RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Assignment

Course Code	**19CSC305A**
Course Name	**Compilers**
Programme	**B.Tech**
Department	**Computer Science and Engineering**
Faculty	**Engineering and Technology**

Name of the Student	**Deepak R**
Reg. No.	**18ETCS002041**
Semester/Year	**5th/2020**
Course Leader(s)	**Mr. Hari Krishna S.**

Declaration Sheet				
Student Name	**Deepak R**			
Reg. No	**18ETCS002041**			
Programme	**B.Tech**	Semester/Year	**5th/2020**	
Course Code	**19CSC305A**			
Course Title	**Compilers**			
Course Date		to		
Course Leader	**Mr. Hari Krishna S.**			

Declaration

The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.

Signature of the Student		Date	
Submission date stamp (by Examination & Assessment Section)			

Signature of the Course Leader and date	Signature of the Reviewer and date

Faculty of Engineering and Technology			
Ramaiah University of Applied Sciences			
Department	**Computer Science and Engineering**	**Programme**	**B. Tech in Computer Science and Engineering**
Semester/Batch	**05th /2018**		
Course Code	**19CSC305A**	**Course Title**	**Compilers**
Course Leader	**Mr. Hari Krishna S. M. & Ms. Suvidha**		

Assignment			
Register No.	**18ETCS002041**	**Name of the Stud ent**	**Deepak R**

Sections		Marking Scheme	Marks		
			Max Marks	First Examiner Moderator Marks	
Part A 1					
	A 1.1	Identification and grouping of Tokens	05		
	A 1.2	Implementation in *Lex*	03		
	A 1.3	Design of Context Free Grammar	05		
	A 1.4	Implementation in *Yacc*	07		
	A 1.5	Results and Comments	05		
		Part-A 1 Max Marks	25		
		Total Assignment Marks	25		

Course Marks Tabulation				
Component- CET B Assignment	**First Examiner**	**Remarks**	**Second Examiner**	**Remarks**
A.1				
Marks (out of 25)				
Signature of First Examiner				**Signature of Moderator**

Please note:

1. **Documental evidence for all the components/parts of the assessment such as the reports, photographs, laboratory exam / tool tests are required to be attached to the assignment report in a proper order.**
2. **The First Examiner is required to mark the comments in RED ink and the Second Examiner's comments should be in GREEN ink.**
3. **The marks for all the questions of the assignment have to be written only in the Component – CET B: Assignment table.**
4. **If the variation between the marks awarded by the first examiner and the second examiner lies within +/- 3 marks, then the marks allotted by the first examiner is considered to be final. If the variation is more than +/- 3 marks then both the examiners should resolve the issue in consultation with the Chairman BoE.**

Assignment

Instructions to students:

1. **The assignment consists of 1 questions: Part A – 1 Question.**
2. **Maximum marks is 25.**
3. **The assignment has to be neatly word processed as per the prescribed format.**
4. **The maximum number of pages should be restricted to 15.**
5. **The printed assignment must be submitted to the course leader.**
6. **Submission Date: 16th Jan 2021**
7. **Submission after the due date is not permitted.**
8. **IMPORTANT: It is essential that all the sources used in preparation of the assignment must be suitably referenced in the text.**
9. **Marks will be awarded only to the sections and subsections clearly indicated as per the problem statement/exercise/question**

Contents

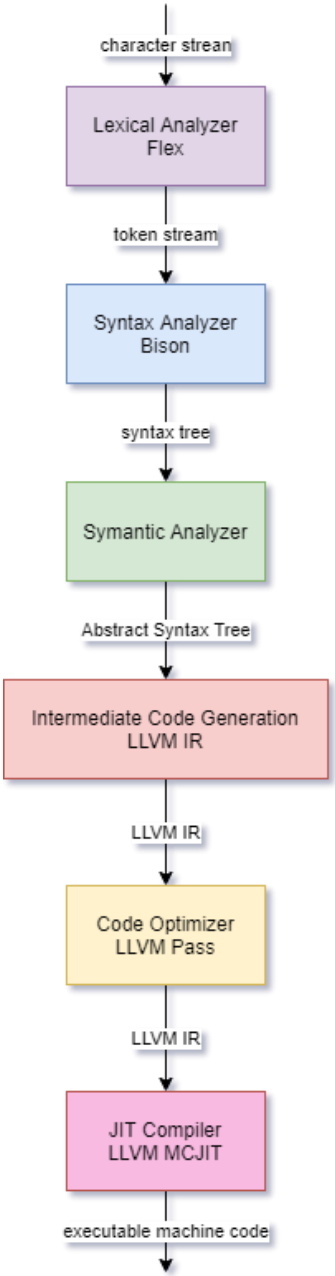
List Of Figures

1 Question A

Solution to Question A

1.1 Introduction

Programming languages are notations for describing computations to people and to machines. The world as we know it depends on programming languages, because all the software running on all the computers was written in some programming language. But, before a program can be run, it must be translated into a form in which it can be executed by a computer. The software systems that do this translation are called compilers. [5]



The assignment is to build such a compiler, to do this we use several tools as shown below,

Figure 1-1 Compiler Recipe

Flex: Flex is a tool for generating scanners, programs which recognized lexical patterns in text. flex reads the given input files, or its standard input if no file names are given, for a description of a scanner to generate.

Bison: Bison is a general-purpose parser generator that converts an annotated context-free grammar into a deterministic LR or generalized LR (GLR) parser employing LALR(1), IELR(1) or canonical LR(1) parser tables.

LLVM: In an LLVM-based compiler, a front end is responsible for parsing, validating and diagnosing errors in the input code, then translating the parsed code into LLVM IR (usually, but not always, by building an AST and then converting the AST to LLVM IR). This IR is optionally fed through a series of analysis and optimization passes which improve the code, then is sent into a code generator to produce native machine code, as shown in Figure 1-2 LLVM Optimizer. This is a very straightforward implementation of the three-phase design, but this simple description glosses over some of the power and flexibility that the LLVM architecture derives from LLVM IR. [1]

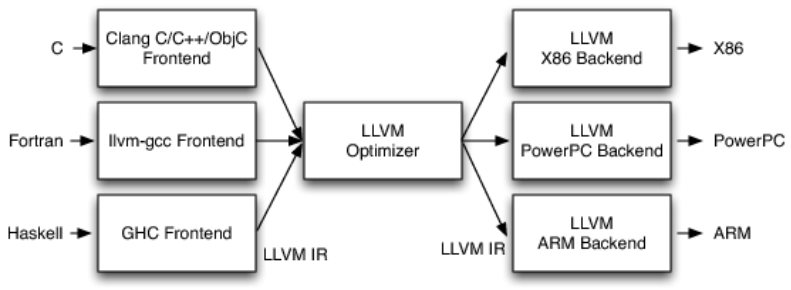


Figure 1-2 LLVM Optimizer

1.2 Identification and grouping of Tokens

1.2.1 Keywords

TOKEN	FOR	IN	RANGE	IF	ELSE
RE	for	in	range	if	else

1.2.2 Operators

Arithmetic Operators

TOKEN	PLUS	MINUS	MUL	DIV	ASSIGN
RE	+	-	*	/	=

Comparison Operators

TOKEN	GRT	GRTEQ	LES	LESEQ	NOTEQ	EQUAL
RE	>	>=	<	<=	!=	==

Boolean Operators

TOKEN	AND	OR	NOT
RE	and	or	not

1.2.3 Special Symbols

TOKEN	LPAREN	RPAREN	LBRACE	RBRACE	LBRACKET	RBRACKET	COMMA
RE	(	)	{	}	[	]	,

TOKEN	EOF
RE	<<eof>>

1.2.4 Literals

TOKEN	DECIMAL	FRACTION	STRING
RE	-?[0-9]+	-?[0-9]+\.[0-9]*	\'([^\\\"]|\\.)*\'

Note: The RE for STRING defined here does not include the newlines and other escape sequences, a DFA was created in lex for doing so, refer to implementation in lex.

1.2.5 Identifier

TOKEN	IDENTIFIER
RE	[a-zA-Z_][a-zA-Z_0-9]*

1.3 Implementation in Lex

tokens.l

```
%{
   #include <string>
   #include <cerrno>
   #include <climits>
   #include <cstdlib>
   #include <cstring> // strerror

   #include "driver/driver.hpp"
   #include "parser.hpp"
   #include "ast/ast_structures.hpp"

   // temporary for storing the string literal
   std::string g_str;

%}

%option noyywrap nounput noinput batch

%x str
%s normal

%{
   yy::parser::symbol_type make_DECIMAL(const std::string& s, const yy::parser::location_type& loc);
   yy::parser::symbol_type make_FRACTION(const std::string& s, const yy::parser::location_type& loc);
   yy::parser::symbol_type make_IDENT(const std::string& s, const yy::parser::location_type& loc);
%}

ident      [a-zA-Z_][a-zA-Z_0-9]*
num        [0-9]
```

```
blank      [ \t\r]

%{
   // runs each time a pattern is matched
   #define YY_USER_ACTION  loc.columns(yyleng);
%}

%%

%{
   yy::location& loc = drv.location;
   loc.step();
%}
//<- one leading blank space; comments should start one blank space after
/* state automata for string literal */

'               { g_str = ""; BEGIN(str); } /*eat '*/
<str>\'           { BEGIN(normal); return yy::parser::make_STRINGLIT(std::make_unique<stringlit>(g_str, loc), loc); }
<str>\\n          g_str += "\n";
<str>\\t          g_str += "\t";
<str>\\r          g_str += "\r";
<str>\\\'         g_str += "'";
<str>\\(.|\n)     g_str += yytext[1];
<str>[^\\']+      g_str += std::string(yytext);

{blank}+          { loc.step(); }
\n+               { loc.lines(yyleng); loc.step(); }
"and"             { return yy::parser::make_AND(loc); }
"or"              { return yy::parser::make_OR(loc); }
"not"             { return yy::parser::make_NOT(loc); }
"if"              { return yy::parser::make_IF(loc); }
"else"            { return yy::parser::make_ELSE(loc); }
"for"             { return yy::parser::make_FOR(loc); }
"in"              { return yy::parser::make_IN(loc); }
"range"           { return yy::parser::make_RANGE(loc); }
"+"               { return yy::parser::make_PLUS(loc); }
"-"               { return yy::parser::make_MINUS(loc); }
"*"               { return yy::parser::make_MUL(loc); }
"/"               { return yy::parser::make_DIV(loc); }
"="               { return yy::parser::make_ASSIGN(loc); }
">"               { return yy::parser::make_GRT(loc); }
">="              { return yy::parser::make_GRTEQ(loc); }
"<"               { return yy::parser::make_LES(loc); }
"<="              { return yy::parser::make_LESEQ(loc); }
"!="              { return yy::parser::make_NOTEQ(loc); }
"=="              { return yy::parser::make_EQUAL(loc); }
{num}+\.{num}*    { return make_FRACTION(yytext, loc); }
-?{num}+          { return make_DECIMAL(yytext, loc); }
{ident}           { return make_IDENT(yytext, loc); }
"("               { return yy::parser::make_LPAREN(loc); }
")"               { return yy::parser::make_RPAREN(loc); }
"{"               { return yy::parser::make_LBRACE(loc); }
"}"               { return yy::parser::make_RBRACE(loc); }
"["               { return yy::parser::make_LBRACKET(loc); }
```

```
"]"             { return yy::parser::make_RBRACKET(loc); }
","             { return yy::parser::make_COMMA(loc); }
#.*             /* eat everything; single line comment */
.               { throw yy::parser::syntax_error
                  (loc, "invalid character: " + std::string(yytext));
                }
<<EOF>>         { return yy::parser::make_END(loc); }

%%

yy::parser::symbol_type make_DECIMAL(const std::string& s, const yy::parser::location_type& loc) {
 std::unique_ptr<decimal> temp = std::make_unique<decimal>(std::strtoll(yytext, NULL, 10), loc);
 return yy::parser::make_DECIMAL(std::move(temp), loc);
}

yy::parser::symbol_type make_FRACTION(const std::string& s, const yy::parser::location_type& loc) {
 std::unique_ptr<fraction> temp = std::make_unique<fraction>(std::strtold(yytext, NULL), loc);
 return yy::parser::make_FRACTION(std::move(temp), loc);
}

yy::parser::symbol_type make_IDENT(const std::string& s, const yy::parser::location_type& loc) {
 std::unique_ptr<identifier> temp = std::make_unique<identifier>(s, loc);
 return yy::parser::make_IDENT(std::move(temp), loc);
}

// code from bison manual: https://www.gnu.org/software/bison/manual/html_node/Calc_002b_002b-
Scanner.html

void driver::scan_begin() {
 if (file.empty() || file == "stdin")
   yyin = stdin;
 else if (!(yyin = fopen(file.c_str(), "r"))) {
    std::cerr << "cannot open " << file << ": " << strerror(errno) << '\n';
    exit (EXIT_FAILURE);
   }
}

void driver::scan_end() {
 fclose(yyin);
}
```

driver.hpp

```
#pragma once

#include <map>
#include <string>
#include "parser.hpp"

// declare the YY_DECL as our custom parser driver
#define YY_DECL yy::parser::symbol_type yylex(driver& drv)

YY_DECL;

class driver {
```

```
  public:
   driver();

   std::map<std::string, int> variables;

   int result;

   // to run the parser on a given file
   int parse(const std::string& f);

   // name of the file being parsed
   std::string file;

   // handling the scanner
   // NOTE: defined in tokens.l
   void scan_begin();
   void scan_end();

   // token location
   yy::location location;
};
```

driver.cpp

```
#include "driver.hpp"

#include "parser.hpp"

driver::driver() { }

int driver::parse(const std::string& f) {
   file = f;
   location.initialize(&file);

   // scan_begin and scan_end are defined in tokens.l
   scan_begin();

   yy::parser parse(*this);

   // int res =
   parse();

   scan_end();

   // return res;
   return 0;
}
```

Note: For the header files and other sources, please refer to Appendix B

1.4 Design of Context Free Grammar

```
program    : stmts

stmts      : stmt
           | stmts stmt

stmt       : expr
           | var_decl
           | conditional
           | for_loop
           | for_range

for_loop   : "for" "(" expr "," expr "," expr ")" block

for_range  : "for" identifier "in" "range" "decimal" block

block      : "{" stmts "}"

conditional    : "if" expr block "else" block
               | "if" expr block

var_decl   : "identifier" "identifier"
           | "identifier" "identifier" "=" expr

literals   : "decimal"
           | "fraction"
           | "stringlit"

expr       : identifier "=" expr
           | identifier "(" call_args ")"
           | identifier
           | literals
           | binop_expr
           | unaryop_expr
           | compare_expr
           | array_access
           | "(" expr ")"

call_args  : /*blank*/
           | expr
           | call_args[arg] "," expr

array_access   : identifier "[" expr "]"
               | array_access "[" expr "]"

binop_expr  : expr "and" expr
           | expr "or" expr
           | expr "+" expr
           | expr "-" expr
           | expr "*" expr
           | expr "/" expr
```

```
compare_expr   :  expr ">" expr
           |  expr ">=" expr
           |  expr "<" expr
           |  expr "<=" expr
           |  expr "==" expr
           |  expr "!=" expr

unaryop_expr   : "not" expr
```

Minimum two data types:

- decimal
- fraction

Minimum two control statements:

- if
- else

Minimum two looping statements:

- for
- for i in range

Input-output functions:

- display
- read

Compound statements and two-dimensional Array:

- { block }
- array[idx]
- array[idx][jdx]
- array[idx][jdx][kdx]

1.5 Implementation in Yacc

```
%skeleton "lalr1.cc"
%require "3.5"
%language "c++"

%defines

// variant will make sure we can use our non-trivial types
%define api.value.type variant
```

```
%define api.token.constructor
%define parse.assert

// this will be added to the parser.cpp file, cyclic-dependecy is resolved by using
// forward declaration of the driver class, this is added verbatim
// if you want to declare any variables do not do in this requires section
%code requires {
   #include <string>
   #include <memory>
   #include <typeinfo>

   class driver;

   #include "ast/ast_structures.hpp"
   // love you c++ gods, g++ gave me much help in debugging
   // <3
   #include "visitor/visitor.hpp"
   #include "visitor/visitor_pprint.hpp"
   #include "external/loguru.hpp"

   static int cnt = 0;
}

// parsing context
%param { driver& drv }

// for location tracking
%locations
%verbose

// because we'll be using the driver class methods
%code {
   #include "driver/driver.hpp"

   std::shared_ptr<block> program_block;

   visitor_pprint v_pprint;
}

// to make sure there are no conflicts prepend TOK_
%define api.token.prefix{TOK_}
%token
   END  0  "end of file"
   AND     "and"
   OR      "or"
   NOT     "not"
   FOR     "for"
   IN      "in"
   RANGE   "range"
   IF      "if"
   ELSE    "else"
   ASSIGN  "="
   PLUS    "+"
   MINUS   "-"
```

```
  MUL    "*"
  DIV    "/"
  LPAREN  "("
  RPAREN  ")"
  LBRACE  "{"
  RBRACE  "}"
  LBRACKET "["
  RBRACKET "]"
  COMMA   ","
  GRT    ">"
  GRTEQ   ">="
  LES    "<"
  LESEQ   "<="
  NOTEQ   "!="
  EQUAL   "=="

%token  <std::unique_ptr<identifier>>   IDENT      "identifier"
%token  <std::unique_ptr<decimal>>     DECIMAL    "decimal"
%token  <std::unique_ptr<fraction>>    FRACTION   "fraction"
%token  <std::unique_ptr<stringlit>>   STRINGLIT   "stringlit"
%nterm  <std::unique_ptr<identifier>>   identifier // add this for verbosity
%nterm  <std::unique_ptr<expression>>   expr
%nterm  <std::unique_ptr<expression>>   literals
%nterm  <std::unique_ptr<expression>>   binop_expr
%nterm  <std::unique_ptr<expression>>   unaryop_expr
%nterm  <std::unique_ptr<expression>>   compare_expr
%nterm  <std::unique_ptr<block>>       stmts
%nterm  <std::unique_ptr<block>>       program
%nterm  <std::unique_ptr<block>>       block
%nterm  <std::unique_ptr<statement>>    stmt
%nterm  <std::unique_ptr<statement>>    conditional
%nterm  <std::unique_ptr<statement>>    for_loop
%nterm  <std::unique_ptr<statement>>    for_range
%nterm  <std::unique_ptr<std::vector<std::unique_ptr<expression>>>> call_args
%nterm  <std::unique_ptr<variable_declaration>> var_decl
%nterm  <std::unique_ptr<array_access>> array_access

%printer { yyo << $$; } <*>;

%start program;

%code {
   #define DEBUG_PARSER
   #undef DEBUG_PARSER
}

%%

// left associativity

%left "+" "-";
%left "*" "/";

// program consists of statements
```

```
program     : stmts {

            program_block = std::move($1);
            program_block->accept(v_pprint);

            }
          ;

// statements can consist of single or multiple statements

stmts[block]      : stmt {

            $block = std::make_unique<block>();

            $block->statements.emplace_back(std::move($1));
            $$->accept(v_pprint);

            }
          | stmts[meow] stmt {

            $meow->statements.emplace_back(std::move($2));

            // i added this because i std::move everytime and this moves the $block also
            // so i std::move back $meow to block to retain the address of main block
            // it was becoming null before, added null check in main.cpp as well
            // - shadowleaf

            $block = std::move($meow);

            }
          ;

// statement can be an expression or an variable declaration

stmt      : expr {
            $$ = std::make_unique<expr_statement>(std::move($1));
            $$->accept(v_pprint);
            }
          | var_decl {
            $$ = std::move($1);
            }
          | conditional {
            $$ = std::move($1);
             }
          | for_loop {
            $$ = std::move($1);
            $$->accept(v_pprint);
            }
          | for_range {
            $$ = std::move($1);
            $$->accept(v_pprint);
          }
```

```
        ;

// for loops

for_loop    : "for" "(" expr "," expr "," expr ")" block {
            $$ = std::make_unique<for_loop>(std::move($3), std::move($5), std::move($7), std::move($9));
        }
        ;

for_range   : "for" identifier "in" "range" "decimal" block {
            $$ = std::make_unique<for_range>(std::move($2), std::move($5), std::move($6));
        }
        ;

// a block

block      : "{" stmts "}" {
              $$ = std::move($2);
              $$->accept(v_pprint);
              }
        ;

// conditional statement

conditional     : "if" expr block "else" block {
                $$ = std::make_unique<conditional>(std::move($2), std::move($3), std::move($5));
                $$->accept(v_pprint);
                }
            | "if" expr block {
                $$ = std::make_unique<conditional>(std::move($2), std::move($3));
                $$->accept(v_pprint);
            }
            ;

// variable declaration and/or assignment

var_decl    : "identifier" "identifier" {
            $$ = std::make_unique<variable_declaration>(std::move($1), std::move($2));
            $$->accept(v_pprint);
        }
        | "identifier" "identifier" "=" expr {
            $$ = std::make_unique<variable_declaration>(std::move($1), std::move($2), std::move($4));
            $$->accept(v_pprint);
        }
        ;

// all the literals, like integers, fractions and string literals

literals    : "decimal"  {

            $$ = std::move($1);
            // LOG_S(INFO) << "found decimal at " << @1.begin.line << "." << @1.begin.column;
            $$->accept(v_pprint);
```

```
            }
        | "fraction" {

            $$ = std::move($1);
            $$->accept(v_pprint);

            }
        | "stringlit" {

            $$ = std::move($1);
            $$->accept(v_pprint);

        }
        ;

// all the expression statements

expr      : identifier "=" expr {

            $$ = std::make_unique<assignment>(std::move($1), std::move($3));
            $$->accept(v_pprint);

            }

        | identifier "(" call_args ")" {
            // function call

            $$ = std::make_unique<function_call>(std::move($1), std::move($3));
            $$->accept(v_pprint);

            }
        | identifier {
            // just an identifier

            $$ = std::move($1);
            $$->accept(v_pprint);

            }

        | literals {

            // literal, either decimal or fractional

            $$ = std::move($1);

            }
        | binop_expr {

            // some binary operation (numeric, not boolean)

            $$ = std::move($1);
            $$->accept(v_pprint);
            }
        | unaryop_expr {
```

```
            // a and or not, unary boolean expression

            $$ = std::move($1);
            }
          | compare_expr {
            // a comparison expression

            $$ = std::move($1);
            }
          | array_access {
            // accessing an element of array

            $$ = std::move($1);
            }
          | "(" expr ")" {
            $$ = std::move($2);
            $$->accept(v_pprint);
            }
          ;

identifier  : "identifier" { $$ = std::move($1); $$->accept(v_pprint); }

// call arguments of a function
// can be blank

call_args[args_list]   : /*blank*/ {
            $args_list = std::make_unique<std::vector<std::unique_ptr<expression>>>();
            }
          | expr {
            $args_list = std::make_unique<std::vector<std::unique_ptr<expression>>>();
            $args_list->push_back(std::move($1));
            }
          | call_args[arg] "," expr {
            $arg->push_back(std::move($3));
            $args_list = std::move($arg);
            }
          ;

// array access for arr[0], arr[<some expr that evaluate to decimal>]
// or for the future can also be arr['string'] for maps
array_access    : identifier "[" expr "]" {
                $$ = std::make_unique<array_access>(std::move($1), std::move($3));
                $$->accept(v_pprint);
                }
              | array_access "[" expr "]" {
                $$ = std::make_unique<array_access>(std::move($1), std::move($3));
                $$->accept(v_pprint);
              }
              ;

// binary operators

binop_expr  : expr "and" expr {
```

```
            $$ = std::make_unique<binary_operator>('&', std::move($1), std::move($3), @$);
            $$->accept(v_pprint);

            }
          | expr "or" expr {

            $$ = std::make_unique<binary_operator>('|', std::move($1), std::move($3), @$);
            $$->accept(v_pprint);

            }
          | expr "+" expr {
            $$ = std::make_unique<binary_operator>('+', std::move($1), std::move($3), @$);
            $$->accept(v_pprint);

            }
          | expr "-" expr {
            $$ = std::make_unique<binary_operator>('-', std::move($1), std::move($3), @$);
            $$->accept(v_pprint);
            }
          | expr "*" expr {
            $$ = std::make_unique<binary_operator>('*', std::move($1), std::move($3), @$);
            $$->accept(v_pprint);
            }
          | expr "/" expr {
            $$ = std::make_unique<binary_operator>('/', std::move($1), std::move($3), @$);
            $$->accept(v_pprint);
            }
          ;

// binary boolean comparison operators
compare_expr    :   expr ">" expr {
                      $$ = std::make_unique<comp_operator>(">", std::move($1), std::move($3));
                      $$->accept(v_pprint);
                    }
              |   expr ">=" expr {
                      $$ = std::make_unique<comp_operator>(">=", std::move($1), std::move($3));
                      $$->accept(v_pprint);
                    }
              |   expr "<" expr {

                      $$ = std::make_unique<comp_operator>("<", std::move($1), std::move($3));
                      $$->accept(v_pprint);
                    }
              |   expr "<=" expr {

                      $$ = std::make_unique<comp_operator>("<=", std::move($1), std::move($3));
                      $$->accept(v_pprint);
                    }
              |   expr "==" expr {

                      $$ = std::make_unique<comp_operator>("==", std::move($1), std::move($3));
                      $$->accept(v_pprint);
                    }
              |   expr "!=" expr {
```

```
                $$ = std::make_unique<comp_operator>("!=", std::move($1), std::move($3));
                $$->accept(v_pprint);
            }
            ;

// unary operations

unaryop_expr    : "not" expr {
                $$ = std::make_unique<unary_operator>('!', std::move($2), @$);
                $$->accept(v_pprint);
                }
            ;

// // boolean expression

// boolean_expr    : expr "and" expr {

//                 }
//             | expr "or" expr {

//                 }
//             | expr "xor" expr {

//                 }

/* testing out a grammar */
/*
program     : expr { std::cout << "expr: " << cnt++ << "\n"; }
        ;

expr        : "decimal" { std::cout<< "decimal: " << cnt++ << "\n"; $$ = std::move($1); }
          | expr "+" expr { std::cout << "expr + expr: " << cnt++ << "\n"; $$ = std::make_unique<binary_operator>('+', s
td::move($1), std::move($3)); }
        ;
*/
%%

void yy::parser::error (const location_type& l, const std::string& m) {
  std::cerr << l << ": " << m << '\n';
}
```

1.6 Testing

To test the grammar various test cases were made, the program made for the compiler can also generate IR code and then use the LLVM MCJIT (Machine Code Just In Time) Compiler to execute the generated IR.

```
shadowleaf@shadowleaf-manjaro …/Projects/ProjektBarium
build/barium/barium -v INFO stdin --parse-only
date       time                           file:line     v|
2020-04-01 04:50:04.907             loguru.cpp:610   INFO| arguments: build/barium/barium -v INFO stdin --parse-only
2020-04-01 04:50:04.907             loguru.cpp:613   INFO| Current dir: /mnt/data/Projects/ProjektBarium
2020-04-01 04:50:04.907             loguru.cpp:615   INFO| stderr verbosity: 0
2020-04-01 04:50:04.907             loguru.cpp:616   INFO| ----------------------------------
2020-04-01 04:50:04.908               main.cpp:82    INFO| DEBUG INFO PARSER
2 + 3 * 4 - 1
2020-04-01 04:50:29.365      visitor_pprint.cpp:34    INFO| created decimal [ addr: 0x558b2ef20900, value: 2 ]
2020-04-01 04:50:29.366      visitor_pprint.cpp:34    INFO| created decimal [ addr: 0x558b2ef209e0, value: 3 ]
2020-04-01 04:50:29.366      visitor_pprint.cpp:34    INFO| created decimal [ addr: 0x558b2ef45230, value: 4 ]
2020-04-01 04:50:29.366      visitor_pprint.cpp:46    INFO| created binary operator [ addr: 0x558b2ef227a0, op: *, lhs addr: 0x558b2ef209e0, rhs addr: 0x558b2ef45230 ]
2020-04-01 04:50:29.366      visitor_pprint.cpp:46    INFO| created binary operator [ addr: 0x558b2ef227a0, op: *, lhs addr: 0x558b2ef209e0, rhs addr: 0x558b2ef45230 ]
2020-04-01 04:50:29.366      visitor_pprint.cpp:46    INFO| created binary operator [ addr: 0x558b2ef21270, op: +, lhs addr: 0x558b2ef20900, rhs addr: 0x558b2ef227a0 ]
2020-04-01 04:50:29.366      visitor_pprint.cpp:46    INFO| created binary operator [ addr: 0x558b2ef21270, op: +, lhs addr: 0x558b2ef20900, rhs addr: 0x558b2ef227a0 ]
2020-04-01 04:50:29.366      visitor_pprint.cpp:34    INFO| created decimal [ addr: 0x558b2ef45290, value: 1 ]
2020-04-01 04:50:32.071      visitor_pprint.cpp:46    INFO| created binary operator [ addr: 0x558b2ef44490, op: -, lhs addr: 0x558b2ef21270, rhs addr: 0x558b2ef45290 ]
2020-04-01 04:50:32.072      visitor_pprint.cpp:46    INFO| created binary operator [ addr: 0x558b2ef44490, op: -, lhs addr: 0x558b2ef21270, rhs addr: 0x558b2ef45290 ]
2020-04-01 04:50:32.072      visitor_pprint.cpp:27    INFO| created expr_statement [ addr: 0x558b2ef20c50 ]
2020-04-01 04:50:32.072      visitor_pprint.cpp:28    INFO| { visit_expr_statement
2020-04-01 04:50:32.072      visitor_pprint.cpp:29    INFO| .   contains expression [ addr: 0x558b2ef44490 ]
2020-04-01 04:50:32.072      visitor_pprint.cpp:28    INFO| } 0.000 s: visit_expr_statement
2020-04-01 04:50:32.072      visitor_pprint.cpp:58    INFO| created block [ addr: 0x558b2ef45a70 ]
2020-04-01 04:50:32.072      visitor_pprint.cpp:60    INFO| { visit_block
2020-04-01 04:50:32.072      visitor_pprint.cpp:62    INFO| .   contains statement [ addr: 0x558b2ef20c50 ]
2020-04-01 04:50:32.073      visitor_pprint.cpp:60    INFO| } 0.000 s: visit_block
2020-04-01 04:50:32.073      visitor_pprint.cpp:58    INFO| created block [ addr: 0x558b2ef45a70 ]
2020-04-01 04:50:32.073      visitor_pprint.cpp:60    INFO| { visit_block
2020-04-01 04:50:32.073      visitor_pprint.cpp:62    INFO| .   contains statement [ addr: 0x558b2ef20c50 ]
2020-04-01 04:50:32.073      visitor_pprint.cpp:60    INFO| } 0.000 s: visit_block
2020-04-01 04:50:32.073             loguru.cpp:489    INFO| atexit
shadowleaf@shadowleaf-manjaro …/Projects/ProjektBarium
```

Figure 1-3 Parsing Simple Arithmetic Expression

Input 2 + 3 * 4 - 1 is given to the program, first the lexical analyzer converts this character stream to tokens, i.e. DECIMAL, PLUS, DECIMAL, MUL, DECIMAL, MINUS, DECIMAL, as defined by our tokens earlier.

These tokens are fed to the scanner, which uses the grammar rules that we provided to perform actions on matching the syntax of the tokens. For example, when 2 + 3 is found, expr + expr is matched and then into a binary operator. Similarly, it is done for the entire program.

The Abstract Syntax Tree is thus generated, while doing so we print the nodes of the tree, which can be seen in the terminal as form of LOG INFO, the address of the node and its contents are printed, we can use this information to create a visual syntax tree, so it'll be easier for us to comprehend. Refer to Figure 1-4 AST for 2 + 3 * 4 - 1, we can see how our expression is converted to a AST, the leaf nodes are the terminals, which are all decimals in our case, few reduce operations are omitted, for optimizations, like decimal is reduced to expr and then attached to binary_operator, but bison optimizes this and directly does the reduction operation.

From the figure, we can see that operator associativity and precedence is maintained as written in the grammar file.

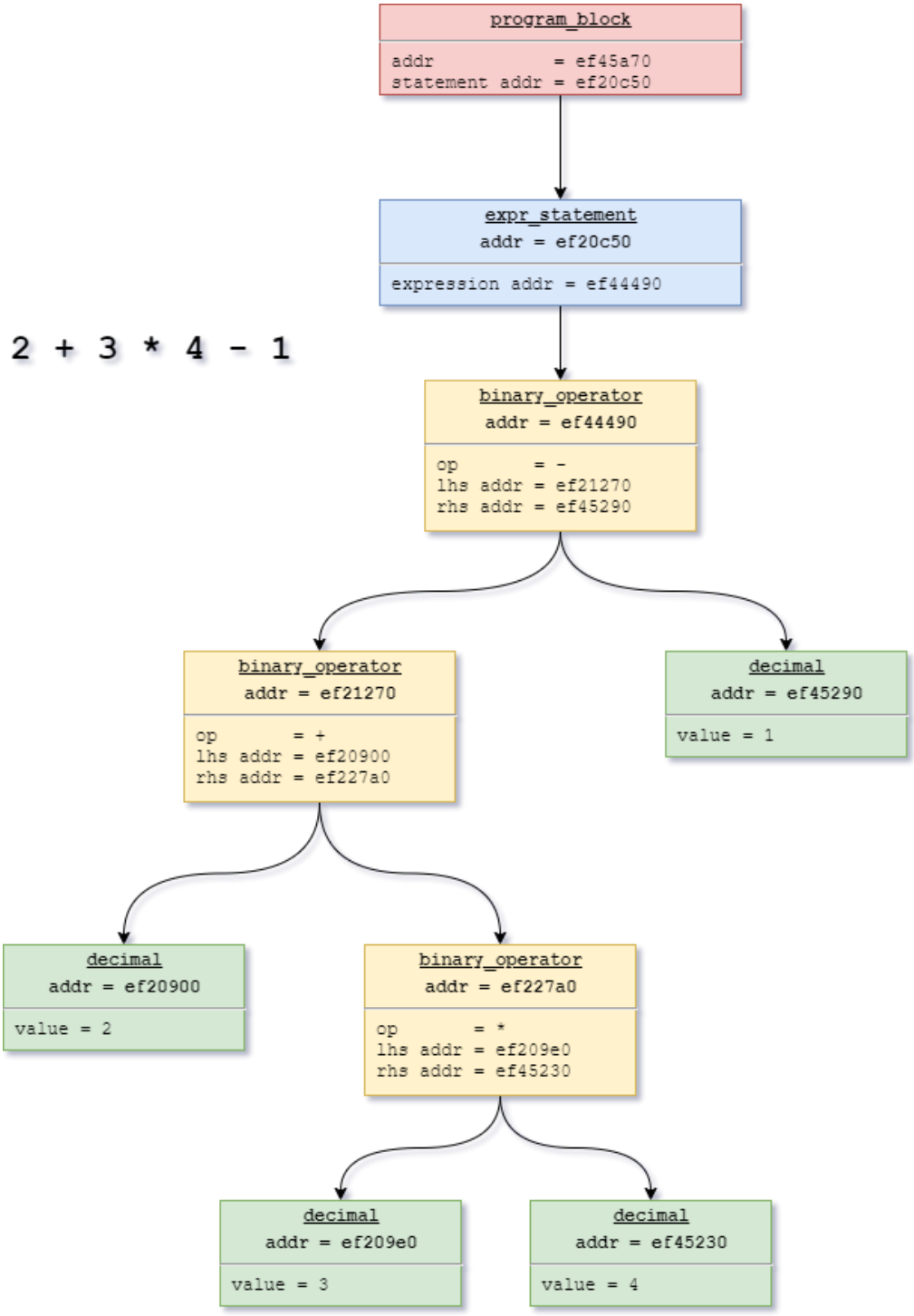


Figure 1-4 AST for 2 + 3 * 4 - 1

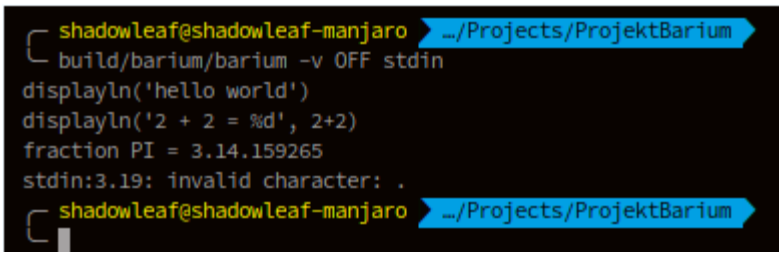


Figure 1-5 Fractional Number Syntax Error

The above figure demonstrates the syntax error caused due to wrong notation of the fraction number.

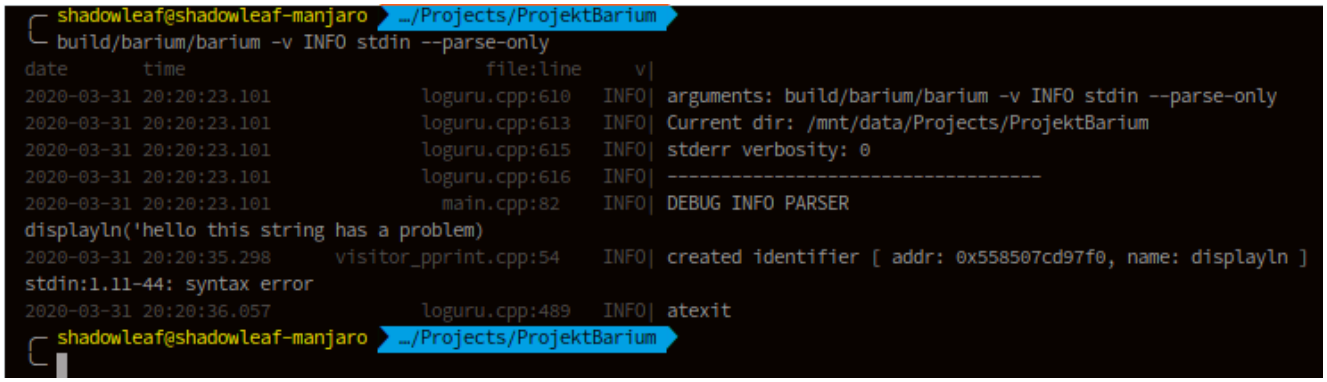


Figure 1-6 String Syntax Error

The above figure demonstrates the syntax error where a single terminating quote of the string is missing.

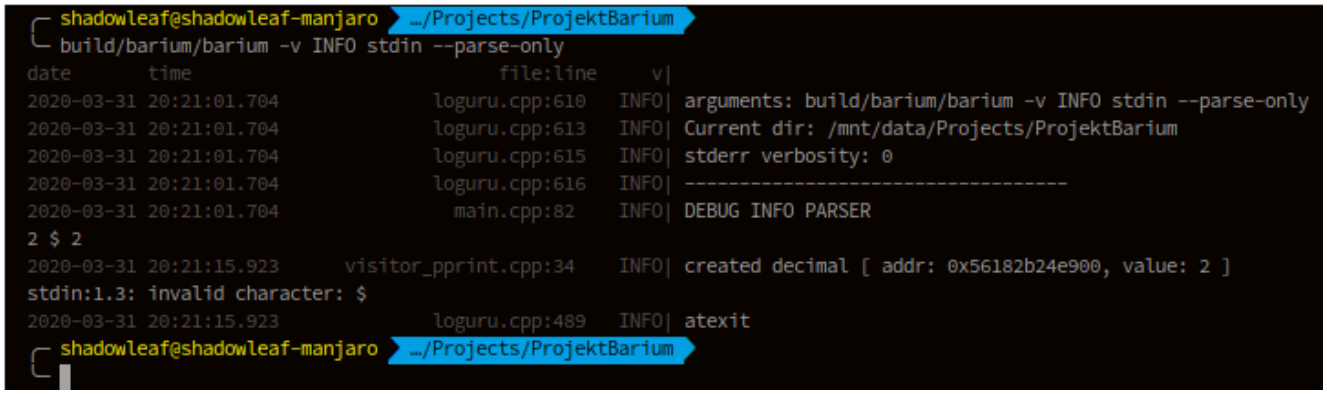


Figure 1-7 Parse Error, unrecognized character

The above figure demonstrates the parser throws a syntax error when an invalid character is given to the program.

Figure 1-8 Simple JIT Compile and Execute

The above figure shows a simple program, that is passed through all the stages of compilation as shown in Figure 1-1 Compiler Recipe, the output is displayed. *Comments are ignored in the source code.*

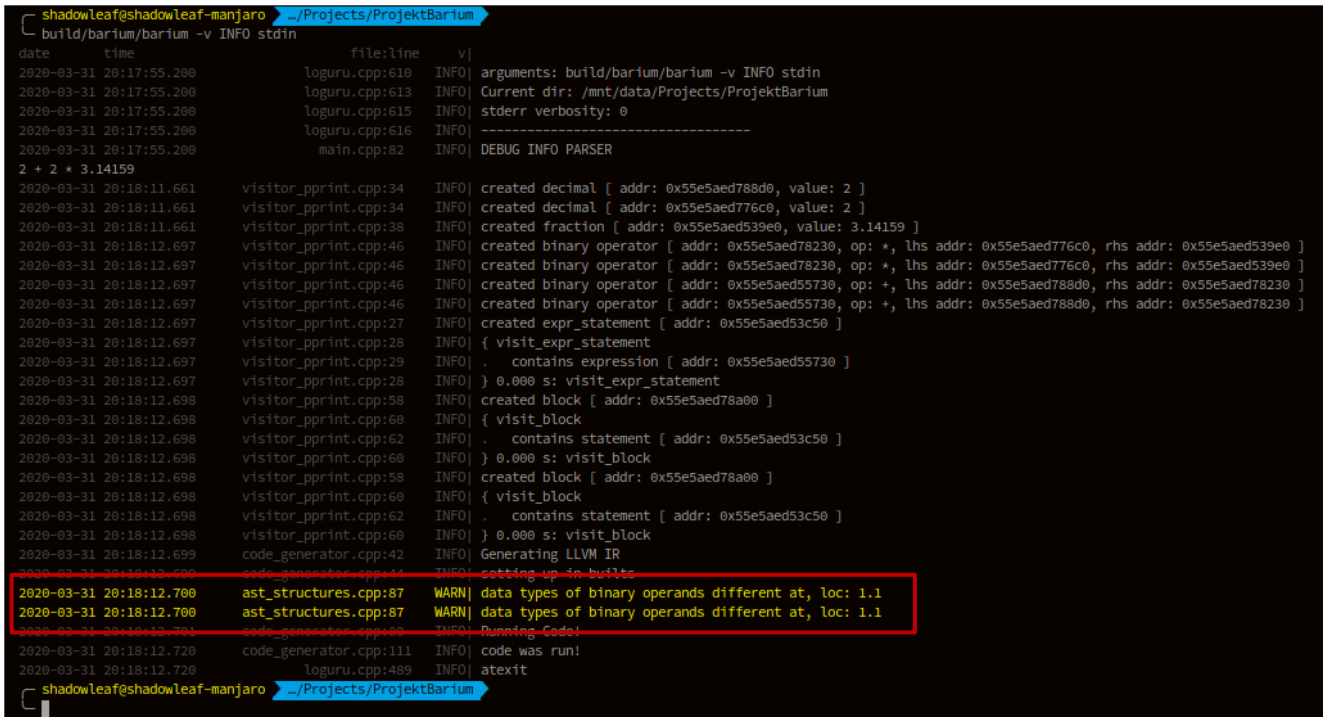


Figure 1-9 Semantic Analyzer data type cast warnings

The above program shows how the program shows a type casting warning when we've tried to do 2 + 2 + 3.14159, i.e. addition of a decimal to a fraction.

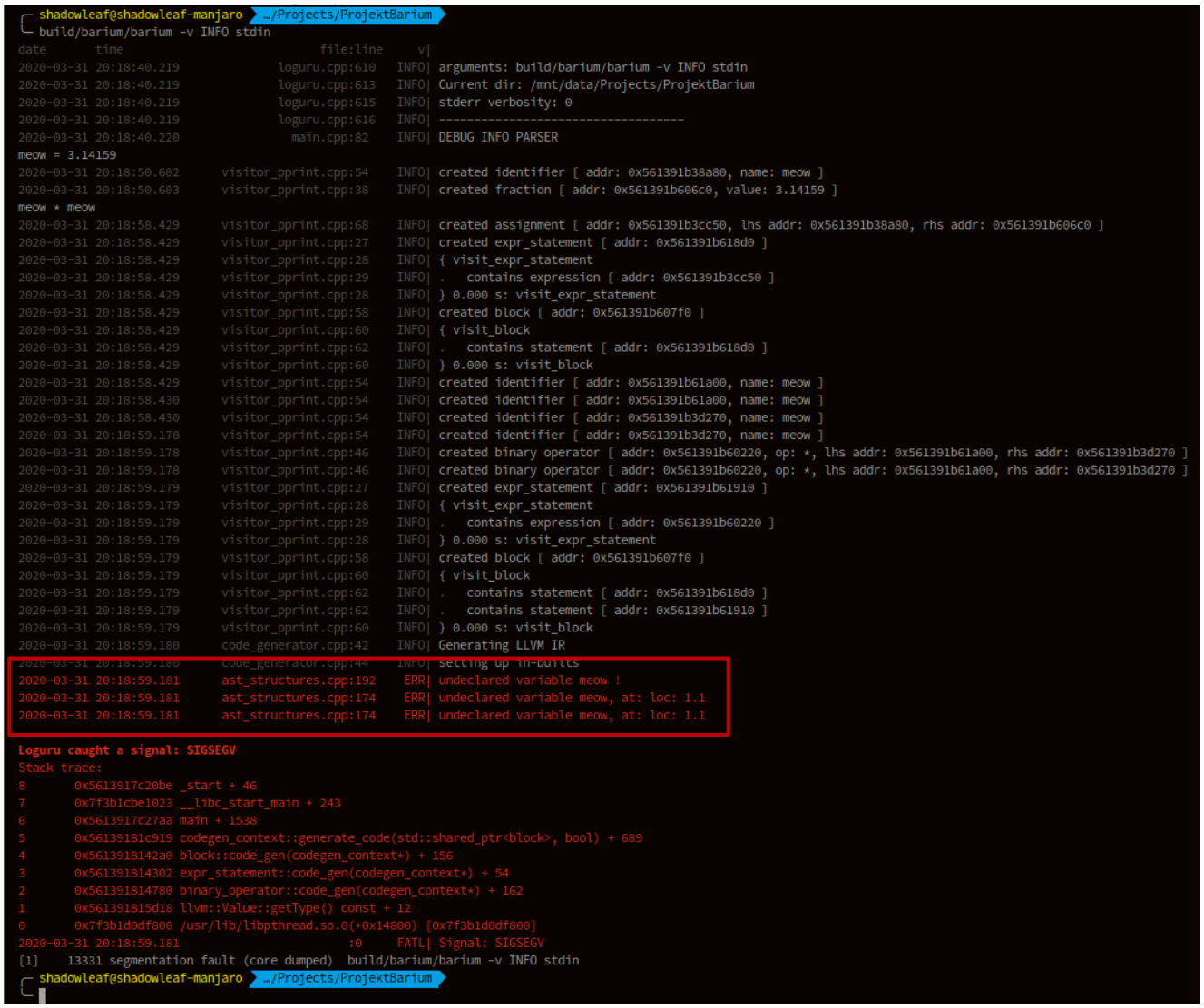


Figure 1-10 Undeclared Variable Error

The above figure shows how the program throws an error when we try to assign value to a variable which is undeclared, this happens during the Intermediate Code Generation stage.

Control Statements, Looping Statements, Arrays and IO Statements run are attached in Appendix A Logs, the program was tested and it works in parsing them.

1.7 Results and Comments

```
shadowleaf@shadowleaf-manjaro …/Projects/ProjektBarium
└─ build/barium/barium -h -v OFF
OVERVIEW: Barium Compiler
Uses STDIN if <input file> is not specified
-v [OFF | INFO | ERROR] sets the verbosity level
USAGE: barium [options] <input file>

OPTIONS:

General options:

  --dump-ir     - Dump the Generated IR on output
  -o=<filename> - Specify output filename
  --parse-only  - Only Parse the source file

Generic Options:

  --help        - Display available options (--help-hidden for more)
  --help-list   - Display list of available options (--help-list-hidden for more)
  --version     - Display the version of this program
shadowleaf@shadowleaf-manjaro …/Projects/ProjektBarium
└─
```

Figure 1-11 ProjektBarium help screen

So, this was ProjektBarium, a simple, tiny compiler using the LLVM Frontend to generate IR and execute the code using the built in MCJIT compiler.

1.7.1 Limitations

The language is very limited and cannot be called a language since we didn't do a full implementation of functions and modules. LLVM has full support for these features and even more!

The language is missing recursion, which is very fundamental when it comes to writing some of our basic data structures like linked lists.

1.7.2 Further Improvements

The goal of project barium was to create a functional language, but things didn't turn out so well, i plan to do so at some later point in time, i.e. restructure the grammar to form a functional language. But why? Function languages are fundamentally simple and have a very simple syntax, but they are very powerful, everything is a function, numbers are encoded as Churchill Encodings, which are Lambdas, which is a function. This makes it easy to create a full-fledged language with very less code.

For Appendix and Header file Source code Please Check https://github.com/ThisisDeepakR/CompilersProject

Bibliography

1. The Architecture of Open Source Applications http://www.aosabook.org/en/llvm.html
2. Loren Segal, Writing Your Own Toy Compiler Using Flex, Bison and LLVM, https://gnuu.org/2009/09/18/writing-your-own-toy-compiler/
3. LLVM Kaleidoscope Tutorial, https://llvm.org/docs/tutorial/MyFirstLanguageFrontend/
4. Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D, Compilers: Principles, Techniques, and Tools
5. Regents of the University of California, Flex, version 2.5 A fast scanner generator https://www.cs.princeton.edu/~appel/modern/c/software/flex/flex.html

For Appendix and Header file Source code Please Check https://github.com/ThisisDeepakR/CompilersProject