Compiler Design Lab

3-Address Code Generation

Submitted By :
Shishir Gangwar (15CO147)
Saumyadip Mandal (15CO144)

Submitted To :
Ms.Sushmita,
Faculty, Dept. of Computer Science,
NITK, Surathkal.

Index

Introduction

Code generation is the process by which a compiler’s code generator converts some intermediate representation of source code into a form (e.g., machine code) that can be readily executed by a machine.

Compilers typically perform multiple passes over various intermediate forms. This multi-stage process is used because many algorithms for code optimization are easier to apply one at a time, or because the input to one optimization relies on the completed processing performed by another optimization. This organization also facilitates the creation of a single compiler that can target multiple architectures, as only the last of the code generation stages (the backend) needs to change from target to target.The input to the code generator typically consists of a parse tree or an abstract syntax tree. The tree is converted into a linear sequence of instructions, usually in an intermediate language such as three-address code. Further stages of compilation may or may not be referred to as "code generation", depending on whether they involve a significant change in the representation of the program.When code generation occurs at runtime, as in just-in-time compilation (JIT), it is important that the entire process be efficient with respect to space and time. For example, when regular expressions are interpreted and used to generate code at runtime, a non-deterministic finite state machine is often generated instead of a deterministic one, because usually the former can be created more quickly and occupies less memory space than the latter. Despite its generally generating less efficient code, JIT code generation can take advantage of profiling information that is available only at runtime.

Role Of Intermediate Code Generator

Jobs which are typically part of a compiler's "code generation" phase include:

- Instruction selection: which instructions to use.
- Instruction scheduling: in which order to put those instructions. Scheduling is a speed optimization that can have a critical effect on pipelined machines.
- Register allocation: the allocation of variables to processor registers
- Debug data generation if required so the code can be debugged.

Instruction selection is typically carried out by doing a recursive postorder traversal on the abstract syntax tree, matching particular tree configurations against templates; for example, the tree W := ADD(X,MUL(Y,Z)) might be transformed into a linear sequence of instructions by recursively generating the sequences for t1 := X and t2 := MUL(Y,Z), and then emitting the instruction ADD W , t1 , t2.

Declarations In ICG Phase

A variable or procedure has to be declared before it can be used. Declaration involves allocation of space in memory and entry of type and name in the symbol table. A program may be coded and designed keeping the target machine structure in mind, but it may not always be possible to accurately convert a source code to its target language. Taking the whole program as a collection of procedures and sub-procedures, it becomes possible to declare all the names local to the procedure. Memory allocation is done in a consecutive manner and names are allocated to memory in the sequence they are declared in the program. We use offset variable and set it to zero {offset = 0} that denote the base address. The source programming language and the target machine architecture may vary in the way names are stored, so relative addressing is used. While the first name is allocated memory starting from the memory location 0 {offset=0}, the next name declared later, should be allocated memory next to the first one.

Intermediate Representation Of ICG

Intermediate codes can be represented in a variety of ways :

- High Level IR - High-level intermediate code representation is very close to the source language itself. They can be easily generated from the source code and we can easily apply code modifications to enhance performance.
- Low Level IR - This one is close to the target machine, which makes it suitable for register and memory allocation, instruction set selection, etc. It is good for machine-dependent optimizations.

Intermediate code can be either language specific (e.g., Bytecode for Java) or language independent (three-address code).

Lex Code

```
%{
    #include<bits/stdc++.h>
    vector<string>lines;
    using namespace std;
    int charcount=0;
    string str;
    string func;
    string fname;
    vector<string>fargs;
      int lineno=1;
%}
identifier      [a-zA-Z][_a-zA-Z0-9]*
header          "#include"[ ]*["<""/""]{identifier}".h"?[">""/""]
type
"const"|"short"|"signed"|"unsigned"|"int"|"float"|"long"|"double"|"ch
ar"|"void"
storage         "auto"|"extern"|"register"|"static"|"typedef"
qualifier  "const"|"volatile"
digits          [0-9]+
decimal    0|[1-9][0-9]*
lint       {decimal}"L"
llint      {decimal}"LL"
double          {decimal}?"."{digits}
float           {double}"f"
scientific   {double}"e"{decimal}
scientificf  {scientific}"f"
str_literal  [a-zA-Z_]?\"(\\.|[^\\"])*"\""
character  "'"."'"
space      " "
tab              \t
next_line  \n
s_operator      "["|"]"|","|":"|"{"|"}"|"("|")"|"="|[-+*%/<>&|^!]

%x mlcomment
%x slcomment
%%
"/*"            {charcount+=yyleng;BEGIN(mlcomment);}
<mlcomment>[^*\n]*   {           charcount+=yyleng;;}
<mlcomment>\n         {     charcount=0;++lineno;}
<mlcomment>"*"+[^/]   {           charcount+=yyleng;;}
<mlcomment>"*"+"/"   { charcount+=yyleng;BEGIN(INITIAL);}

"//"            {charcount+=yyleng;BEGIN(slcomment);}
<slcomment>[^\n]*     {charcount+=yyleng;        ;}
<slcomment>\n         {charcount+=yyleng; BEGIN(INITIAL);}
```

```
{tab}                       {charcount+=1;}
{space}                 {charcount+=1;}
{next_line}             {charcount=0;lineno++;}
{header}                {charcount+=yyleng;;}
"long long int"             {charcount+=yyleng;yylval.s = 
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);str = 
yytext;return TYPE;}
"long int"                  {charcount+=yyleng;yylval.s = 
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);str = 
yytext;return TYPE;}
"short int"                 {charcount+=yyleng;yylval.s = 
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);str = 
yytext;return TYPE;}
{type}                  {charcount+=yyleng;yylval.s = 
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);str = 
yytext;return TYPE;}
{storage}               {charcount+=yyleng;yylval.s = 
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return 
STORAGE;}
{qualifier}             {charcount+=yyleng;yylval.s = 
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return 
QUALIFIER;}
printf                  {charcount+=yyleng;yylval.s = 
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return PRINTF;}
scanf                   {charcount+=yyleng;yylval.s = 
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return SCANF;}
while                   {charcount+=yyleng;yylval.s = 
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return WHILE;}
if                      {charcount+=yyleng;yylval.s = 
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return IF;}
else                    {charcount+=yyleng;yylval.s = 
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return ELSE;}
return                  {charcount+=yyleng;yylval.s = 
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return RETURN;}
do                      {charcount+=yyleng;yylval.s = 
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return DO;}
continue                {charcount+=yyleng;yylval.s = 
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return
CONTINUE;}
break                   {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return BREAK;}
goto                    {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return GOTO;}
default                 {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return
DEFAULT;}
enum                    {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return ENUM;}
case                    {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return CASE;}
switch                  {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return SWITCH;}
for                     {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return FOR;}
sizeof                  {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return SIZEOF;}
struct                  {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return STRUCT;}
union                   {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return UNION;}
{decimal}               {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return NUM;}
{float}                 {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return NUM;}
{double}                {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return NUM;}
{llint}                 {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return NUM;}
{lint}                  {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return NUM;}
{scientific}            {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return NUM;}
{scientificf}           {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return NUM;}
{character}             {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return STR;}
```

```
{str_literal}          {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return STR;}
{identifier}           {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return ID;}
"+="                   {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return PE;}
"-="                   {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return MI;}
"*="                   {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return ME;}
"/="                   {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return DE;}
">>"                   {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return RS;}
"<<"                   {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return LS;}
"++"                   {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return PP;}
"--"                   {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return MM;}
"=="                   {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return EE;}
"!="                   {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return NE;}
">="                   {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return GE;}
"<="                   {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return LE;}
"||"                   {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return LO;}
"&&"                   {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return LA;}

{s_operator}           {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return
yytext[0];}
";"                    {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return
yytext[0];}
.                     {charcount+=yyleng;printf("error at line no
%d",lineno,"NA");}
%%
int yywrap(){return 1;}
```

Yacc Code

```
%{
     #include <bits/stdc++.h>
     #include "symbol.cpp"
     using namespace std;
     int yylex(void);
    void yyerror(char *);
    void trim(string& s)
      {    size_t p = s.find_first_not_of(" \t");
      s.erase(0, p);
           p = s.find_last_not_of(" \t");
      if (string::npos != p)
     s.erase(p+1);
      }
      int first,chc,lno;string typ,dtyp;
    string filename;
    extern string fname;
    extern vector<string>fargs;
    extern int charcount;
    extern string str;
    extern string func;
     extern int lineno;
     int ff=0,fflag=0,dd=0;
     val *fornode;
%}
%union{
    char *s;
    val *t;
    int i;
}
%token NUM INT LONG VOID ID
%token STR STORAGE QUALIFIER TYPE
%token FLOAT DOUBLE BOOL CHAR
%token IF ELSE WHILE RETURN SCANF
%token GE ">=" LE "<=" EE "==" NE "!=" LO "||" LA "&&" PP "++" MM
"--"
%token PE "+=" MI "-=" ME "*=" DE "/=" LS "<<" RS ">>"
%token PRINTF GOTO CONTINUE DEFAULT BREAK ENUM DO
%token CASE SWITCH FOR SIZEOF STRUCT UNION
%right '='
%right "+=" "-=" "*=" "/="
%left "||"
%left "&&"
%left '|'
%left '&'
%left "==" "!="
%left '<' ">=" '>' "<="
%left "<<" ">>"
%left '+' '-'
%left '*' '/' '%'
%right '!'
%left '(' ')' '[' ']' "++" "--"
%%
start:           function {temp = head; func = "";} start
|                declaration {temp = head;} start
|
;
    function:         TYPE ID '(' {
insVar("function",$<s>1,$<s>2,lineno,charcount,true);lno=lineno;func
= $<s>2;chc=charcount;typ=$<s>2;dtyp=$<s>1;} declordef
                             {if(fflag)
search(head,"function",typ,dtyp,chc);ff=0;fflag=0;}
    ;
      declordef:      paramdecls ')' {tac_func(func);}compstmt
{temp->func_def_flag=0;tac_func(func,0);}
      |               fparamdecls ')' ';'
```

```
{temp->func_def_flag=1;}
       |                paramdecls ')' ';'    {temp->func_def_flag=1;}
       ;
       paramdecls:     paramdecl
       |
       ;
             paramdecl:     param ',' paramdecl
             |              param
             ;
                   param:    TYPE ID         {
if(temp->func_def_flag==0)insVar("argument",$<s>1,$<s>2,lineno,charco
unt);
                                                            else{

if(!fflag && ff<temp->token.size() && temp->datatype[ff]==$<s>1){

       temp->type[ff] = $<s>2;temp->position[ff]=charcount;++ff;
                                                                  }

else{

       fflag=1;
                                                                  }
                                                            }
                                                      }
                   ;
       fparamdecls:    fparamdecl
       ;
             fparamdecl:    fparam ',' fparamdecl
             |              fparam
             |                fparam ',' paramdecl
             |                param ',' fparamdecl
             ;
                   fparam:   TYPE     {
insVar("argument",$<s>1,"---",lineno,charcount);}
                   ;
    declaration:      declstmt ';'
    |                 expr ';'    {bfs($<t>1);tac_exp();}
    ;


       declstmt:      TYPE decllist
       ;
            decllist:      ID assign {
insVar("variable",str,$<s>1,lineno,charcount);
                                                         val *x1 =
mknode(NULL,$<s>1,NULL);if($<t>2)$<t>$ = mknode(x1,"=",$<t>2);else
$<t>$ = NULL;

bfs($<t>$);tac_exp();
                                                    } ',' decllist
            |              ID assign {
insVar("variable",str,$<s>1,lineno,charcount);
                                                         val *x1 =
mknode(NULL,$<s>1,NULL);if($<t>2)$<t>$ = mknode(x1,"=",$<t>2);else
$<t>$ = NULL;

bfs($<t>$);tac_exp();
                                                    }
            |              ID arraydecl ',' decllist    {
insVar("variable",str,$<s>1,lineno,charcount);checkDim();fargs.clear(
);dd=0;}
            |              ID arraydecl     {
insVar("variable",str,$<s>1,lineno,charcount);checkDim();fargs.clear(
);dd=0;}
            |              ID arraydecl'=' '{' arrassign '}' {
insVar("variable",str,$<s>1,lineno,charcount);checkDim();fargs.clear(
);dd=0;}
            ;
                 arraydecl:     '[' const ']' {++dd;} arraydecl

                 |               '[' const ']'
{++dd;}
                 ;
                 arrassign:      NUM ',' arrassign
                 |              NUM
                 ;
                 assign:        '=' expr {$<t>$ = $<t>2;}
                 |              {$<t>$ = NULL;}
                 ;
stmt:           compstmt
|               forstmt
|               whilestmt
|               dowhilestmt
|               ifstmt
|               returnstmt
|               declaration
|               ';'
;
    compstmt:        '{' { if(temp!=head && temp->first!=1)
insVar("_block_","null","    null",lineno,charcount,true); else
if(temp!=head) ++temp->first;} stmtlist '}' {if(temp!=head)temp =
temp->parent;}
    ;
      stmtlist:      stmt stmtlist
      |
      ;
    forstmt:    FOR   {$<i>1=tac_while(1);} '(' fexp ';' fexp ';'
{tac_while(2);} fexpt ')' stmt
{bfs(fornode);tac_exp();tac_while(3,$<i>1);}
    ;
      fexp:          expt
      |
      ;
      fexpt:         expr     {fornode = $<t>1;}
      |              {fornode = NULL;}
      ;
    whilestmt:       WHILE {$<i>1=tac_while(1);}'(' expt ')'
{tac_while(2);} stmt {tac_while(3,$<i>1);}
    ;
    dowhilestmt:     DO {$<i>1=tac_while(1);} compstmt WHILE '('
expt ')' {tac_while(2);tac_while(3,$<i>1);} ';'
    ;
    ifstmt:          IF '(' expt ')'  {$<i>1=tac_if(1);} stmt
```

```
{tac_if(0,$<i>1);}
    ;
    returnstmt:       RETURN {checkReturn(1);} ';'
    |                 RETURN expr ';'
{checkReturn(0);bfs($<t>2);tac_ret();}
    ;
    expt:          expr {bfs($<t>1);tac_exp();}
    ;
functcall:      ID  '(' argdecls ')'   {fname=$<s>1;int f1
=paramcheck();tac_call(fname,0,f1);}
;
    argdecls:    argdecl
    |
    ;
      argdecl:         const  ',' argdecl
      |                const
      ;
           const:           NUM
{tac_call($<s>1);fargs.push_back($<s>1);}
           |                     ID
{tac_call($<s>1);fargs.push_back($<s>1);}
           ;
expr :          '(' expr ')'    {   $<t>$ = $<t>2;}
|               ID '=' expr      {    scope(temp,$<s>1,charcount);
                                           $<t>1 =
mknode(NULL,$<s>1,NULL);
                                           $<t>$ =
mknode($<t>1,$<s>2,$<t>3);
                                      }
|                 ID '=' error
|               ID "+=" expr     {    scope(temp,$<s>1,charcount);
                                           val *x1 =
mknode(NULL,$<s>1,NULL);
                                           val *x2 =
mknode(x1,"+",$<t>3);
                                           val *x3 =
mknode(NULL,$<s>1,NULL);
                                           $<t>$ =
mknode(x3,"=",x2);
                                                }
|                 ID "+=" error
|               ID "-=" expr     {    scope(temp,$<s>1,charcount);
                                                      val *x1 =
mknode(NULL,$<s>1,NULL);
                                                      val *x2 =
mknode(x1,"+",$<t>3);
                                                      val *x3 =
mknode(NULL,$<s>1,NULL);
                                                      $<t>$ =
mknode(x3,"=",x2);
                                                      }
|                 ID "-=" error
|               ID "*=" expr     {    scope(temp,$<s>1,charcount);
                                                      val *x1 =
mknode(NULL,$<s>1,NULL);
                                                      val *x2 =
mknode(x1,"+",$<t>3);
                                                      val *x3 =
mknode(NULL,$<s>1,NULL);
                                                      $<t>$ =
mknode(x3,"=",x2);
                                                }
|                 ID "*=" error
|               ID "/=" expr     {    scope(temp,$<s>1,charcount);
                                                      val *x1 =
mknode(NULL,$<s>1,NULL);
                                                      val *x2 =
mknode(x1,"+",$<t>3);
                                                      val *x3 =
mknode(NULL,$<s>1,NULL);
                                                      $<t>$ =
mknode(x3,"=",x2);
                                                }
|                 ID "/=" error
|               expr '+' expr     {    $<t>$ =
mknode($<t>1,$<s>2,$<t>3);}| expr '+' error
```

```
|               expr '-' expr     {    $<t>$ = 
mknode($<t>1,$<s>2,$<t>3);}| expr '-' error
|               expr '*' expr     {    $<t>$ = 
mknode($<t>1,$<s>2,$<t>3);}| expr '*' error
|               expr '/' expr     {    $<t>$ = 
mknode($<t>1,$<s>2,$<t>3);}| expr '/' error
|               expr '<' expr     {    $<t>$ = 
mknode($<t>1,$<s>2,$<t>3);}| expr '<' error
|               expr '>' expr     {    $<t>$ = 
mknode($<t>1,$<s>2,$<t>3);}| expr '>' error
|               "++" ID     {    scope(temp,$<s>2,charcount);
                                             val *x1 = 
mknode(NULL,$<s>2,NULL);val *x2 = mknode(NULL,"1",NULL);
                                             val *x3 = 
mknode(x1,"+",x2);val *x4 = mknode(NULL,$<s>2,NULL);
                                             $<t>$ = 
mknode(x1,"=",x3);
                                        }
|                 "++" error
|                 "--" ID        {    scope(temp,$<s>2,charcount);
                                             val *x1 = 
mknode(NULL,$<s>2,NULL);val *x2 = mknode(NULL,"1",NULL);
                                             val *x3 = 
mknode(x1,"-",x2);val *x4 = mknode(NULL,$<s>2,NULL);
                                             $<t>$ = 
mknode(x1,"=",x3);
                                        }
|                 "--" error
|               ID "++"          {    scope(temp,$<s>1,charcount);
                                             val *x1 = 
mknode(NULL,$<s>2,NULL);val *x2 = mknode(NULL,"1",NULL);
                                             val *x3 = 
mknode(x1,"+",x2);val *x4 = mknode(NULL,$<s>2,NULL);
                                             $<t>$ = 
mknode(x1,"=",x3);
                                        }
|                 ID "--"        {    scope(temp,$<s>1,charcount);
                                             val *x1 = 
mknode(NULL,$<s>2,NULL);val *x2 = mknode(NULL,"1",NULL);
                                             val *x3 =
mknode(x1,"-",x2);val *x4 = mknode(NULL,$<s>2,NULL);
                                             $<t>$ =
mknode(x1,"=",x3);
                                        }
|                expr "<<" expr {$<t>$ = mknode($<t>1,$<s>2,$<t>3);}|
expr "<<" error
|                expr ">>" expr {$<t>$ = mknode($<t>1,$<s>2,$<t>3);}|
expr ">>" error
|                expr "==" expr {$<t>$ = mknode($<t>1,$<s>2,$<t>3);}|
expr "==" error
|                expr "!=" expr {$<t>$ = mknode($<t>1,$<s>2,$<t>3);}|
expr "!=" error
|                expr ">=" expr {val *x1 =
mknode($<t>1,">",$<t>3);val *x2 = mknode($<t>1,"==",$<t>3);$<t>$ =
mknode(x1,"||",x2);}| expr ">=" error
|                expr "<=" expr {val *x1 =
mknode($<t>1,"<",$<t>3);val *x2 = mknode($<t>1,"==",$<t>3);$<t>$ =
mknode(x1,"||",x2);}| expr "<=" error
|                expr "||" expr {$<t>$ = mknode($<t>1,$<s>2,$<t>3);}|
expr "||" error
|                expr "&&" expr {$<t>$ = mknode($<t>1,$<s>2,$<t>3);}|
expr "&&" error
|                functcall {$<t>$ = mknode(NULL,"_ra",NULL);}
|                '-' expr  %prec '!'     {$<t>1 =
mknode(NULL,"-1",NULL);$<t>$ = mknode($<t>1,"*",$<t>2);}|
       '-' error
|                ID  {scope(temp,$<s>1,charcount);$<t>$ =
mknode(NULL,$<s>1,NULL);}
|                NUM {$<t>$ = mknode(NULL,$<s>1,NULL);}
|                  STR {$<t>$ = mknode(NULL,$<s>1,NULL);}
;
%%
#include"lex.yy.c"
int count=0;
int main(int argc, char *argv[])
{
   yyin = fopen(argv[1], "r");
   filename = argv[1];
   ifstream myfile(argv[1]);
   string line;
   int cc = 0;
   cout << "\n____________________________________________";
   cout << "\n___________S_O_U_R_C_E___C_O_D_E____________\n";
      if (myfile.is_open()) {
      while (getline (myfile, line)) {
           trim(line);
           cout << "|" << ++cc << "|    " << line << endl;
           lines.push_back(line);
      }
      myfile.close();
      }
   cout << "|_|___________________________________________\n\n";
   if(!yyparse()){
      cout << "\nParsing complete\n";
            print_tac();
   }
      else
      cout << "\nParsing failed\n";

      fclose(yyin);
      return 0;
}

void yyerror(char *s) {
      printf("Line %d : %s before '%s'\n", lineno, s, yytext);
}
```

Semantic , Symbol Table & 3 Address

Code

```
#include<bits/stdc++.h>
using namespace std;
extern vector<string>lines,fargs;
extern string filename,fname;
extern string func;
string last;
extern int lineno;
extern int first,lno;
extern int charcount;
extern int fflag,dd;

map<int,vector<int> >mp;
map<int,int>mpf;
map<string,int>mpfuncf;
map<string,vector<int> >mpfunc;
int address=100,addr=100;
template <typename T>
string to_string(T val)
{
     stringstream stream;
     stream << val;
     return stream.str();
}
struct val{
    string data;
    val *left,*right;
};

struct node{
    vector<string>type, datatype, token;
    vector<int>lineno;
    vector<int>dim;
    vector<int>position;
    vector<node*>next;
    node *parent;
    string name;
    int no_of__block_s;
    int first;
    int func_def_flag;
};
node *temp2;
node *head = new node();
node *temp = head;
vector<string>ans;
vector<string>commands;
void bfs(val *x){
    if(x==NULL) return;
    bfs(x->left);
    bfs(x->right);
    cout << x->data;
    ans.push_back(x->data);
}
string arr_op[] =
{"+","-","*","/","||","&&",">=","<=","<<",">>","!=","==",">","<","="}
;
bool is_operator(string a){
    for(int i=0;i<15;++i){
      if(a==arr_op[i])
           return true;
    }
    return false;
}
int j=0;
int ifk=0;
int tac_if(int x,int ifkk=0){
    if(x){
      string label = "_L"+to_string(ifk);
      string cc = "IfZ "+last+" Goto ";
      commands.push_back(cc);++addr;
      mp[ifk].push_back(commands.size()-1);
      ++ifk;
      return (ifk-1);
```

```
        }
        else{
          string label = "_L"+to_string(ifkk);
          string cc =  "Goto "+label+";";
          label = "_L"+to_string(ifkk);
          cc =  label+":";
          for(int i=0;i<mp[ifkk].size();++i){
               commands[mp[ifkk][i]]+=to_string(addr);
          }
          commands.push_back(cc);
          ++addr;
        }
        ++ifk;
    }
    void tac_ret(){
        vector<string>a;

        for(int i=0;i<ans.size();++i){
          a.push_back(ans[i]);
          if(is_operator(a.back())){
               if(a.back()=="="){
                    string op = a.back();a.pop_back();
                    string x = a.back();a.pop_back();
                    string y = a.back();a.pop_back();
                    string cc = y +" "+ op +" "+ x +";";
                    a.push_back(y);
                    commands.push_back(cc);++addr;
                    continue;
               }
               // cout << a.back() << " ";
              string op = a.back();a.pop_back();
               string x = a.back();a.pop_back();
               string y = a.back();a.pop_back();
               if((x[0]=='_' && y[0]!='_')||(x[0]!='_' && y[0]=='_')){
                    if(x[0]!='_'){
                         string t = "_t" + to_string(j);
          //               cout << t << " ";
                         string cc = t + " = " + x +";";
```

```
                        commands.push_back(cc);++addr;
                        x = t;
                        ++j;
                  }
                  if(y[0]!='_'){
                        string t = "_t" + to_string(j);
      //                cout << t << " ";
                        string cc = t + " = " + y +";";
                        commands.push_back(cc);++addr;
                        y = t;
                        ++j;
                  }
            }
            string t = "_t" + to_string(j);
    //            cout << t << " ";
            a.push_back(t);
           string cc = t + " = " + y +" "+ op +" "+ x +";";
            commands.push_back(cc);++addr;
            last = t;
            ++j;
      }
    }
    if(a.size()){
      string cc = a.back();
      cc = "_ra = " + cc +";";
      commands.push_back(cc);++addr;
    }
    ans.clear();
    cout << endl;
}
void tac_call(string s,int x=1,int fl=0){
    if(x){
      string t = "_t" + to_string(j);
      string cc = t + " = " + s + ";";
      commands.push_back(cc);++addr;
      cc = "PushParam "+t+";";
      commands.push_back(cc);++addr;
      ++j;
    }
    else{
      string cc = "LCall " + s+";";
      if(mpfuncf[s])
           cc+=to_string(mpfuncf[s]);
      mpfunc[s].push_back(commands.size());
      if(mpfuncf[s]){
           for(int i=0;i<mpfunc[func].size();++i){
                commands[mpfunc[func][i]]+=to_string(addr)+";";
           }
      }
      commands.push_back(cc);++addr;
      int k1=j-1;
      if(fl){
           for(int i=0;(i<temp2->token.size() &&
temp2->token[i]=="argument");++i){
                cc = "PopParam _t" + to_string(k1)+";";
                commands.push_back(cc);++addr;
                k1--;
           }
      }
    }
}
int tac_while(int x,int val=0){
    if(x==1){
      string label = "_L"+to_string(ifk);
      string cc = label+":";
      mpf[ifk]=addr;
      commands.push_back(cc);++addr;
      ++ifk;
      return (ifk-1);
    }
    else if(x==2){
      string label = "_L"+to_string(ifk)+":";
      string cc = "IfZ "+last+" Goto "+ label +";";
      mp[ifk].push_back(commands.size());
      commands.push_back(cc);++addr;
      ++ifk;
```

```
        }
        else{
          string label = "_L"+to_string(val)+":"+to_string(mpf[val]);
          string cc =  "Goto "+label+";";
          mp[val].push_back(commands.size());
          commands.push_back(cc);++addr;
          label = "_L"+to_string(val+1);
          cc =  label+":";
          for(int i=0;i<mp[val+1].size();++i){
                commands[mp[val+1][i]]+=to_string(addr)+";";
          }
          commands.push_back(cc);++addr;
        }
    }
    void tac_func(string func,int beg=1){
        if(beg){
          string cc = func + ":";
          mpfuncf[func]=addr;
          if(mpfunc[func].size()){
                for(int i=0;i<mpfunc[func].size();++i){
                      commands[mpfunc[func][i]]+=to_string(addr)+";";
                }
          }
          commands.push_back(cc);++addr;
          cc = "beginFunc ;";
          commands.push_back(cc);++addr;
        }
        else{
          string cc = "endFunc ;";
          commands.push_back(cc);++addr;
        }
    }
    void tac_exp(){
        vector<string>a;
        if(ans.size()==1){
                string y = ans[0];
                string t = "_t" + to_string(j);
                string cc = t + " = " + y +" > 0;";
```

```
            commands.push_back(cc);++addr;
            y = t;
            last = t;
            ++j;
            return;
    }
    for(int i=0;i<ans.size();++i){
      a.push_back(ans[i]);
      if(is_operator(a.back())){
            if(a.back()=="="){
                  string op = a.back();a.pop_back();
                  string x = a.back();a.pop_back();
                  string y = a.back();a.pop_back();
                  string cc = y +" "+ op +" "+ x +";";
                  a.push_back(y);
                  commands.push_back(cc);++addr;
                  continue;
            }
            // cout << a.back() << " ";
           string op = a.back();a.pop_back();
            string x = a.back();a.pop_back();
            string y = a.back();a.pop_back();
            if((x[0]=='_' && y[0]!='_')||(x[0]!='_' && y[0]=='_')){
                  if(x[0]!='_'){
                        string t = "_t" + to_string(j);
      //                cout << t << " ";
                        string cc = t + " = " + x +";";
                        commands.push_back(cc);++addr;
                        x = t;
                        ++j;
                  }
                  if(y[0]!='_'){
                        string t = "_t" + to_string(j);
      //                cout << t << " ";
                        string cc = t + " = " + y +";";
                        commands.push_back(cc);++addr;
                        y = t;
                        ++j;
```

```
                 }
            }
            string t = "_t" + to_string(j);
                 a.push_back(t);
           string cc = t + " = " + y +" "+ op +" "+ x +";";
            commands.push_back(cc);++addr;
            last = t;
            ++j;
      }
    }
    ans.clear();
    cout << endl;
}
void print_tact(){
    for(int i=0;i<commands.size();++i){
      reverse(commands[i].begin(),commands[i].end());
      char tt1 = *commands[i].begin();
      reverse(commands[i].begin(),commands[i].end());
      if(tt1==':'){

      }
    }
}
void print_tac(){
    cout << "\n___________________________________________";
    cout << "\n_____T_H_R_E_E__A_D_D_R_E_S_S__C_O_D_E_____\n";
    for(int i=0;i<commands.size();++i){
      reverse(commands[i].begin(),commands[i].end());
      char tt1 = *commands[i].begin();
      reverse(commands[i].begin(),commands[i].end());
      if(tt1==':')
            cout << address << ") " << commands[i] << endl;
      else{
            cout << "\t" << address << ") " << commands[i] << endl;
      }
      ++address;
    }
    cout << "_____________________________________________\n\n";
```

```
}
val *mknode(val *l,string parent,val *r){
    val *p = new val();
    p->data = parent;
    p->left = l;
    p->right = r;
    return p;
}
void checkReturn(int flag){
    int i=0;
    while(head->next[i]){
      if(head->next[i]->name==temp->name)
           break;
      ++i;
    }
    if(head->datatype[i]=="void"){
      if(flag) return;
      cout << filename << ":" << lineno  << ":" << charcount << ":"
<< "error: return type mismatch\n";
    }
    else{
      if(!flag) return;
      cout << filename << ":" << lineno  << ":" << charcount << ":"
<< "error: return type mismatch\n";
    }
}
void checkDim(){
    for(int i=0;i<fargs.size();++i){
      if(fargs[i][0]=='-'){
           cout << filename << ":" << lineno  << ":" << charcount <<
":" << "error: size of array is negative \n";
           cout << lines[lineno-1];
      }
    }
}
void prtformat(node *temp,int level){
      for(int i=0;i<temp->token.size();++i){
      string fff="-";
```

```
        if(temp->token[i]=="function")
             fff = "1";
        if(temp->token[i]=="_block_")
             fff = "null";
        if(temp->token[i]=="_block_")
             cout <<"\t" << temp->token[i] << "\t\t" << temp->type[i]
<< "\t\t" <<  temp->datatype[i] << "\t\t" << temp->dim[i]  << "\t\t"
<< fff << "\t\t" << level << endl;
        else
             cout <<"\t" << temp->token[i] << "\t" << temp->type[i] <<
"\t\t" <<  temp->datatype[i] << "\t\t" << temp->dim[i]  << "\t\t" <<
fff << "\t\t" << level << endl;

      }
    int j=0;
    for(int i=0;i<temp->token.size();++i){
      if(temp->token[i]=="function" || temp->token[i]=="_block_"){
             prtformat(temp->next[j++],level+1);
      }
     }
}
void printformat(node *temp){
    cout << "\n\n\t\tGLOBAL\n\n";
    cout <<
"\t---------------------------------------------------------------------
------------------------\n";
      cout << "\tType\t         Name        Datatype
Dimension    flag     nesting level\n";
      cout <<
"\t---------------------------------------------------------------------
------------------------\n";
    prtformat(temp,0);
}

void prt(node *temp){
    cout << "\t---------------------------------------------\n";
      cout << "\tToken\t        Type   Datatype   line\n";
      cout << "\t---------------------------------------------\n";
      for(int i=0;i<temp->token.size();++i){
       cout << "\t" << temp->token[i] << "\t" << temp->type[i] <<
"\t" << setw(5) << temp->datatype[i] << "\t" << setw(5) <<
temp->lineno[i] << endl;
      }
      cout << "\t---------------------------------------------\n";

    int j=0;
    for(int i=0;i<temp->token.size();++i){
      if(temp->token[i]=="function" || temp->token[i]=="_block_"){
           cout << "\n\n\t\t" << temp->type[i]<< "\n\n";
           prt(temp->next[j++]);
      }
     }
}
void print(node *temp){
    cout << "\n\n\t\tGLOBAL\n\n";
    prt(temp);
}
int paramcheck(){
    int k=0;
    temp2 = head;
    for(int i=0;i<head->token.size();++i){
      if(head->token[i]=="function"){
           temp2 = head->next[k++];
           if(head->type[i]==fname){
                break;
           }
      }
    }
    --k;
    if(!temp){
      cout << filename << ":" << lineno  << ":" << charcount << ":"
<< "error: undefined reference to '"<< fname << "'\n";return 0;
    }
    int i;

    for(i=0;i<temp2->token.size();++i){
       if(temp2->token[i]=="argument"){
             cout << temp2->type[i] <<"arg\n";
             if(i==fargs.size()){
                   cout << filename << ":" << lineno  << ":" <<
charcount << ":" << "error: too few arguments to function '"<< fname
<< "'\n " << lines[lineno-1] << endl;
                   fargs.clear();
                   return 0;
             }
       }
       else
             break;
    }
    if(i==fargs.size()){    fargs.clear();return 1;}

    cout << filename << ":" << lineno  << ":" << charcount << ":" <<
"error: too many arguments to function '"<< fname << "'\n " <<
lines[lineno-1] << endl;
    fargs.clear();
    return 0;
}
void placePoint(int position){
    for(int i=1;i<position;++i){
       cout << " ";
    }
    cout << "^" << endl;
}

bool search(node *temp, string token, string type, string
datatype,int position){
    int flag=0;
    for(int i=0;i<temp->token.size();++i){
       if(fflag && temp->type[i]==type && temp->token[i]==token){
             cout << filename << "::" << lineno  << ":" <<
(position-1) << ":" << "error: conflicting types for '"<<
temp->type[i] << "'\n " << lines[lno-1] << endl;
             cout << filename << "::" << temp->lineno[i]  << ":" <<
temp->position[i] << ":" << "note: previous definition of '" <<
```

```
temp->type[i] << "' was here\n " << lines[i+1] << endl;
            flag=1;
            return true;
        }
        if(temp->type[i]==type && temp->token[i]==token &&
temp->datatype[i]!=datatype){
            cout << filename << ":" << lineno  << ":" << (position-1)
<< ":" << "error: conflicting types for '"<< temp->type[i] << "'\n "
<< lines[lineno-1] << endl;
            cout << filename << ":" << (i+1)  << ":" <<
temp->position[i] << ":" << "note: previous definition of '" <<
temp->type[i] << "' was here\n " << lines[i+2] << endl;
            flag=1;
        }
        if(temp->type[i]==type && temp->token[i]==token &&
temp->datatype[i]==datatype){
            if(token=="function"){
                int k=0;
                node *temp2 = head->next[k++];
                while(temp2->name!=type){temp2=head->next[k++];}
                if(temp2->func_def_flag==1){
                    ::temp = temp2;
                }
                return true;
            }
            cout << filename << ":" << lineno  << ":" << (position-1)
<< ":" << "error: redeclaration of '"<< temp->type[i] << "' with no
linkage\n " << lines[lineno-1] << endl;
            cout << filename << ":" << (i+1)  << ":" <<
temp->position[i] << ":" << "note: previous declaration of '" <<
temp->type[i] << "' was here\n " << lines[i+2] << endl;
            flag=1;
        }
        if(temp->type[i]==type && temp->token[i]!=token){
            cout << filename << ":" << lineno << ":" << (position-1)
<< ":" << "error: '" << temp->type[i] << "' redeclared as different
kind of symbol\n " << lines[lineno-1] << endl;
        //    cout << filename << ":" << (i+1) << ":" <<
```

```
temp->position[i] << ":" << "note: previous definition of '"<<
temp->type[i] << "' was here\n " << lines[i+2] << endl;
            flag=1;
      }
      if(flag) return true;
    }
    return false;
}
void insVar(string token, string datatype, string type, int lineno,
int position, bool isfunc=false){
    if(isfunc==true){
      if(token=="_block_"){
            stringstream ss;
            temp->no_of__block_s++;
            ss << temp->no_of__block_s;
            string str = ss.str();
            node *ptr = new node();
            temp->next.push_back(new node());
            temp->token.push_back(token);
            temp->dim.push_back(-1);
            temp->lineno.push_back(lineno);
            temp->type.push_back(temp->name + "." + str);
            temp->position.push_back(position);
            temp->datatype.push_back(datatype);
            temp->next.back()->parent = temp;
            temp = temp->next.back();
            temp->no_of__block_s=0;
            temp->name = temp->parent->name + "." + str;
            temp->first=2;
            return;
      }
      if(search(head,token,type,datatype,position))
            return;

      node *ptr = new node();
      head->next.push_back(new node());
      head->token.push_back(token);
      temp->dim.push_back(-1);
```

```
        head->lineno.push_back(lineno);
        head->type.push_back(type);
        head->no_of__block_s=0;
        head->position.push_back(position);
        head->datatype.push_back(datatype);
        head->next.back()->parent = head;
        temp = head->next.back();
        temp->name = type;
        temp->first=1;
      }
      else{
        if(search(temp,token,type,datatype,position))
             return;
        temp->token.push_back(token);
        temp->type.push_back(type);
        temp->dim.push_back(dd);
        temp->lineno.push_back(lineno);
        temp->position.push_back(position);
        temp->datatype.push_back(datatype);
      }
}
void scope(node *temp,string type, int position){
      if(temp==NULL){
        if(func!=""){
             cout << filename << ": In function '" << func << "' :\n";
             cout << filename << ":" << lineno  << ":" << (position-1)
<< ":" << "error: '" << type << "' undeclared (first use of this
function)\n " << lines[lineno-1] << endl;
             placePoint(position);
             cout << filename << ":" << lineno  << ":" << (position-1)
<< ":" << "note: each undeclared identifier is reported only once for
each function it appears in\n";
             insVar("  NULL","NULL",type,lineno,position);
        }
        else{
             cout << filename << ":" << lineno  << ":" << (position-1)
<< ":" << "warning: data definition has no type or storage class \n "
<< lines[lineno-1] << endl;
```

```
            placePoint(position);
            insVar("  NULL","NULL",type,lineno,position);
        }
        return;
    }
    for(int i=0;i<head->token.size();++i){
        if(head->type[i]==type && head->token[i]=="variable"){
            return;
        }
        if(head->type[i]==type && head->token[i]=="argument"){
            return;
        }
    }
    for(int i=0;i<temp->token.size();++i){
        if(temp->type[i]==type && temp->token[i]=="variable"){
            return;
        }
        if(temp->type[i]==type && temp->token[i]=="argument"){
            return;
        }
    }
    scope(temp->parent,type,position);
}
```

Implementation Details

We have used structure data structure to build ICG phase. First we had to modify the parser code to insert the values in the symbol table. The structure contains all the information related to the function it contains like line no, datatype, token, position, its parent pointer.

```
struct node{
    vector<string>type, datatype, token;
    vector<int>lineno;
    vector<int>dim;
    vector<int>position;
    vector<node*>next;
    node *parent;
    string name;
    int no_of__block_s;
    int first;
    int func_def_flag;
};
```

- The print format function is there to print the values which are inside the symbol table. There is separate symbol tables for separate functions.
- The symbol table contains all the information related to size , datatype , array dimension if used,etc.
- The tac_if function takes care of the if statement and checks the condition goes to the next statement after this execution.
- Similarly for return statement there is a function which generates the three address code for return statement. The tac_func function is called whenever there is a function call.
- The tac_exp converts all the infix expression to postfix then it evaluates the value of the expression, stack is used to evaluate this.
- Backpaching is also taken care.
- Back patching usually refers to the process of resolving forward branches that have been planted in the code, e.g. at 'if' statements, when the value of the target

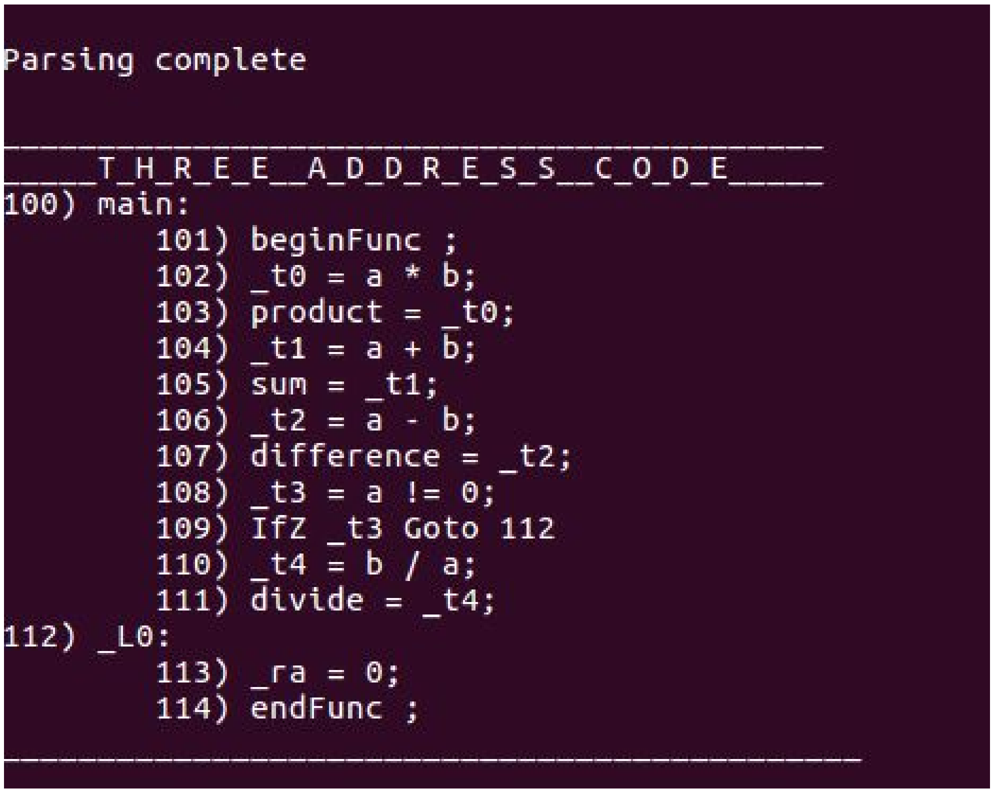
becomes known, e.g. when the closing brace or matching 'else' is encountered.

Test Cases

Test case #1

```
#include<stdio.h>
int main()
{
    int a;
    int b;

    int product=a*b;
    int sum=a+b;
    int difference = a-b;

    if(a!=0)
    {
      double divide = b/a;
    }

    return 0;
}
```

Output :

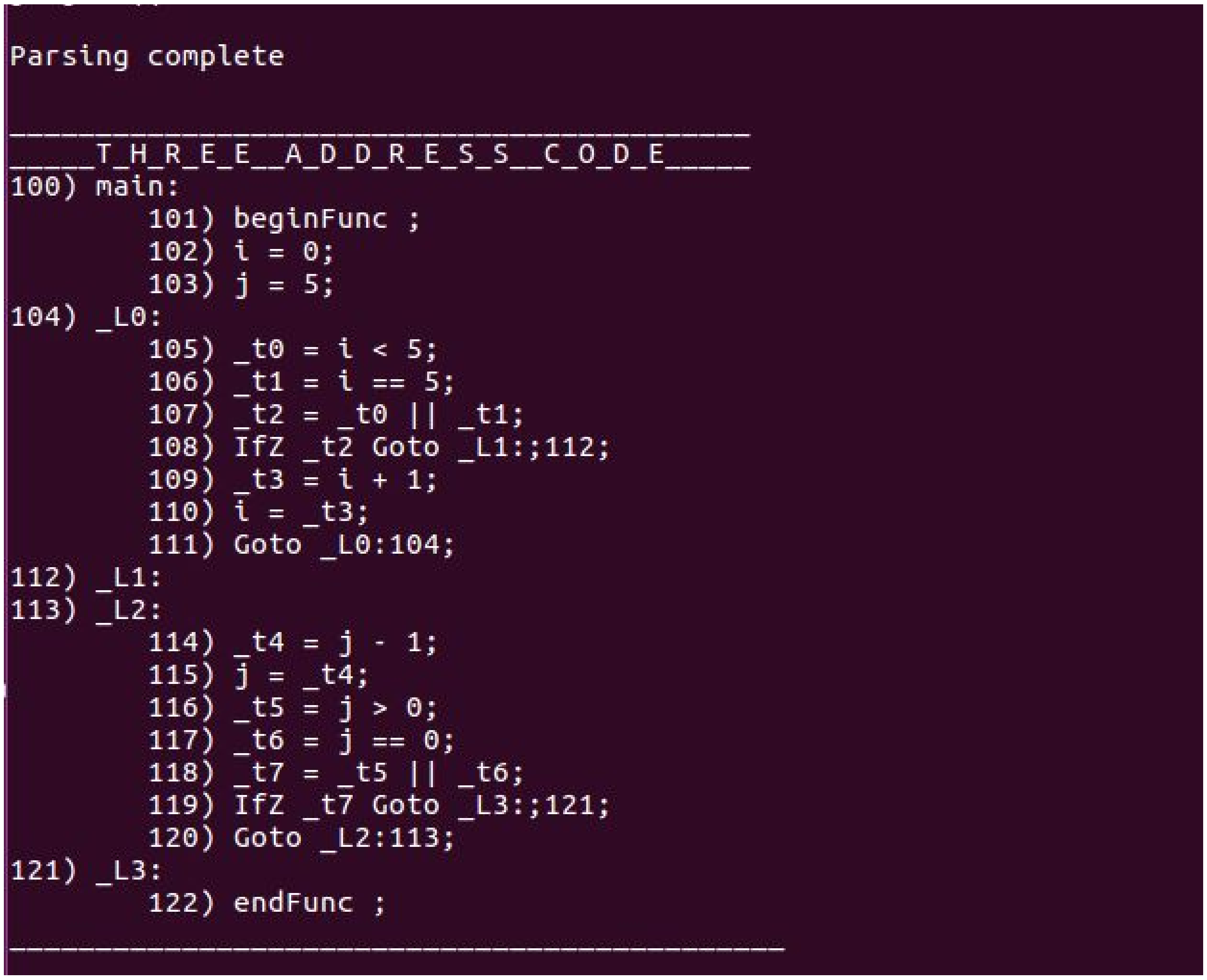


```
Parsing complete

______________________________________________
_____T_H_R_E_E__A_D_D_R_E_S_S__C_O_D_E_____
100) main:
        101) beginFunc ;
        102) _t0 = a * b;
        103) product = _t0;
        104) _t1 = a + b;
        105) sum = _t1;
        106) _t2 = a - b;
        107) difference = _t2;
        108) _t3 = a != 0;
        109) IfZ _t3 Goto 112
        110) _t4 = b / a;
        111) divide = _t4;
112) _L0:
        113) _ra = 0;
        114) endFunc ;
________________________________________________
```

Test case #2

```
#include<stdio.h>
int main()
{
    long long int i=0,j=5;
    while (i <= 5)
    {
      i = i + 1;
    }
    do{
      j = j - 1;
    }while (j >= 0);
}
```

Output

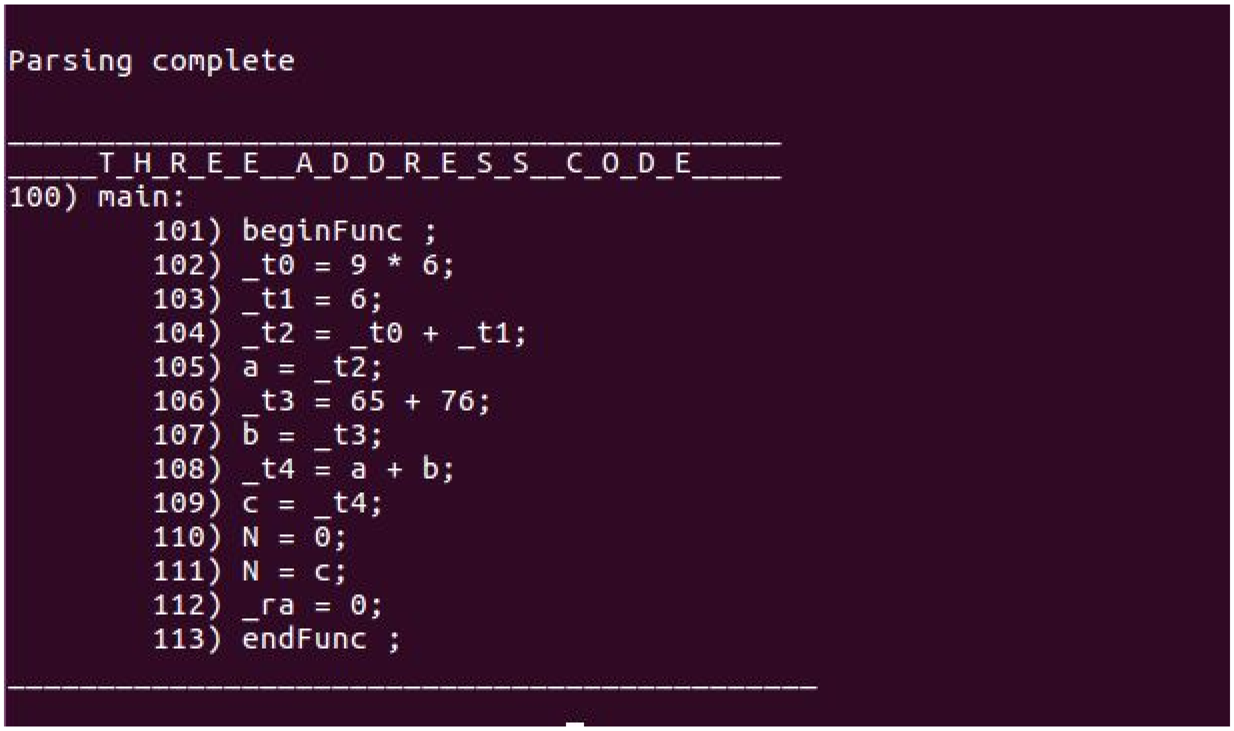


```
Parsing complete

_________________________________________
_____T_H_R_E_E__A_D_D_R_E_S_S__C_O_D_E_____
100) main:
        101) beginFunc ;
        102) i = 0;
        103) j = 5;
104) _L0:
        105) _t0 = i < 5;
        106) _t1 = i == 5;
        107) _t2 = _t0 || _t1;
        108) IfZ _t2 Goto _L1:;112;
        109) _t3 = i + 1;
        110) i = _t3;
        111) Goto _L0:104;
112) _L1:
113) _L2:
        114) _t4 = j - 1;
        115) j = _t4;
        116) _t5 = j > 0;
        117) _t6 = j == 0;
        118) _t7 = _t5 || _t6;
        119) IfZ _t7 Goto _L3:;121;
        120) Goto _L2:113;
121) _L3:
        122) endFunc ;

___________________________________________
```

Test case #3

```
#include<stdio.h>

int main()
{
    int N,i,j,k;

    int a=9*6+6;
    int b=65+76;
    int c=a+b;

    N=0;
     N=c;
 return 0;
}
```

Output



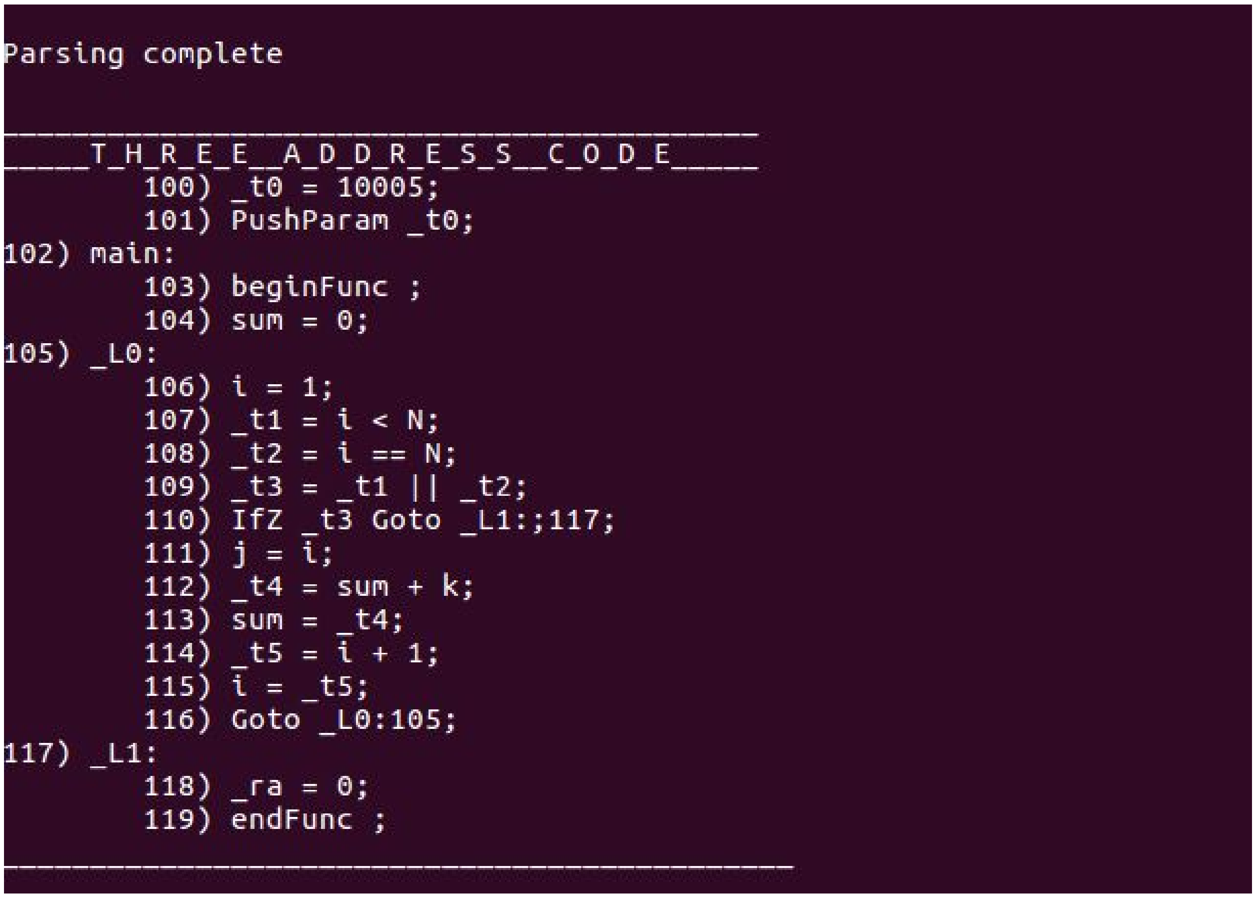
```
Parsing complete

_______________________________________________
_____T_H_R_E_E__A_D_D_R_E_S_S__C_O_D_E_____
100) main:
        101) beginFunc ;
        102) _t0 = 9 * 6;
        103) _t1 = 6;
        104) _t2 = _t0 + _t1;
        105) a = _t2;
        106) _t3 = 65 + 76;
        107) b = _t3;
        108) _t4 = a + b;
        109) c = _t4;
        110) N = 0;
        111) N = c;
        112) _ra = 0;
        113) endFunc ;

_____________________________________________
```

Test case #4

```
#include<stdio.h>
int A[10005];

int main()
{
    int N,i,j,k;
    int sum=0;
    for(i=1;i<=N;++i)
    {
    j=i;
    sum+=k;
    }
```

```
     return 0;
 }
```

Output

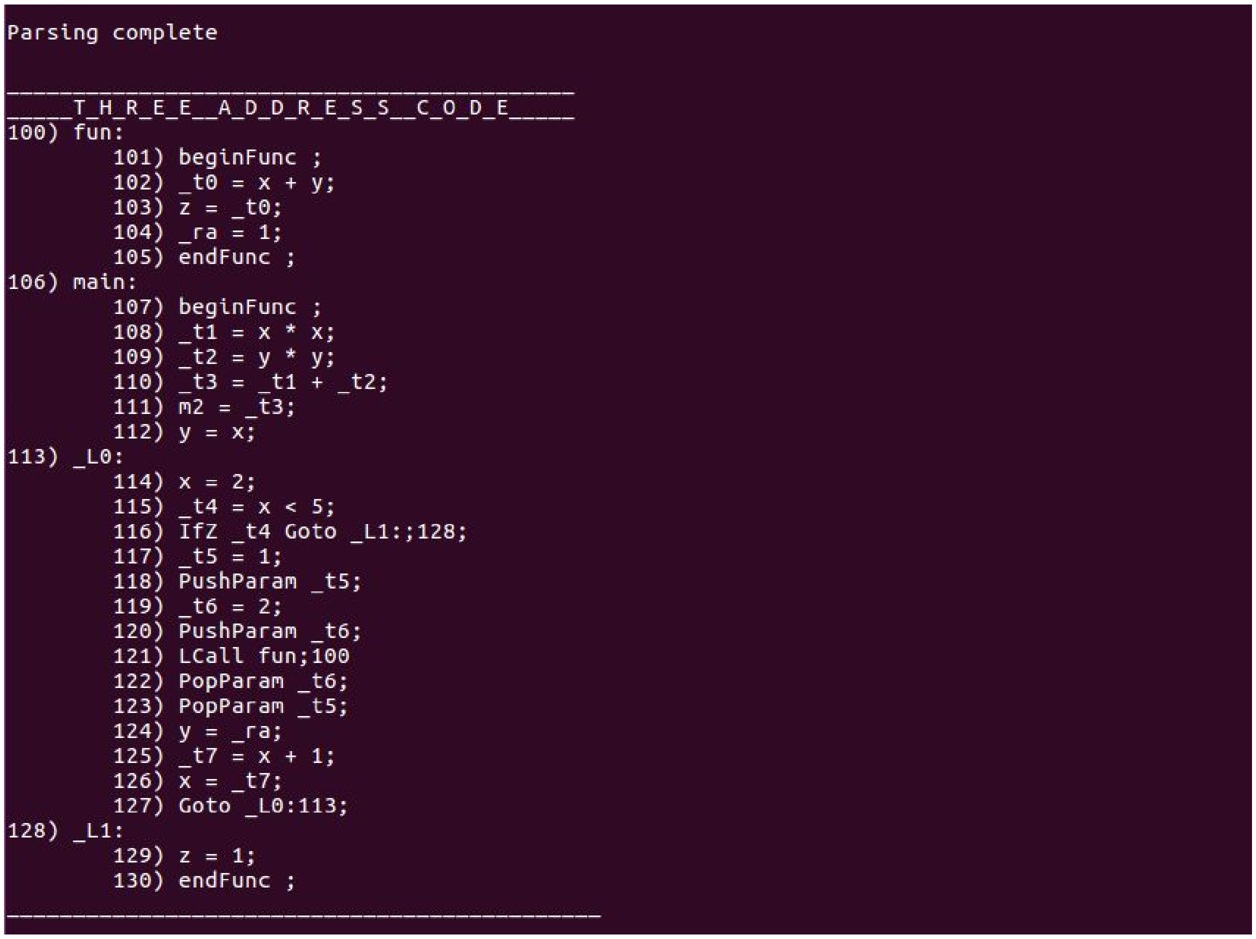


```
Parsing complete

_______________________________________
_____T_H_R_E_E__A_D_D_R_E_S_S__C_O_D_E_____
        100) _t0 = 10005;
        101) PushParam _t0;
102) main:
        103) beginFunc ;
        104) sum = 0;
105) _L0:
        106) i = 1;
        107) _t1 = i < N;
        108) _t2 = i == N;
        109) _t3 = _t1 || _t2;
        110) IfZ _t3 Goto _L1:;117;
        111) j = i;
        112) _t4 = sum + k;
        113) sum = _t4;
        114) _t5 = i + 1;
        115) i = _t5;
        116) Goto _L0:105;
117) _L1:
        118) _ra = 0;
        119) endFunc ;

_____________________________________________
```

Test case #5

```
 int fun(int x,int y){
        int z = x+y;
        return 1;
}
int main(){
    int x, y;
        int m2 = x * x + y * y;
    y = x;
    for(x=2;x<5;++x){
        y=fun(1,2);
    }
    int z = 1;
}
```

Output :



```
Parsing complete

___________________________________________
_____T_H_R_E_E__A_D_D_R_E_S_S__C_O_D_E_____
100) fun:
        101) beginFunc ;
        102) _t0 = x + y;
        103) z = _t0;
        104) _ra = 1;
        105) endFunc ;
106) main:
        107) beginFunc ;
        108) _t1 = x * x;
        109) _t2 = y * y;
        110) _t3 = _t1 + _t2;
        111) m2 = _t3;
        112) y = x;
113) _L0:
        114) x = 2;
        115) _t4 = x < 5;
        116) IfZ _t4 Goto _L1:;128;
        117) _t5 = 1;
        118) PushParam _t5;
        119) _t6 = 2;
        120) PushParam _t6;
        121) LCall fun;100
        122) PopParam _t6;
        123) PopParam _t5;
        124) y = _ra;
        125) _t7 = x + 1;
        126) x = _t7;
        127) Goto _L0:113;
128) _L1:
        129) z = 1;
        130) endFunc ;
_____________________________________________
```

Conclusion

The Intermediate phase has handled all the cases that are present in the course plan. The following tasks are performed in ICG(intermediate Code Generator) phase backpaching, expression reduction, etc. We have not made any changes in the Lex code but changes have been there in the yacc or syntax part. We have used attribute grammar to the grammar to provide context sensitive information. Each attribute has well-defined domain of values, such as integer, float, character, string, and expressions. We have added the semantic code in the symbol table code itself. The symbol table shows all the information about the function like line number , dimension of array , all variables , data types used , etc. The ICG phase takes a base address of 100 and it begins to convert the source code into 3 address code. In this Compiler Project we have successfully implemented all the four phases of the compiler and code is given in this Report.