str_val> ID
%token ICONST
%token  INT
%type <int_val> INT  ICONST
%token  RETTYPE SURU  SESH  CIN  COUT

%left LT GT /*LT GT has lowest precedence*/
%left ADDOP
%left MULOP /*MULOP has lowest precedence*/


%start program

%%
program:RETTYPE ID LPAREN RPAREN SURU code SESH ;
code:code stmnt | ;
stmnt : dec | assign | printfun ;
dec : INT CIN ID ;
assign : ID ASSIGN exp ;
exp : exp ADDOP exp
    | exp SUBOP exp
    | T ;
T : ID | ICONST ;
printfun : PRINT COUT ID ;
%%

void yyerror ()
{
    printf("Syntax error at line %d\n", lineno);
    exit(1);
```

```
}

int main (int argc, char *argv[])
{
    yyparse();
    printf("Parsing finished!\n");

    //printf("============= INTERMEDIATE CODE===============\n");
    print_code();

   // printf("============= ASM CODE===============\n");
    print_assembly();

    return 0;
}
```

Output (Screen/SnapShot):

```
C:\Users\USER\Desktop\labfinal\LF_1803067_Q1a>make
bison -d parser.y
parser.y: conflicts: 3 shift/reduce
flex lexer.l
gcc parser.tab.c lex.yy.c
a <input.txt
Parsing finished!
```

Lab Task Q1b

Question:

Solution (Bold your own written code):

.y file

```
%{
    #include <stdio.h>
    #include <stdlib.h>
```

```
    #include <string.h>
    #include "symtab.c"
    #include "codeGen.c"
    void yyerror();
    extern int lineno;
    extern int yylex();

%}

%union
{
    char str_val[100];
    int int_val;
}

%token PRINT SCAN
%token ADDOP SUBOP MULOP DIVOP EQUOP LT GT
%token LPAREN RPAREN LBRACE RBRACE SEMI ASSIGN
%token<str_val> ID
%token ICONST
%token  INT
%type <int_val> INT  ICONST
%token  RETTYPE SURU  SESH  CIN  COUT

%left LT GT /*LT GT has lowest precedence*/
%left ADDOP
%left MULOP /*MULOP has lowest precedence*/


%start program

%%
program:RETTYPE ID LPAREN RPAREN SURU {gen_code(START, -1);}
code  {gen_code(HALT, -1);} SESH ;
```

```
code:code stmnt | ;
stmnt : dec | assign | printfun ;
dec : INT CIN ID {
    insert($3,$1);
} ;
assign : ID ASSIGN exp
    {
        int address = idcheck($1);
        if(address!=-1)
        {
            gen_code(STORE,address);
        }
        else{
             yyerror ();
        }

    };
exp : exp ADDOP exp { gen_code(ADD,-1);}
    | exp SUBOP exp { gen_code(SUB,-1);}
    | T ;
T : ID
    {
        int address = idcheck($1);
        if(address!=-1)
        {
            gen_code(LD_VAR,address);
        }
        else{
             yyerror ();
        }

    }

    | ICONST
```

```
    { gen_code(LD_INT,$1);} ;
printfun : PRINT COUT ID
    {
        int address = idcheck($3);
        if(address!=-1)
        {
            gen_code(PRINT_INT_VALUE,address);
        }
        else{
             yyerror ();
        }

    };
%%

void yyerror ()
{
    printf("Syntax error at line %d\n", lineno);
    exit(1);
}

int main (int argc, char *argv[])
{
    yyparse();
    printf("Parsing finished!\n");

    printf("============= INTERMEDIATE CODE===============\n");
    print_code();

    printf("============= ASM CODE===============\n");
    print_assembly();

    return 0;
}
```

gencode.c file

```
#include "codeGen.h"

int gen_label()
{
    return code_offset;
}

void gen_code(enum code_ops op, int arg)
{
    code[code_offset].op = op;
    code[code_offset].arg = arg;

    code_offset++;
}

void print_code()
{
    int i = 0;

    for(i=0; i<code_offset; i++)
    {
        printf("%3d: %-15s  %4d\n", i, op_name[code[i].op],
code[i].arg);
    }
}

void print_assembly()
{
    int i = 0;
    int j = 0;
```

```
    int stack_variable_counter = 0;

    for(i=0; i<code_offset; i++)
    {
        printf("\n;%s %d\n", op_name[code[i].op], code[i].arg);

        if(code[i].op == LD_INT || code[i].op == LD_VAR)
            stack_variable_counter++;

        if(code[i].op == ADD)
            stack_variable_counter--;

        switch(code[i].op)
        {
            case START:
                            printf(".686\n");
                            printf(".model flat, c\n");
                            printf("include
C:\\masm32\\include\\msvcrt.inc\n");
                            printf("includelib
C:\\masm32\\lib\\msvcrt.lib\n");
                            printf("\n");
                            printf(".stack 100h\n");
                            printf("printf PROTO arg1:Ptr Byte,
printlist:VARARG\n");
                            printf("scanf PROTO arg2:Ptr Byte,
inputlist:VARARG\n");
                            printf("\n");
                            printf(".data\n");
                            printf("output_integer_msg_format byte \"\%
%d\", 0Ah, 0\n");
                            printf("output_string_msg_format byte \"\%
%s\", 0Ah, 0\n");
```

```
                        printf("input_integer_format byte \"\%
%d\",0\n");
                        printf("\n");
                        printf("number sdword ?\n");
                        printf("\n");
                        printf(".code\n");
                        printf("\n");
                        printf("main proc\n");
                        printf("\tpush ebp\n");
                        printf("\tmov ebp, esp\n");
                        printf("\tsub ebp, 100\n");
                        printf("\tmov ebx, ebp\n");
                        printf("\tadd ebx, 4\n");
                        break;

            case HALT:
                        printf("\tadd ebp, 100\n");
                        printf("\tmov esp, ebp\n");
                        printf("\tpop ebp\n");
                        printf("\tret\n");
                        printf("main endp\n");
                        printf("end\n");
                        break;

            case STORE:
                        printf("\tmov eax, [ebx-4]\n");
                        printf("\tmov dword ptr [ebp-%d], eax\n",
4*code[i].arg);
                        break;

            case SCAN_INT_VALUE:
                        printf("\tpush eax\n");
                        printf("\tpush ebx\n");
                        printf("\tpush ecx\n");
```

```
                            printf("\tpush edx\n");
                            for(j=address-1; j>=0; j--)
                                printf("\tpush [ebp-%d]\n", 4*j);
                            for(j=1; j<=stack_variable_counter; j++)
                                printf("\tpush [ebp+%d]\n", 4*j);
                            printf("\tpush ebp\n");

                            printf("\tINVOKE scanf, ADDR
input_integer_format, ADDR number\n");

                            printf("\tpop ebp\n");
                            for(j=stack_variable_counter; j>=1; j--)
                                printf("\tpop [ebp+%d]\n", 4*j);
                            for(j=0; j<=address-1; j++)
                                printf("\tpop [ebp-%d]\n", 4*j);

                            printf("\tmov eax, number\n");
                            printf("\tmov dword ptr [ebp-%d], eax\n",
4*code[i].arg);

                            printf("\tpop edx\n");
                            printf("\tpop ecx\n");
                            printf("\tpop ebx\n");
                            printf("\tpop eax\n");
                            break;


            case PRINT_INT_VALUE:
                            printf("\tpush eax\n");
                            printf("\tpush ebx\n");
                            printf("\tpush ecx\n");
                            printf("\tpush edx\n");
                            for(j=address-1; j>=0; j--)
                                printf("\tpush [ebp-%d]\n", 4*j);
```

```
                            for(j=1; j<=stack_variable_counter; j++)
                                printf("\tpush [ebp+%d]\n", 4*j);
                            printf("\tpush ebp\n");

                            printf("\tmov eax, [ebp-%d]\n",
4*code[i].arg);
                            printf("\tINVOKE printf, ADDR
output_integer_msg_format, eax\n");

                            printf("\tpop ebp\n");
                            for(j=stack_variable_counter; j>=1; j--)
                                printf("\tpop [ebp+%d]\n", 4*j);
                            for(j=0; j<=address-1; j++)
                                printf("\tpop [ebp-%d]\n", 4*j);
                            printf("\tpop edx\n");
                            printf("\tpop ecx\n");
                            printf("\tpop ebx\n");
                            printf("\tpop eax\n");
                            break;

            case LD_VAR:
                            printf("\tmov eax, [ebp-%d]\n",
4*code[i].arg);
                            printf("\tmov dword ptr [ebx], eax\n");
                            printf("\tadd ebx, 4\n");
                            printf("\n");
                            break;

            case LD_INT:
                            printf("\tmov eax, %d\n", code[i].arg);
                            printf("\tmov dword ptr [ebx], eax\n");
                            printf("\tadd ebx, 4\n");
                            printf("\n");
                            break;
```

```
        case ADD:
                        printf("\tsub ebx, 4\n");
                        printf("\tmov eax, [ebx]\n");
                        printf("\tsub ebx, 4\n");
                        printf("\tmov edx, [ebx]\n");
                        printf("\tadd eax, edx\n");
                        printf("\tmov dword ptr [ebx], eax\n");
                        printf("\tadd ebx, 4\n");
                        printf("\n");
                        break;
        case SUB:
                        printf("\tsub ebx, 4\n");
                        printf("\tmov eax, [ebx]\n");
                        printf("\tsub ebx, 4\n");
                        printf("\tmov edx, [ebx]\n");
                        printf("\tsub edx, eax\n");
                        printf("\tmov eax, edx\n");
                        printf("\tmov dword ptr [ebx], eax\n");
                        printf("\tadd ebx, 4\n");
                        printf("\n");
                        break;
        case MUL:
                        printf("\tsub ebx, 4\n");
                        printf("\tmov eax, [ebx]\n");
                        printf("\tsub ebx, 4\n");
                        printf("\tmov edx, [ebx]\n");
                        printf("\tIMUL eax, edx\n");
                        printf("\tmov dword ptr [ebx], eax\n");
                        printf("\tadd ebx, 4\n");
                        printf("\n");
                        break;
        case GT_OP:
                        printf("\tsub ebx, 4\n");
```

```
                        printf("\tmov eax, [ebx]\n");
                        printf("\tsub ebx, 4\n");
                        printf("\tmov edx, [ebx]\n");
                        printf("\tcmp edx, eax\n");

                        {
                            char relop_start_label[50]="LS";
                            char relop_end_label[50]="LE";
                            char number[10];
                            itoa(code[i].arg, number, 10);
                            strcat(relop_end_label, number);
                            strcat(relop_start_label, number);

                            printf("\tjg %s\n", relop_start_label);
                            printf("\tmov dword ptr [ebx], 0\n");
                            printf("\tjmp %s\n", relop_end_label);
                            printf("\t%s: mov dword ptr [ebx], 1\n",
relop_start_label);
                            printf("\t%s: add ebx, 4\n\n",
relop_end_label);
                        }
                        printf("\n");
                        break;
            case LT_OP:
                        printf("\tsub ebx, 4\n");
                        printf("\tmov eax, [ebx]\n");
                        printf("\tsub ebx, 4\n");
                        printf("\tmov edx, [ebx]\n");
                        printf("\tcmp edx, eax\n");

                        {
                            char relop_start_label[50]="LS";
                            char relop_end_label[50]="LE";
                            char number[10];
```

```
                                itoa(code[i].arg, number, 10);
                                strcat(relop_end_label, number);
                                strcat(relop_start_label, number);

                                printf("\tjl %s\n", relop_start_label);
                                printf("\tmov dword ptr [ebx], 0\n");
                                printf("\tjmp %s\n", relop_end_label);
                                printf("\t%s: mov dword ptr [ebx], 1\n",
relop_start_label);
                                printf("\t%s: add ebx, 4\n\n",
relop_end_label);
                            }
                            printf("\n");
                            break;
                case LTE_OP:
                            printf("\tsub ebx, 4\n");
                            printf("\tmov eax, [ebx]\n");
                            printf("\tsub ebx, 4\n");
                            printf("\tmov edx, [ebx]\n");
                            printf("\tcmp edx, eax\n");

                            {
                                char relop_start_label[50]="LS";
                                char relop_end_label[50]="LE";
                                char number[10];
                                itoa(code[i].arg, number, 10);
                                strcat(relop_end_label, number);
                                strcat(relop_start_label, number);

                                printf("\tjle %s\n", relop_start_label);
                                printf("\tmov dword ptr [ebx], 0\n");
                                printf("\tjmp %s\n", relop_end_label);
                                printf("\t%s: mov dword ptr [ebx], 1\n",
relop_start_label);
```

```
                                printf("\t%s: add ebx, 4\n\n",
relop_end_label);
                            }
                            printf("\n");
                            break;
            case EQL_OP:
                            printf("\tsub ebx, 4\n");
                            printf("\tmov eax, [ebx]\n");
                            printf("\tsub ebx, 4\n");
                            printf("\tmov edx, [ebx]\n");
                            printf("\tcmp edx, eax\n");

                            {
                                char relop_start_label[50]="LS";
                                char relop_end_label[50]="LE";
                                char number[10];
                                itoa(code[i].arg, number, 10);
                                strcat(relop_end_label, number);
                                strcat(relop_start_label, number);

                                printf("\tje %s\n", relop_start_label);
                                printf("\tmov dword ptr [ebx], 0\n");
                                printf("\tjmp %s\n", relop_end_label);
                                printf("\t%s: mov dword ptr [ebx], 1\n",
relop_start_label);
                                printf("\t%s: add ebx, 4\n\n",
relop_end_label);
                            }
                            printf("\n");
                            break;
            case IF_START:
                            printf("\tmov eax, [ebx-4]\n");
                            printf("\tcmp eax, 0\n");
                            {
```

```
                                char
else_start_label[]="ELSE_START_LABEL_";
                                char number[10];
                                strcat(else_start_label,itoa(code[i].arg
, number, 10));
                                printf("\tjle %s\n", else_start_label);
                            }
                            printf("\n");
                            break;
            case ELSE_START:
                            {
                                char
else_start_label[50]="ELSE_START_LABEL_";
                                char
else_end_label[50]="ELSE_END_LABEL_";
                                char number[10];
                                itoa(code[i].arg, number, 10);
                                strcat(else_end_label, number);
                                printf("\tjmp %s\n", else_end_label);
                                strcat(else_start_label, number);
                                printf("%s:\n", else_start_label);
                            }
                            printf("\n");
                            break;
            case ELSE_END:
                            {
                                char
else_end_label[50]="ELSE_END_LABEL_";
                                char number[10];
                                itoa(code[i].arg, number, 10);
                                strcat(else_end_label, number);
                                printf("%s:\n", else_end_label);
                            }
                            printf("\n");
```

```
                            break;
            case WHILE_LABEL:
                            {
                                char
while_start_label[]="WHILE_START_LABEL_";
                                char number[10];
                                strcat(while_start_label,itoa(code[i].ar
g, number, 10));
                                printf("%s:\n", while_start_label);
                            }
                            printf("\n");
                            break;
            case WHILE_START:
                            printf("\tmov eax, [ebx-4]\n");
                            printf("\tcmp eax, 0\n");
                            {
                                char
while_end_label[]="WHILE_END_LABEL_";
                                char number[10];
                                strcat(while_end_label,itoa(code[i].arg,
number, 10));
                                printf("\tjle %s\n", while_end_label);
                            }
                            printf("\n");
                            break;
            case WHILE_END:
                            {
                                char
while_start_label[50]="WHILE_START_LABEL_";
                                char
while_end_label[50]="WHILE_END_LABEL_";
                                char number[10];
                                itoa(code[i].arg, number, 10);
                                strcat(while_start_label, number);
```

```
                                printf("\tjmp %s\n", while_start_label);
                                strcat(while_end_label, number);
                                printf("%s:\n", while_end_label);
                            }
                            printf("\n");
                            break;




            case LOOP_INI:
                            {
                                char
for_start_label[]="LOOP_START_LABEL_";
                                char number[10];
                                strcat(for_start_label,itoa(code[i].arg,
number, 10));
                                printf("%s:\n", for_start_label);
                            }
                            printf("\n");
                            break;
            case LOOP_START:
                            printf("\tmov eax, [ebx-4]\n");
                            printf("\tcmp eax, 0\n");
                            {
                                char for_end_label[]="LOOP_END_LABEL_";
                                char number[10];
                                strcat(for_end_label,itoa(code[i].arg,
number, 10));
                                printf("\tjle %s\n", for_end_label);
                            }
                            printf("\n");
                            break;
```

```
            case LOOP_END:
                                {
                                    char
for_start_label[50]="LOOP_START_LABEL_";
                                    char
for_end_label[50]="LOOP_END_LABEL_";
                                    char number[10];
                                    itoa(code[i].arg, number, 10);
                                    strcat(for_start_label, number);
                                    printf("\tjmp %s\n", for_start_label);
                                    strcat(for_end_label, number);
                                    printf("%s:\n", for_end_label);
                                }
                                printf("\n");
                                break;









            default:
                                break;
        }

    }
}
```

Output (Screen/SnapShot):

```
In line no 3, Inserting p with type INT_TYPE in symbol table.
In line no 4, Inserting q with type INT_TYPE in symbol table.
Parsing finished!
============= INTERMEDIATE CODE===============
  0: start          -1
```

```
  1: ld_int            8
  2: store             0
  3: ld_var            0
  4: ld_int           12
  5: ld_int            1
  6: sub              -1
  7: add              -1
  8: store             1
  9: print_int_value     1
 10: halt             -1
============= ASM CODE===============

;start -1
.686
.model flat, c
include C:\masm32\include\msvcrt.inc
includelib C:\masm32\lib\msvcrt.lib

.stack 100h
printf PROTO arg1:Ptr Byte, printlist:VARARG
scanf PROTO arg2:Ptr Byte, inputlist:VARARG

.data
output_integer_msg_format byte "%d", 0Ah, 0
output_string_msg_format byte "%s", 0Ah, 0
input_integer_format byte "%d",0

number sdword ?

.code

main proc
        push ebp
        mov ebp, esp
        sub ebp, 100
        mov ebx, ebp
        add ebx, 4

;ld_int 8
        mov eax, 8
        mov dword ptr [ebx], eax
        add ebx, 4



;store 0
        mov eax, [ebx-4]
```

```
        mov dword ptr [ebp-0], eax

;ld_var 0
        mov eax, [ebp-0]
        mov dword ptr [ebx], eax
        add ebx, 4


;ld_int 12
        mov eax, 12
        mov dword ptr [ebx], eax
        add ebx, 4


;ld_int 1
        mov eax, 1
        mov dword ptr [ebx], eax
        add ebx, 4


;sub -1
        sub ebx, 4
        mov eax, [ebx]
        sub ebx, 4
        mov edx, [ebx]
        sub edx, eax
        mov eax, edx
        mov dword ptr [ebx], eax
        add ebx, 4


;add -1
        sub ebx, 4
        mov eax, [ebx]
        sub ebx, 4
        mov edx, [ebx]
        add eax, edx
        mov dword ptr [ebx], eax
        add ebx, 4


;store 1
        mov eax, [ebx-4]
        mov dword ptr [ebp-4], eax

;print_int_value 1
```

```
        push eax
        push ebx
        push ecx
        push edx
        push [ebp-4]
        push [ebp-0]
        push [ebp+4]
        push [ebp+8]
        push [ebp+12]
        push ebp
        mov eax, [ebp-4]
        INVOKE printf, ADDR output_integer_msg_format, eax
        pop ebp
        pop [ebp+12]
        pop [ebp+8]
        pop [ebp+4]
        pop [ebp-0]
        pop [ebp-4]
        pop edx
        pop ecx
        pop ebx
        pop eax

;halt -1
        add ebp, 100
        mov esp, ebp
        pop ebp
        ret
main endp
end
```

```
Microsoft Windows [Version 10.0.22621.2070]
(c) Microsoft Corporation. All rights reserved.

C:\Users\USER\Desktop\labfinal\LF_1803067_Q1b>C:\masm32\bin\ml /c /coff /Cp dip.asm
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997.  All rights reserved.

 Assembling: dip.asm

C:\Users\USER\Desktop\labfinal\LF_1803067_Q1b>C:\masm32\bin\link -entry:main /subsystem:console dip.obj
Microsoft (R) Incremental Linker Version 5.12.8078
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.


C:\Users\USER\Desktop\labfinal\LF_1803067_Q1b>dip
19
```

Lab Task Q2

Question:

Q2. Consider following Version2 of Code Snippet:

SUB main ()

suru

INT --> p

INT --> q

p soman 8

q soman p jog 12 - 1

case_check_kori q ar

suru

20 : q++

19 : q--

sesh

sesh

Design Lexical Analysis, Syntax Analysis, Intermediate Code Generation and Code Generation part of compiler based on the version2 of code snippet.

Solution (Bold your own written code):

. l file

```
%option noyywrap

%{
    #define UNDEF_TYPE 0
    #define INT_TYPE 1
    #define REAL_TYPE 2
    #define CHAR_TYPE 3
    #define SINGLE_TYPE 4
    #include <stdio.h>
    #include <stdlib.h>
```

```
    #include <string.h>
    #include "parser.tab.h"

    int lineno = 1; // initialize to 1
    void yyerror();
%}

alpha     [a-zA-Z]
digit     [0-9]
alnum     {alpha}|{digit}
print     [ -~]

ID        {alpha}{alnum}*
ICONST    [0-9]{digit}*
%%

"//".*        { }

"INT"       {yylval.int_val=INT_TYPE; return INT; }
"SUB"           { return RETTYPE; }
"suru"          { return SURU ;}
"sesh"          { return SESH ;}
"-->"           {  return CIN ;}
"<--"           { return COUT; }
"case_check_kori" { return CASE;}
":"         { return COLON ;}
"++"        { return INC;}
"--"        { return DEC;}
"ar"        { return AR;}

"jog"       { return ADDOP; }
"-"       { return SUBOP; }
"*"       { return MULOP; }
"/"       { return DIVOP; }
```

```
"=="      { return EQUOP; }
">"       { return GT; }
"<"       { return LT; }

"("       { return LPAREN; }
")"       { return RPAREN; }
"{"       { return LBRACE; }
"}"       { return RBRACE; }
";"       { return SEMI; }
"soman"       { return ASSIGN; }
"print"       { return PRINT; }
"scan"       { return SCAN; }

{ID}        {strcpy(yylval.str_val, yytext); return ID;}
{ICONST}    {yylval.int_val=atoi(yytext); return ICONST;}

"\n"        { lineno += 1; }
[ \t\r\f]+

.       { yyerror("Unrecognized character"); }
```

.y file

```
%{
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include "symtab.c"
    #include "codeGen.c"
    void yyerror();
    extern int lineno;
    extern int yylex();
    /*
```

```
    START,
    HALT,
    LD_INT,               gen_code(LD_INT, data);     push data at
OP-stack
    LD_VAR,               gen_code(LD_VAR, address);  load from V-
stack at address , push at OP-stack
    STORE,                gen_code(STORE, address);   Load
operation stack top data , store varible stack at Address
    SCAN_INT_VALUE,       gen_code(SCAN_INT_VALUE,
address);       Scan data, Store V-stack at address
    PRINT_INT_VALUE,      gen_code(PRINT_INT_VALUE,
address);      load data from V-stack at address , print
    ADD,                  gen_code(ADD, -1)           pop 2 data
from operation stack , ADD them , STORE operation-stack
    SUB,                  gen_code(SUB, -1)
    MUL,                  gen_code(MUL, -1)
    GT_OP,                gen_code(GT_OP,gen_label());
    LT_OP,                gen_code(LT_OP,gen_label());

    IF_START,
    ELSE_START,
    ELSE_END,

    WHILE_LABEL,
    WHILE_START,
    WHILE_END

    LOOP_INI
    LOOP_START
    LOOP_END


    UNDEF_TYPE 0
    INT_TYPE 1
```

```
    REAL_TYPE 2
    CHAR_TYPE 3

    void insert(char* name, int type);
    list_t* search(char *name);
    int idcheck(char* name);
    int gettype(char *name);
    int typecheck(int type1, int type2);

    gen_label()

    */
%}

%union
{
    char str_val[100];
    int int_val;
}

%token PRINT SCAN
%token ADDOP SUBOP MULOP DIVOP EQUOP LT GT
%token LPAREN RPAREN LBRACE RBRACE SEMI ASSIGN
%token<str_val> ID
%token ICONST
%token  INT
%type <int_val> INT  ICONST COLON
%token  RETTYPE SURU  SESH  CIN  COUT CASE COLON INC DEC AR

%left LT GT /*LT GT has lowest precedence*/
%left ADDOP
%left MULOP /*MULOP has lowest precedence*/
```

```
%start program

%%
program:RETTYPE ID LPAREN RPAREN SURU {gen_code(START, -1); }
code  {gen_code(HALT, -1);} SESH ;
code:code stmnt | ;
stmnt : dec | assign | printfun |casechk ;
dec : INT CIN ID {
    insert($3,$1);
} ;
assign : ID ASSIGN exp
    {
        int address = idcheck($1);
        if(address!=-1)
        {
            gen_code(STORE,address);
        }
        else{
             yyerror ();
        }

    };
exp : exp ADDOP exp { gen_code(ADD,-1);}
    | exp SUBOP exp { gen_code(SUB,-1);}
    | T ;
T : ID
    {
        int address = idcheck($1);
        if(address!=-1)
        {
            gen_code(LD_VAR,address);
        }
        else{
             yyerror ();
```

```
        }

    }

    | ICONST
    { gen_code(LD_INT,$1);} ;
printfun : PRINT COUT ID
    {
        int address = idcheck($3);
        if(address!=-1)
        {
            gen_code(PRINT_INT_VALUE,address);
        }
        else{
             yyerror ();
        }

    };
casechk : CASE ID AR SURU ICONST COLON ID INC ICONST COLON ID DEC
SESH {

    $6=gen_label();
    $10=gen_label();
    int address=idcheck($2);
    if(address!=-1)
    {
        // for case 20 : q++
        gen_code(LD_VAR,address);
        gen_code(LD_INT,$5);
        gen_code(EQL_OP,gen_label());
        gen_code(IF_START,$6);

        gen_code(LD_VAR,address);
        gen_code(LD_INT,1);
```

```
        gen_code(ADD,-1);
        gen_code(STORE,address);

        gen_code(ELSE_START,$6);
        gen_code(ELSE_END,$6);


        //19 : q--
        gen_code(LD_VAR,address);
        gen_code(LD_INT,$9);
        gen_code(EQL_OP,gen_label());
        gen_code(IF_START,$10);

        gen_code(LD_VAR,address);
        gen_code(LD_INT,1);
        gen_code(SUB,-1);
        gen_code(STORE,address);

        gen_code(ELSE_START,$10);
        gen_code(ELSE_END,$10);



    }
    else{
        yyerror ();
    }

} ;
%%

void yyerror ()
{
    printf("Syntax error at line %d\n", lineno);
```

```
    exit(1);
}

int main (int argc, char *argv[])
{
    yyparse();
    printf("Parsing finished!\n");

    printf("============= INTERMEDIATE CODE==============\n");
    print_code();

    printf("============= ASM CODE==============\n");
    print_assembly();

    return 0;
}
```

Gencode.c

```
#include "codeGen.h"

int gen_label()
{
    return code_offset;
}

void gen_code(enum code_ops op, int arg)
{
    code[code_offset].op = op;
    code[code_offset].arg = arg;

    code_offset++;
}
```

```
void print_code()
{
    int i = 0;

    for(i=0; i<code_offset; i++)
    {
        printf("%3d: %-15s  %4d\n", i, op_name[code[i].op],
code[i].arg);
    }
}

void print_assembly()
{
    int i = 0;
    int j = 0;

    int stack_variable_counter = 0;

    for(i=0; i<code_offset; i++)
    {
        printf("\n;%s %d\n", op_name[code[i].op], code[i].arg);

        if(code[i].op == LD_INT || code[i].op == LD_VAR)
            stack_variable_counter++;

        if(code[i].op == ADD)
            stack_variable_counter--;

        switch(code[i].op)
        {
            case START:
                            printf(".686\n");
                            printf(".model flat, c\n");
```

```
                            printf("include
C:\\masm32\\include\\msvcrt.inc\n");
                            printf("includelib
C:\\masm32\\lib\\msvcrt.lib\n");
                            printf("\n");
                            printf(".stack 100h\n");
                            printf("printf PROTO arg1:Ptr Byte,
printlist:VARARG\n");
                            printf("scanf PROTO arg2:Ptr Byte,
inputlist:VARARG\n");
                            printf("\n");
                            printf(".data\n");
                            printf("output_integer_msg_format byte
\"\% %d\", 0Ah, 0\n");
                            printf("output_string_msg_format byte
\"\% %s\", 0Ah, 0\n");
                            printf("input_integer_format byte \"\%
%d\",0\n");
                            printf("\n");
                            printf("number sdword ?\n");
                            printf("\n");
                            printf(".code\n");
                            printf("\n");
                            printf("main proc\n");
                            printf("\tpush ebp\n");
                            printf("\tmov ebp, esp\n");
                            printf("\tsub ebp, 100\n");
                            printf("\tmov ebx, ebp\n");
                            printf("\tadd ebx, 4\n");
                            break;

            case HALT:
                            printf("\tadd ebp, 100\n");
                            printf("\tmov esp, ebp\n");
```

```
                        printf("\tpop ebp\n");
                        printf("\tret\n");
                        printf("main endp\n");
                        printf("end\n");
                        break;

            case STORE:
                        printf("\tmov eax, [ebx-4]\n");
                        printf("\tmov dword ptr [ebp-%d],
eax\n", 4*code[i].arg);
                        break;

            case SCAN_INT_VALUE:
                        printf("\tpush eax\n");
                        printf("\tpush ebx\n");
                        printf("\tpush ecx\n");
                        printf("\tpush edx\n");
                        for(j=address-1; j>=0; j--)
                            printf("\tpush [ebp-%d]\n", 4*j);
                        for(j=1; j<=stack_variable_counter;
j++)
                            printf("\tpush [ebp+%d]\n", 4*j);
                        printf("\tpush ebp\n");

                        printf("\tINVOKE scanf, ADDR
input_integer_format, ADDR number\n");

                        printf("\tpop ebp\n");
                        for(j=stack_variable_counter; j>=1; j-
-)
                            printf("\tpop [ebp+%d]\n", 4*j);
                        for(j=0; j<=address-1; j++)
                            printf("\tpop [ebp-%d]\n", 4*j);
```

```
                            printf("\tmov eax, number\n");
                            printf("\tmov dword ptr [ebp-%d],
eax\n", 4*code[i].arg);

                            printf("\tpop edx\n");
                            printf("\tpop ecx\n");
                            printf("\tpop ebx\n");
                            printf("\tpop eax\n");
                            break;


            case PRINT_INT_VALUE:
                            printf("\tpush eax\n");
                            printf("\tpush ebx\n");
                            printf("\tpush ecx\n");
                            printf("\tpush edx\n");
                            for(j=address-1; j>=0; j--)
                                printf("\tpush [ebp-%d]\n", 4*j);
                            for(j=1; j<=stack_variable_counter;
j++)
                                printf("\tpush [ebp+%d]\n", 4*j);
                            printf("\tpush ebp\n");

                            printf("\tmov eax, [ebp-%d]\n",
4*code[i].arg);
                            printf("\tINVOKE printf, ADDR
output_integer_msg_format, eax\n");

                            printf("\tpop ebp\n");
                            for(j=stack_variable_counter; j>=1; j-
-)
                                printf("\tpop [ebp+%d]\n", 4*j);
                            for(j=0; j<=address-1; j++)
                                printf("\tpop [ebp-%d]\n", 4*j);
```

```
                        printf("\tpop edx\n");
                        printf("\tpop ecx\n");
                        printf("\tpop ebx\n");
                        printf("\tpop eax\n");
                        break;

            case LD_VAR:
                        printf("\tmov eax, [ebp-%d]\n",
4*code[i].arg);
                        printf("\tmov dword ptr [ebx],
eax\n");
                        printf("\tadd ebx, 4\n");
                        printf("\n");
                        break;

            case LD_INT:
                        printf("\tmov eax, %d\n",
code[i].arg);
                        printf("\tmov dword ptr [ebx],
eax\n");
                        printf("\tadd ebx, 4\n");
                        printf("\n");
                        break;

            case ADD:
                        printf("\tsub ebx, 4\n");
                        printf("\tmov eax, [ebx]\n");
                        printf("\tsub ebx, 4\n");
                        printf("\tmov edx, [ebx]\n");
                        printf("\tadd eax, edx\n");
                        printf("\tmov dword ptr [ebx],
eax\n");
                        printf("\tadd ebx, 4\n");
                        printf("\n");
```

```
                        break;
            case SUB:
                        printf("\tsub ebx, 4\n");
                        printf("\tmov eax, [ebx]\n");
                        printf("\tsub ebx, 4\n");
                        printf("\tmov edx, [ebx]\n");
                        printf("\tsub edx, eax\n");
                        printf("\tmov eax, edx\n");
                        printf("\tmov dword ptr [ebx],
eax\n");
                        printf("\tadd ebx, 4\n");
                        printf("\n");
                        break;
            case MUL:
                        printf("\tsub ebx, 4\n");
                        printf("\tmov eax, [ebx]\n");
                        printf("\tsub ebx, 4\n");
                        printf("\tmov edx, [ebx]\n");
                        printf("\tIMUL eax, edx\n");
                        printf("\tmov dword ptr [ebx],
eax\n");
                        printf("\tadd ebx, 4\n");
                        printf("\n");
                        break;
            case GT_OP:
                        printf("\tsub ebx, 4\n");
                        printf("\tmov eax, [ebx]\n");
                        printf("\tsub ebx, 4\n");
                        printf("\tmov edx, [ebx]\n");
                        printf("\tcmp edx, eax\n");

                        {
                            char relop_start_label[50]="LS";
                            char relop_end_label[50]="LE";
```

```
                                char number[10];
                                itoa(code[i].arg, number, 10);
                                strcat(relop_end_label, number);
                                strcat(relop_start_label, number);

                                printf("\tjg %s\n",
relop_start_label);
                                printf("\tmov dword ptr [ebx],
0\n");
                                printf("\tjmp %s\n",
relop_end_label);
                                printf("\t%s: mov dword ptr [ebx],
1\n", relop_start_label);
                                printf("\t%s: add ebx, 4\n\n",
relop_end_label);
                            }
                            printf("\n");
                            break;
                    case LT_OP:
                            printf("\tsub ebx, 4\n");
                            printf("\tmov eax, [ebx]\n");
                            printf("\tsub ebx, 4\n");
                            printf("\tmov edx, [ebx]\n");
                            printf("\tcmp edx, eax\n");

                            {
                                char relop_start_label[50]="LS";
                                char relop_end_label[50]="LE";
                                char number[10];
                                itoa(code[i].arg, number, 10);
                                strcat(relop_end_label, number);
                                strcat(relop_start_label, number);
```

```
                                printf("\tjl %s\n",
relop_start_label);
                                printf("\tmov dword ptr [ebx],
0\n");
                                printf("\tjmp %s\n",
relop_end_label);
                                printf("\t%s: mov dword ptr [ebx],
1\n", relop_start_label);
                                printf("\t%s: add ebx, 4\n\n",
relop_end_label);
                            }
                            printf("\n");
                            break;
                case LTE_OP:
                            printf("\tsub ebx, 4\n");
                            printf("\tmov eax, [ebx]\n");
                            printf("\tsub ebx, 4\n");
                            printf("\tmov edx, [ebx]\n");
                            printf("\tcmp edx, eax\n");

                            {
                                char relop_start_label[50]="LS";
                                char relop_end_label[50]="LE";
                                char number[10];
                                itoa(code[i].arg, number, 10);
                                strcat(relop_end_label, number);
                                strcat(relop_start_label, number);

                                printf("\tjle %s\n",
relop_start_label);
                                printf("\tmov dword ptr [ebx],
0\n");
                                printf("\tjmp %s\n",
relop_end_label);
```

```
                                printf("\t%s: mov dword ptr [ebx],
1\n", relop_start_label);
                                printf("\t%s: add ebx, 4\n\n",
relop_end_label);
                            }
                            printf("\n");
                            break;
            case EQL_OP:
                            printf("\tsub ebx, 4\n");
                            printf("\tmov eax, [ebx]\n");
                            printf("\tsub ebx, 4\n");
                            printf("\tmov edx, [ebx]\n");
                            printf("\tcmp edx, eax\n");

                            {
                                char relop_start_label[50]="LS";
                                char relop_end_label[50]="LE";
                                char number[10];
                                itoa(code[i].arg, number, 10);
                                strcat(relop_end_label, number);
                                strcat(relop_start_label, number);

                                printf("\tje %s\n",
relop_start_label);
                                printf("\tmov dword ptr [ebx],
0\n");
                                printf("\tjmp %s\n",
relop_end_label);
                                printf("\t%s: mov dword ptr [ebx],
1\n", relop_start_label);
                                printf("\t%s: add ebx, 4\n\n",
relop_end_label);
                            }
                            printf("\n");
```

```
                            break;
            case IF_START:
                            printf("\tmov eax, [ebx-4]\n");
                            printf("\tcmp eax, 0\n");
                            {
                                char
else_start_label[]="ELSE_START_LABEL_";
                                char number[10];
                                strcat(else_start_label,itoa(code[
i].arg, number, 10));
                                printf("\tjle %s\n",
else_start_label);
                            }
                            printf("\n");
                            break;
            case ELSE_START:
                            {
                                char
else_start_label[50]="ELSE_START_LABEL_";
                                char
else_end_label[50]="ELSE_END_LABEL_";
                                char number[10];
                                itoa(code[i].arg, number, 10);
                                strcat(else_end_label, number);
                                printf("\tjmp %s\n",
else_end_label);
                                strcat(else_start_label, number);
                                printf("%s:\n", else_start_label);
                            }
                            printf("\n");
                            break;
            case ELSE_END:
                            {
```

```
                                char
else_end_label[50]="ELSE_END_LABEL_";
                                char number[10];
                                itoa(code[i].arg, number, 10);
                                strcat(else_end_label, number);
                                printf("%s:\n", else_end_label);
                            }
                            printf("\n");
                            break;
            case WHILE_LABEL:
                            {
                                char
while_start_label[]="WHILE_START_LABEL_";
                                char number[10];
                                strcat(while_start_label,itoa(code
[i].arg, number, 10));
                                printf("%s:\n",
while_start_label);
                            }
                            printf("\n");
                            break;
            case WHILE_START:
                            printf("\tmov eax, [ebx-4]\n");
                            printf("\tcmp eax, 0\n");
                            {
                                char
while_end_label[]="WHILE_END_LABEL_";
                                char number[10];
                                strcat(while_end_label,itoa(code[i
].arg, number, 10));
                                printf("\tjle %s\n",
while_end_label);
                            }
                            printf("\n");
```

```
                            break;
            case WHILE_END:
                            {
                                char
while_start_label[50]="WHILE_START_LABEL_";
                                char
while_end_label[50]="WHILE_END_LABEL_";
                                char number[10];
                                itoa(code[i].arg, number, 10);
                                strcat(while_start_label, number);
                                printf("\tjmp %s\n",
while_start_label);
                                strcat(while_end_label, number);
                                printf("%s:\n", while_end_label);
                            }
                            printf("\n");
                            break;




            case LOOP_INI:
                            {
                                char
for_start_label[]="LOOP_START_LABEL_";
                                char number[10];
                                strcat(for_start_label,itoa(code[i
].arg, number, 10));
                                printf("%s:\n", for_start_label);
                            }
                            printf("\n");
                            break;
            case LOOP_START:
```

```
                            printf("\tmov eax, [ebx-4]\n");
                            printf("\tcmp eax, 0\n");
                            {
                                char
for_end_label[]="LOOP_END_LABEL_";
                                char number[10];
                                strcat(for_end_label,itoa(code[i].
arg, number, 10));
                                printf("\tjle %s\n",
for_end_label);
                            }
                            printf("\n");
                            break;
            case LOOP_END:
                            {
                                char
for_start_label[50]="LOOP_START_LABEL_";
                                char
for_end_label[50]="LOOP_END_LABEL_";
                                char number[10];
                                itoa(code[i].arg, number, 10);
                                strcat(for_start_label, number);
                                printf("\tjmp %s\n",
for_start_label);
                                strcat(for_end_label, number);
                                printf("%s:\n", for_end_label);
                            }
                            printf("\n");
                            break;
```

```
            default:
                                break;
        }

    }
}
```

Output (Screen/SnapShot):

```
In line no 3, Inserting p with type INT_TYPE in symbol table.
In line no 4, Inserting q with type INT_TYPE in symbol table.
Parsing finished!
============= INTERMEDIATE CODE===============
  0: start            -1
  1: ld_int            8
  2: store             0
  3: ld_var            0
  4: ld_int           12
  5: ld_int            1
  6: sub              -1
  7: add              -1
  8: store             1
  9: ld_var            1
 10: ld_int           20
 11: eql              11
 12: if_start          9
 13: ld_var            1
 14: ld_int            1
 15: add              -1
 16: store             1
 17: else_start        9
 18: else_end          9
 19: ld_var            1
 20: ld_int           19
 21: eql              21
 22: if_start          9
 23: ld_var            1
 24: ld_int            1
 25: sub              -1
 26: store             1
 27: else_start        9
 28: else_end          9
 29: halt            -1
============= ASM CODE===============

;start -1
```

```
.686
.model flat, c
include C:\masm32\include\msvcrt.inc
includelib C:\masm32\lib\msvcrt.lib

.stack 100h
printf PROTO arg1:Ptr Byte, printlist:VARARG
scanf PROTO arg2:Ptr Byte, inputlist:VARARG

.data
output_integer_msg_format byte "%d", 0Ah, 0
output_string_msg_format byte "%s", 0Ah, 0
input_integer_format byte "%d",0

number sdword ?

.code

main proc
        push ebp
        mov ebp, esp
        sub ebp, 100
        mov ebx, ebp
        add ebx, 4

;ld_int 8
        mov eax, 8
        mov dword ptr [ebx], eax
        add ebx, 4


;store 0
        mov eax, [ebx-4]
        mov dword ptr [ebp-0], eax

;ld_var 0
        mov eax, [ebp-0]
        mov dword ptr [ebx], eax
        add ebx, 4


;ld_int 12
        mov eax, 12
        mov dword ptr [ebx], eax
        add ebx, 4


;ld_int 1
        mov eax, 1
        mov dword ptr [ebx], eax
        add ebx, 4


;sub -1
        sub ebx, 4
```

```
        mov eax, [ebx]
        sub ebx, 4
        mov edx, [ebx]
        sub edx, eax
        mov eax, edx
        mov dword ptr [ebx], eax
        add ebx, 4


;add -1
        sub ebx, 4
        mov eax, [ebx]
        sub ebx, 4
        mov edx, [ebx]
        add eax, edx
        mov dword ptr [ebx], eax
        add ebx, 4


;store 1
        mov eax, [ebx-4]
        mov dword ptr [ebp-4], eax

;ld_var 1
        mov eax, [ebp-4]
        mov dword ptr [ebx], eax
        add ebx, 4


;ld_int 20
        mov eax, 20
        mov dword ptr [ebx], eax
        add ebx, 4


;eql 11
        sub ebx, 4
        mov eax, [ebx]
        sub ebx, 4
        mov edx, [ebx]
        cmp edx, eax
        je LS11
        mov dword ptr [ebx], 0
        jmp LE11
        LS11: mov dword ptr [ebx], 1
        LE11: add ebx, 4



;if_start 9
        mov eax, [ebx-4]
        cmp eax, 0
        jle ELSE_START_LABEL_9
```

```
;ld_var 1
        mov eax, [ebp-4]
        mov dword ptr [ebx], eax
        add ebx, 4


;ld_int 1
        mov eax, 1
        mov dword ptr [ebx], eax
        add ebx, 4


;add -1
        sub ebx, 4
        mov eax, [ebx]
        sub ebx, 4
        mov edx, [ebx]
        add eax, edx
        mov dword ptr [ebx], eax
        add ebx, 4


;store 1
        mov eax, [ebx-4]
        mov dword ptr [ebp-4], eax

;else_start 9
        jmp ELSE_END_LABEL_9
ELSE_START_LABEL_9:


;else_end 9
ELSE_END_LABEL_9:


;ld_var 1
        mov eax, [ebp-4]
        mov dword ptr [ebx], eax
        add ebx, 4


;ld_int 19
        mov eax, 19
        mov dword ptr [ebx], eax
        add ebx, 4


;eql 21
        sub ebx, 4
        mov eax, [ebx]
        sub ebx, 4
        mov edx, [ebx]
        cmp edx, eax
        je LS21
        mov dword ptr [ebx], 0
```

```
        jmp LE21
        LS21: mov dword ptr [ebx], 1
        LE21: add ebx, 4



;if_start 9
        mov eax, [ebx-4]
        cmp eax, 0
        jle ELSE_START_LABEL_9


;ld_var 1
        mov eax, [ebp-4]
        mov dword ptr [ebx], eax
        add ebx, 4


;ld_int 1
        mov eax, 1
        mov dword ptr [ebx], eax
        add ebx, 4


;sub -1
        sub ebx, 4
        mov eax, [ebx]
        sub ebx, 4
        mov edx, [ebx]
        sub edx, eax
        mov eax, edx
        mov dword ptr [ebx], eax
        add ebx, 4


;store 1
        mov eax, [ebx-4]
        mov dword ptr [ebp-4], eax

;else_start 9
        jmp ELSE_END_LABEL_9
ELSE_START_LABEL_9:


;else_end 9
ELSE_END_LABEL_9:


;halt -1
        add ebp, 100
        mov esp, ebp
        pop ebp
        ret
main endp
end
```