

***Report on***

**“C++ Compiler for IF-ELSE and WHILE construct”**

*Submitted in partial fulfillment of the requirements for* ***Sem VI***

***Compiler Design Laboratory***

**Bachelor of Technology**

**in**

**Computer Science & Engineering**

***Submitted by:***

|  |  |
| --- | --- |
| **Vipul Raj**  **Vishwas M Joshi**  **Vivek R** | **01FB16ECS448**  **01FB16ECS453**  **01FB16ECS455** |

*Under the guidance of*

|  |
| --- |
| Mr. Prakash C O  Asst. Professor,  Dept. of CSE, PESU, |

**January – May 2019**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

FACULTY OF ENGINEERING

**PES UNIVERSITY**

(Established under Karnataka Act No. 16 of 2013)

100ft Ring Road, Bengaluru – 560 085, Karnataka, India

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
|  | **INTRODUCTION (Mini-Compiler is built for which language. Provide sample input and output of your project)** |

Mini Compiler is built for C++ language which is an imperative, object-oriented and generic programming features. C++ runs on lots of platform like Windows, Linux, Unix, Mac etc.

The input for our project is a file with .cpp extension which is a c++ code .

The output contains Intermediate code and abstract syntax tree for the same c++ code .

|  |  |
| --- | --- |
| **2.** | **ARCHITECTURE OF LANGUAGE:**   * **What all have you handled in terms of syntax and semantics for the chosen language.** |

The compiler designed handles syntax errors like keyword names , parenthesis matching . In terms for semantic analysis it does intermediate code generation and syntax tree generation .

|  |  |
| --- | --- |
| **3.** | **LITERATURE SURVEY (if any paper referred or link used)** |

Lex and YACC by Tom Niemann

PES slides by C O Prakash Sir

|  |  |
| --- | --- |
| **4.** | **CONTEXT FREE GRAMMAR (which you used to implement your project)** |

start: Declaration

| Function

| start Function

| start Declaration

;

/\* Declaration block \*/

Declaration: Type Assignment ';'

| Assignment ';'

| FunctionCall ';'

| ArrayUsage ';'

| Type ArrayUsage ';'

| StructStmt ';'

| ClassStmt ';'

| error

;

Arg: Type ID

;

/\* Assignment block \*/

Assignment: ID '=' Assignment

| ID '=' FunctionCall

| ID '=' ArrayUsage

| ArrayUsage '=' Assignment

| ID ',' Assignment

| NUM ',' Assignment

| ID '+' Assignment

| ID '-' Assignment

| ID '\*' Assignment

| ID '/' Assignment

| NUM '+' Assignment

| NUM '-' Assignment

| NUM '\*' Assignment

| NUM '/' Assignment

| '\'' Assignment '\''

| '(' Assignment ')'

| '-' '(' Assignment ')'

| '-' NUM

| '-' ID

| NUM

| ID

;

/\* Function Call Block \*/

FunctionCall : ID'('')'

| ID'('Assignment')'

| CUSTOM DOT FNAME '('')'

| CUSTOM DOT NFNAM '('')' {printf("unknown function\n");return 0;}

;

/\* Array Usage \*/

ArrayUsage : ID'['Assignment']'

;

/\* Function block \*/

Function: Type ID '(' ArgListOpt ')' CompoundStmt

| Type FNAME '('')' CompoundStmt

;

ArgListOpt: ArgList

|

;

ArgList: ArgList ',' Arg

| Arg

;

CompoundStmt: '{' StmtList '}'

;

StmtList: StmtList Stmt

|

;

Stmt: WhileStmt

| Declaration

| ForStmt

| IfStmt

| PrintFunc

| coutstatement

| cinstatement

| ';'

;

/\* Type Identifier block \*/

Type: INT

| FLOAT

| CHAR

| DOUBLE

| VOID

| CUSTOM {printf("here\n");

if (className[$1] == 0) {

printf("Error: Unknown Class ID\nQuiting!");

return 0;

}

}

;

/\* Loop Blocks \*/

WhileStmt: WHILE '(' Expr ')' Stmt

| WHILE '(' Expr ')' CompoundStmt

;

/\* For Block \*/

ForStmt: FOR '(' Expr ';' Expr ';' Expr ')' Stmt

| FOR '(' Expr ';' Expr ';' Expr ')' CompoundStmt

| FOR '(' Expr ')' Stmt

| FOR '(' Expr ')' CompoundStmt

;

/\* IfStmt Block \*/

IfStmt : IF '(' Expr ')'

Stmt

;

/\* Struct Statement \*/

StructStmt : STRUCT ID '{' Declaration '}'

;

ClassStmt : CLASS ID '{' ACCESS ':' start ACCESS ':' start '}'

{ className[$2] = 1; printf("value of classvariable = %d\n", $2);}

| CLASS ID ':' ACCESS ID '{' ACCESS ':' start ACCESS ':' start '}'

{ className[$2] = 1; printf("value of classvariable = %d\n", $2); }

;

/\* Print Function \*/

PrintFunc : PRINTF '(' Expr ')' ';'

;

/\*Expression Block\*/

Expr:

| Expr LE Expr

| Expr GE Expr

| Expr NE Expr

| Expr EQ Expr

| Expr GT Expr

| Expr LT Expr

| Assignment

| ArrayUsage

;

|  |  |
| --- | --- |
| **5.** | **DESIGN STRATEGY (used to implement the following)**   * **SYMBOL TABLE CREATION** * **ABSTRACT SYNTAX TREE** * **INTERMEDIATE CODE GENERATION** * **CODE OPTIMIZATION** * **ERROR HANDLING *-* strategies and solutions used in your Mini-Compiler implementation (in its scanner, parser, semantic analyzer, and code generator).** |

Symbol table contains 4 columns in which it contains token , type associated with token , line number and variables scope .

The syntax tree generated is in inorder where operator is the root and operands are its children .

Intermediate allocates new variables with codegen() function and contains label and goto constructs .

For error handling checking for valid type and parenthesis matching by incrementing a flag variable for opening parenthesis and decrementing for closing parenthesis .

|  |  |
| --- | --- |
| **6.** | **IMPLEMENTATION DETAILS (TOOL AND DATA STRUCTURES USED in order to implement the following):**   * **SYMBOL TABLE CREATION** * **ABSTRACT SYNTAX TREE (internal representation)** * **INTERMEDIATE CODE GENERATION** * **CODE OPTIMIZATION** * **ERROR HANDLING *-* strategies and solutions used in your Mini-Compiler implementation (in its scanner, parser, semantic analyzer, and code generator).** * **Provide instructions on how to build and run your computer** |

The data structure used for symbol table creature is structure with id\_type , id\_name,lineno as its instance .

The design used for AST is tree with linked list as data structure containing token left and right attributes of a node .

The data structure used for intermediate code generation is stack with top pointer containing element at top position

Intermediate allocates new variables with codegen() function and contains label and goto constructs .

For error handling checking for valid type and parenthesis matching by incrementing a flag variable for opening parenthesis and decrementing for closing parenthesis .

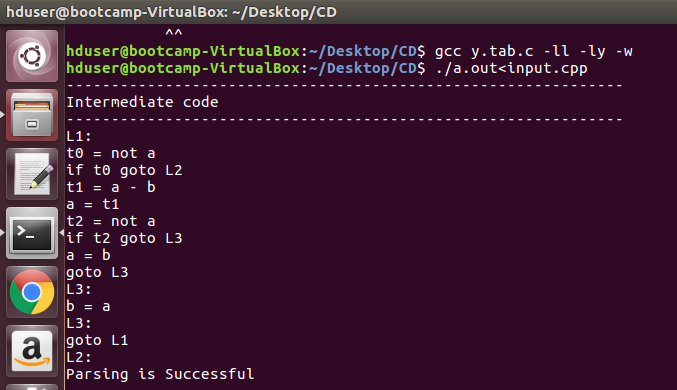
|  |  |
| --- | --- |
| **7.** | **RESULTS AND possible shortcomings of your Mini-Compiler** |

The result of our compiler will be output of the parser where it does semantic analysis by generating intermediate code and abstract syntax tree.

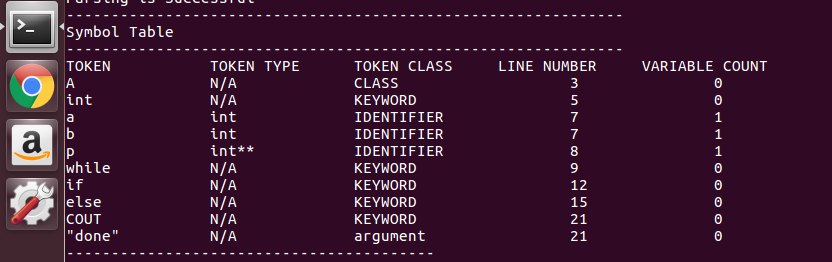
The short coming of our project is it doesn’t check looping construct like DO-WHILE and FOR loop and also doesn’t check for array pointers .

|  |  |  |
| --- | --- | --- |
|  | **SNAPSHOTS (of different outputs)** |  |

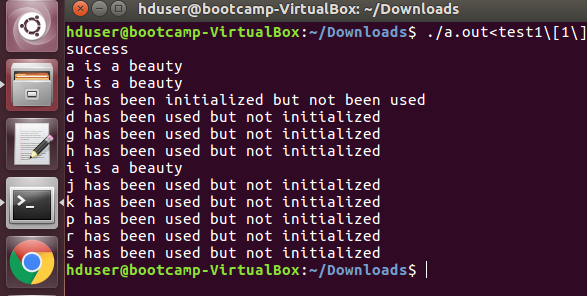
**Intermediate Code :**



**Symbol Table :**



**Abstract Syntax Tree**



|  |  |
| --- | --- |
| **9.** | **CONCLUSIONS** |

Many high-level libraries use C++ as the core programming language. For instance, several Machine Learning libraries use C++ in the backend because of its speed. [Tensorflow](https://www.tensorflow.org/), one of the most widely used Machine Learning libraries uses C++ as the backend programming language. So with an idea as to how the compiler works we can able to create own compiler with keeping the principles of programming language in mind by which the compiler can be used for various fields .

|  |  |
| --- | --- |
| **10.** | **FURTHER ENHANCEMENTS** |

The compilers offered by the programming language are strongly typed which makes grammar very strict so many programmers find it difficult to maintain discipline everytime they code . So with growing technology coders want to reduce time for writing code as a result we can make it weakly typed so that time is saved . Thus the grammar can be made less strict which would be a small contribution to time demanding technology .