**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):**

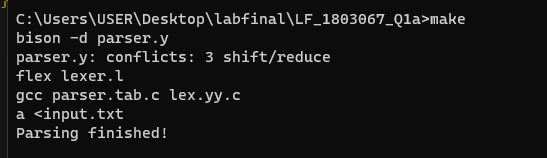
.l file

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

****

**Lab Task Q1b**

**Question:**

**Solution (Bold your own written code):**

.y file

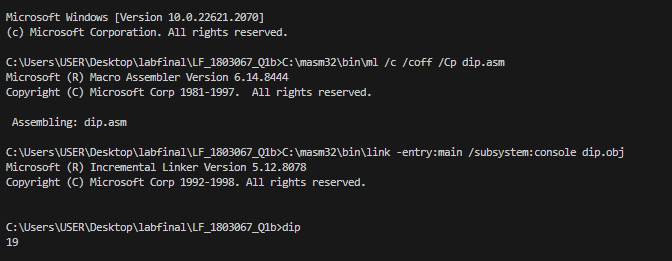
|  |
| --- |
| %{      #include <stdio.h>      #include <stdlib.h>      #include <string.h>      #include "symtab.c"      #include "codeGen.c"      void yyerror();      extern int lineno;      extern int yylex();    %}  %union  {      char str\_val[100];      int int\_val;  }  %token PRINT SCAN  %token 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].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: 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** |

****

**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 |