National Institute of Technology, Delhi

CSB 353: Compiler Design

NP-C Compiler Project Report

Submitted By:

Abel Sherin Ukken, 191210001

Jeremy Joseph Abraham, 191210025

Submitted to:

Dr Shelly Sachdeva

Department of Computer Science and Engineering, NIT DELHI

Table of Contents

Chapter 1: Introduction

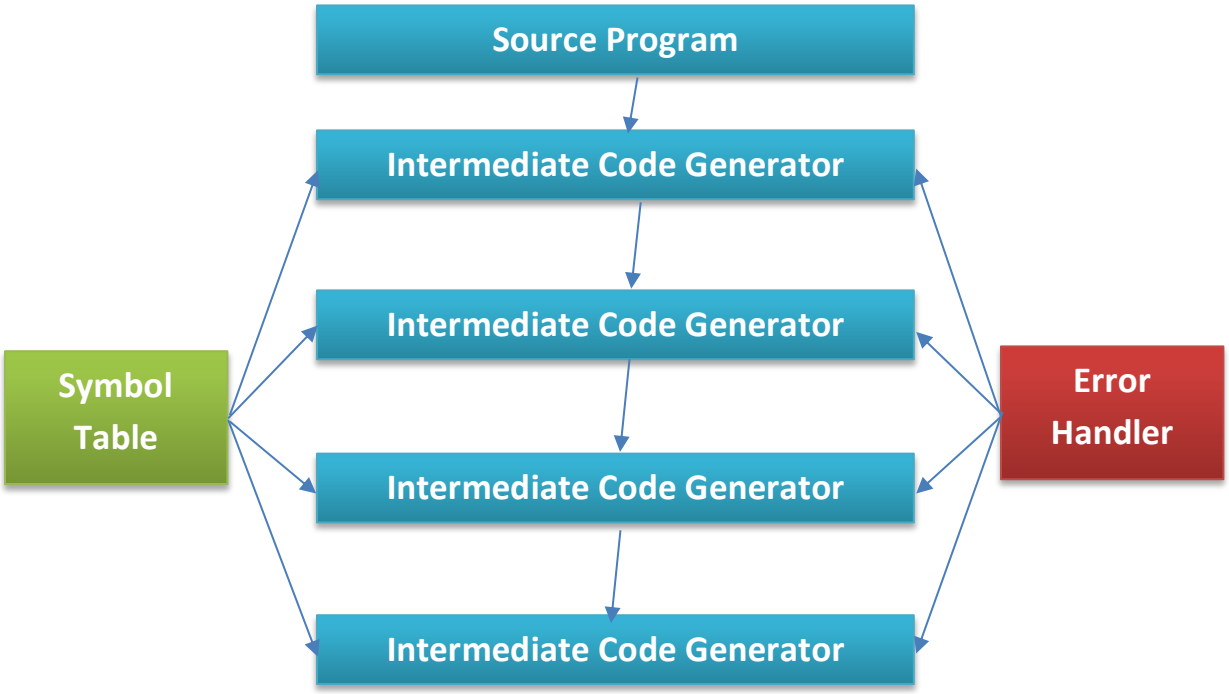
Objective

To construct an NP – C Compiler with the following functionalities:

- Taking Source code as input and generating the three address intermediate code
- Identifying various types of errors.
- Display the result of each phase with errors if they occur.

Introduction

A compiler is a software that converts a program written in a high-level language to an equivalent program in a low-level language.



Domain Language

The NP – C Compiler is a compiler based on the C languages but simplifies it to its basic funcationalites. The compiler is able to parse loops, conditional statements, arrays etc. The main features of our language are:

- Data Types: int, float, char

- Identifiers: Abc, d2, _ef (Same as C)
- Arithemetic Operators: +, -, *, /
- Relational Operators: ==, >=, <=, !=
- Conditional Statements: if, if … else
- Loops: for, while
- Comments: // (for single-line comments), /*…*/ (for multiline comments)
- Function Statements: type functionName (arguments)

Target Language

The compiler generates the three address code for the given source program provided there are no errors while parsing the program.

Phases of our Compiler

- Lexical Analysis: The lexical scanner takes the source program as the input and generates the token stream for the program. A symbol table is also created and displayed.
- Syntax Analysis: The parser checks for any syntax errors while parsing the program. If an error is found, the parsing stops and the error is reported.
- Semantic Analysis: The semantics of the code will be checked, i.e., type checking, function declarations.
- Intermediate Code Generation: An intermediate representation of the code will be generated. We will be using three address code representations for this phase.

After each phase, we have displayed the results of the parsing along with the symbol and constant tables.

Hardware Requirements

- Operating System – Linux / Windows
- Memory (RAM): 2 GB RAM required.
- Hard Disk Space: 200 MB of free space required.
- Processor: Intel Pentium 4 or later.

Software Requirements

- FLEX – Fast Lexical Analyzer Generator.
- YACC – Yet Another Compiler Compiler.
- GCC – GNU Compiler Collection.
- Text Editor (VSCode or any equivalent software).

Chapter 2: Lexical Analysis

The lexical scanner takes the source program as the input and generates the token stream for the program. A symbol table is also created and displayed.

Test Case 1:

Input:

```
#include<stdio.h>

int fun(int x){
    return x*x;
}

int main(){
    int a=2,b,c,d,e,f,g,h;

    c=a+b;
    d=a*b;
    e=a/b;
    f=a%b;
    g=a&&b;
    h=a||b;
    h=a*(a+b);
    h=a*a+b*b;
    h=fun(b);

    //contains operator,structure,delimeters,Function;
}
```

Output:

```
======================================================================
#include<stdio.h>       -Pre Processor directive
int     - KEYWORD
fun     - IDENTIFIER
(       - OPENING BRACKETS
int     - KEYWORD
x       - IDENTIFIER
)       - CLOSING BRACKETS
{       - OPENING BRACES
return  - KEYWORD
x       - IDENTIFIER
*       - OPERATOR
x       - IDENTIFIER
;       - SEMICOLON DELIMITER
}       - CLOSING BRACES
int     - KEYWORD
main    - KEYWORD
(       - OPENING BRACKETS
)       - CLOSING BRACKETS
```

```
{       - OPENING BRACES
int     - KEYWORD
a       - IDENTIFIER
=       - OPERATOR
2       - NUMBER CONSTANT
,       - COMMA DELIMITER
b       - IDENTIFIER
,       - COMMA DELIMITER
c       - IDENTIFIER
,       - COMMA DELIMITER
d       - IDENTIFIER
,       - COMMA DELIMITER
e       - IDENTIFIER
,       - COMMA DELIMITER
f       - IDENTIFIER
,       - COMMA DELIMITER
g       - IDENTIFIER
,       - COMMA DELIMITER
h       - IDENTIFIER
;       - SEMICOLON DELIMITER
c       - IDENTIFIER
=       - OPERATOR
a       - IDENTIFIER
+       - OPERATOR
b       - IDENTIFIER
;       - SEMICOLON DELIMITER
d       - IDENTIFIER
=       - OPERATOR
a       - IDENTIFIER
a       - IDENTIFIER
+       - OPERATOR
b       - IDENTIFIER
*       - OPERATOR
b       - IDENTIFIER
;       - SEMICOLON DELIMITER
h       - IDENTIFIER
=       - OPERATOR
fun     - IDENTIFIER
(       - OPENING BRACKETS
b       - IDENTIFIER
)       - CLOSING BRACKETS
;       - SEMICOLON DELIMITER
//contains operator,structure,delimeters,Function;      - SINGLE LINE COMMENT
}       - CLOSING BRACES


SYMBOL TABLE

a       IDENTIFIER
b       IDENTIFIER
c       IDENTIFIER
d       IDENTIFIER
e       IDENTIFIER
f       IDENTIFIER
g       IDENTIFIER
h       IDENTIFIER
x       IDENTIFIER
fun     IDENTIFIER
return  KEYWORD
int     KEYWORD
main    KEYWORD


CONSTANT TABLE

2       NUMBER CONSTANT
```

Test Case 2:

Input:

```c
#include<stdio.h>

int main()
{
    //Program to add 2 numbers and increment by 1
    int a[3] = { 1, 2 };
    a[2] = a[1] + a[2];
    a[2]++;

    printf("%d", a[2]);

    return 0;
}
```

Output:

```
#include<stdio.h>       -Pre Processor directive
int     - KEYWORD
main    - KEYWORD
(       - OPENING BRACKETS
)       - CLOSING BRACKETS
{       - OPENING BRACES
//Program to add 2 numbers and increment by 1    - SINGLE LINE COMMENT
int     - KEYWORD
a       - ARRAY IDENTIFIER
[       - SQUARE OPENING BRACKETS
3       - NUMBER CONSTANT
]       - SQUARE CLOSING BRACKETS
=       - OPERATOR
{       - OPENING BRACES
1       - NUMBER CONSTANT
,       - COMMA DELIMITER
2       - NUMBER CONSTANT
}       - CLOSING BRACES
;       - SEMICOLON DELIMITER
a       - ARRAY IDENTIFIER
[       - SQUARE OPENING BRACKETS
2       - NUMBER CONSTANT
]       - SQUARE CLOSING BRACKETS
=       - OPERATOR
a       - ARRAY IDENTIFIER
[       - SQUARE OPENING BRACKETS
1       - NUMBER CONSTANT
]       - SQUARE CLOSING BRACKETS
+       - OPERATOR
a       - ARRAY IDENTIFIER
[       - SQUARE OPENING BRACKETS
2       - NUMBER CONSTANT
]       - SQUARE CLOSING BRACKETS
;       - SEMICOLON DELIMITER
a       - ARRAY IDENTIFIER
[       - SQUARE OPENING BRACKETS
```

```
;       - SEMICOLON DELIMITER
return  - KEYWORD
0       - NUMBER CONSTANT
;       - SEMICOLON DELIMITER
}       - CLOSING BRACES


SYMBOL TABLE

a       IDENTIFIER
return  KEYWORD
int     KEYWORD
main    KEYWORD
printf  IDENTIFIER


CONSTANT TABLE

"%d"    STRING CONSTANT
0       NUMBER CONSTANT
1       NUMBER CONSTANT
2       NUMBER CONSTANT
3       NUMBER CONSTANT
```

Test Case 3:

Input:

```c
#include<stdio.h>

int main()
{
    int a = 2;
    printf("%d",a);
    a++;
    int b = 4;
    int c = 3;

    int b = 8; /* assigning b=8
    int c = 3;
    int d = c*(a+b);
    a--;
}
```

Output:

```
#include<stdio.h>       -Pre Processor directive
int     - KEYWORD
main    - KEYWORD
(       - OPENING BRACKETS
)       - CLOSING BRACKETS
{       - OPENING BRACES
int     - KEYWORD
a       - IDENTIFIER
=       - OPERATOR
2       - NUMBER CONSTANT
;       - SEMICOLON DELIMITER
printf  - IDENTIFIER
(       - OPENING BRACKETS
"%d"    - STRING CONSTANT
,       - COMMA DELIMITER
a       - IDENTIFIER
)       - CLOSING BRACKETS
;       - SEMICOLON DELIMITER
a       - IDENTIFIER
++      - OPERATOR
;       - SEMICOLON DELIMITER
int     - KEYWORD
b       - IDENTIFIER
=       - OPERATOR
4       - NUMBER CONSTANT
;       - SEMICOLON DELIMITER
int     - KEYWORD
c       - IDENTIFIER
=       - OPERATOR
3       - NUMBER CONSTANT
;       - SEMICOLON DELIMITER
int     - KEYWORD
b       - IDENTIFIER
=       - OPERATOR
8       - NUMBER CONSTANT
;       - SEMICOLON DELIMITER
ERR_UNMATCHED_COMMENT at line no. 11
/


 SYMBOL TABLE

 a       IDENTIFIER
 b       IDENTIFIER
 c       IDENTIFIER
 int     KEYWORD
 main    KEYWORD
 printf  IDENTIFIER


 CONSTANT TABLE

 "%d"    STRING CONSTANT
 2       NUMBER CONSTANT
 3       NUMBER CONSTANT
 4       NUMBER CONSTANT
 8       NUMBER CONSTANT
```

Chapter 3: Syntax Parsing

The parser checks for any syntax errors while parsing the program. If an error is found, the parsing stops and the error is reported.

Test Case 1:

Input:

```c
#include<stdio.h>

int main()
{
    int a = 5;
    while(a>0)
    {
        printf("Hello world");
        a--;
    }

    a=4;
    while(a>0)
    {
        printf("%d",a);
        a--;
        int b;
        b= 4;
        while(b>0)
        {
            printf("%d", a*b);
            b--;
        }
    }
}
```

Output:

```
Status: Syntax Parsing Complete
                              SYMBOL TABLE
                               ------------
    SYMBOL |           CLASS |       TYPE |      VALUE |    LINE NO
-------------------------------------------------------------------------------
         a |      Identifier |        int |          0 |          5
         b |      Identifier |        int |          0 |         17
       int |         Keyword |            |            |          3
      main |      Identifier |        int |            |          3
    printf |      Identifier |            |       "%d" |          8
     while |         Keyword |            |            |          6
```

```
                          CONSTANT TABLE
                           --------------
      NAME |            TYPE
-------------------------------------------------------------------------------
"Hello world" | String Constant
      "%d" | String Constant
         0 | Number Constant
         4 | Number Constant
         5 | Number Constant
```

Test Case 2:

Input:

```c
#include<stdio.h>

struct student
{
  int rollNum;
  int marks;
};

int main()
{
 int a = 1, b=0;
 struct student student1;
 student1.rollNum = 1;
 student1.marks = 90;

 if(a >= 1 && a <= 10)
    b++;

 else
       {  b--;
        /* }
}
```

Output:

```
21 syntax error /
Status: Parsing Failed - Invalid
```

Test Case 3:

Input:

```
// Implicit Error that our Language doesn't support

#include<stdio.h>

int main() {
    char @hello;
    @hello = 'c';
}
```

Output:

```
ERROR at line no. 6
@
6 syntax error @
Status: Parsing Failed - Invalid
```

Chapter 4: Semantic Parsing

The semantics of the code will be checked, i.e., type checking, function declarations.

Test Case 1:

Input:

```
#include<stdio.h>

int myfunc(float c, float d)
{

}

void main()
{
    float a,e;
    float b;
    myfunc(e, b);
}
```

Output:

```
Status: Semantic Parsing Complete - Valid
                            SYMBOL TABLE
                             ------------
    SYMBOL |           CLASS |       TYPE |      VALUE |    LINE NO | PARAMS COUNT |
-------------------------------------------------------------------------------------------
         a |      Identifier |      float |            |         10 |          -1 |
         b |      Identifier |      float |            |         11 |          -1 |
         c |      Identifier |      float |            |          3 |          -1 |
         d |      Identifier |      float |            |          3 |          -1 |
         e |      Identifier |      float |            |         10 |          -1 |
       int |         Keyword |            |            |          3 |          -1 |
     float |         Keyword |            |            |          3 |          -1 |
      main |        Function |       void |            |          8 |          -1 |
    myfunc |        Function |        int |            |          3 |           2 |
      void |         Keyword |            |            |          8 |          -1 |


                            CONSTANT TABLE
                             --------------
      NAME |            TYPE
---------------------------------------
```

Test Case 2:

Input:

```
#include<stdio.h>

void main()
{
    int i,n;

    myfunc(i);

}
```

Output:

```
 7 Function not declared (
Status: Parsing Failed - Invalid
```

Test Case 3:

Input:

```
#include<stdio.h>

void main()
{
    int i=3,n=6;
    float a=0.0;
    a = i+n;
}
```

Output:

```
7 Type Mismatch
 ;
Status: Parsing Failed - Invalid
```

Chapter 5: Intermediate Code Generation

An intermediate representation of the code will be generated. We will be using three address code representations for this phase.

Test Case 1:

Input:

```c
#include <stdio.h>

void main()
{
    int i,n;


        do
        {
            printf("hi");
        }while(i<n);
}
```

Output:

```
func begin main
L0:
refparam "hi"
refparam result
call printf, 1
t0 = i < n
IF not t0 GoTo L1
GoTo L0:
L1:
func end

Status: Parsing Complete - Valid
                            SYMBOL TABLE
                             ------------
    SYMBOL |           CLASS |       TYPE |      VALUE |    LINE NO |
-------------------------------------------------------------------------------
         i |      Identifier |        int |            |          5 |
         n |      Identifier |        int |            |          5 |
        do |         Keyword |            |            |          8 |
       int |         Keyword |            |            |          5 |
      main |        Function |       void |            |          3 |
    printf |        Function |            |            |         10 |
     while |         Keyword |            |            |         11 |
      void |         Keyword |            |            |          3 |


                            CONSTANT TABLE
                             --------------
      NAME |            TYPE
-------------------------------------------------------------------------------
      "hi" | String Constant
```

Test Case 2:

Input:

```c
#include <stdio.h>

void main()
{
    int a;
    a = 0;
}
```

Output:

```
func begin main
t0 = 0
a = t0
func end

Status: Parsing Complete - Valid
                            SYMBOL TABLE
                             ------------
    SYMBOL |           CLASS |       TYPE |      VALUE |    LINE NO |
-------------------------------------------------------------------------------
         a |      Identifier |        int |          0 |          5 |
       int |         Keyword |            |            |          5 |
      main |        Function |       void |            |          3 |
      void |         Keyword |            |            |          3 |


                            CONSTANT TABLE
                             -------------
      NAME |            TYPE
-------------------------------------------------------------------------------
         0 | Number Constant
```

Test Case 3:

Input:

```c
#include <stdio.h>
int main()
{
    int 9abi = 10;
}
```

Output:

```
func begin main
ERROR at line no. 4
9
4 syntax error 9
Status: Parsing Failed - Invalid
```

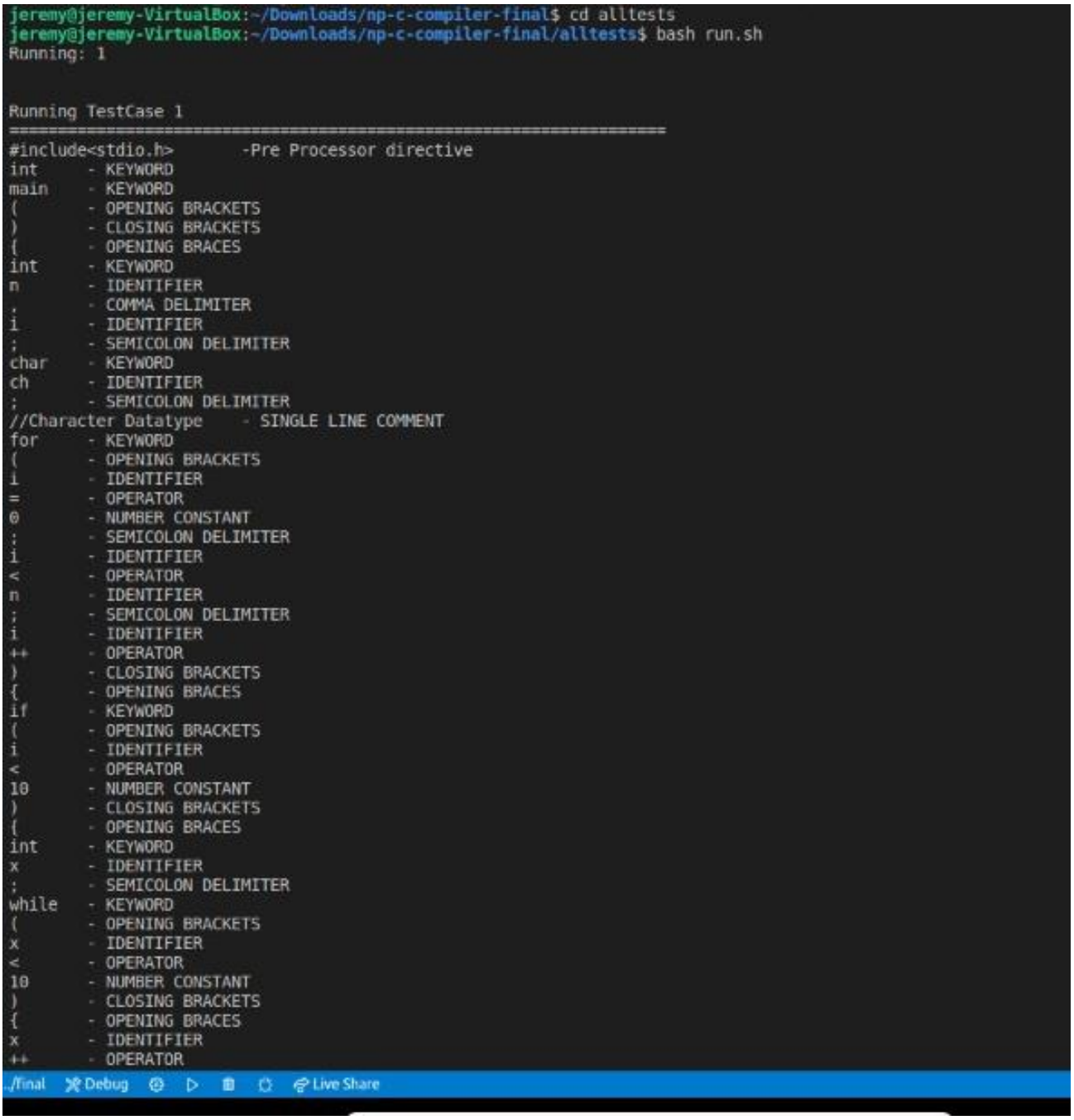
Chapter 6: Final Compilation

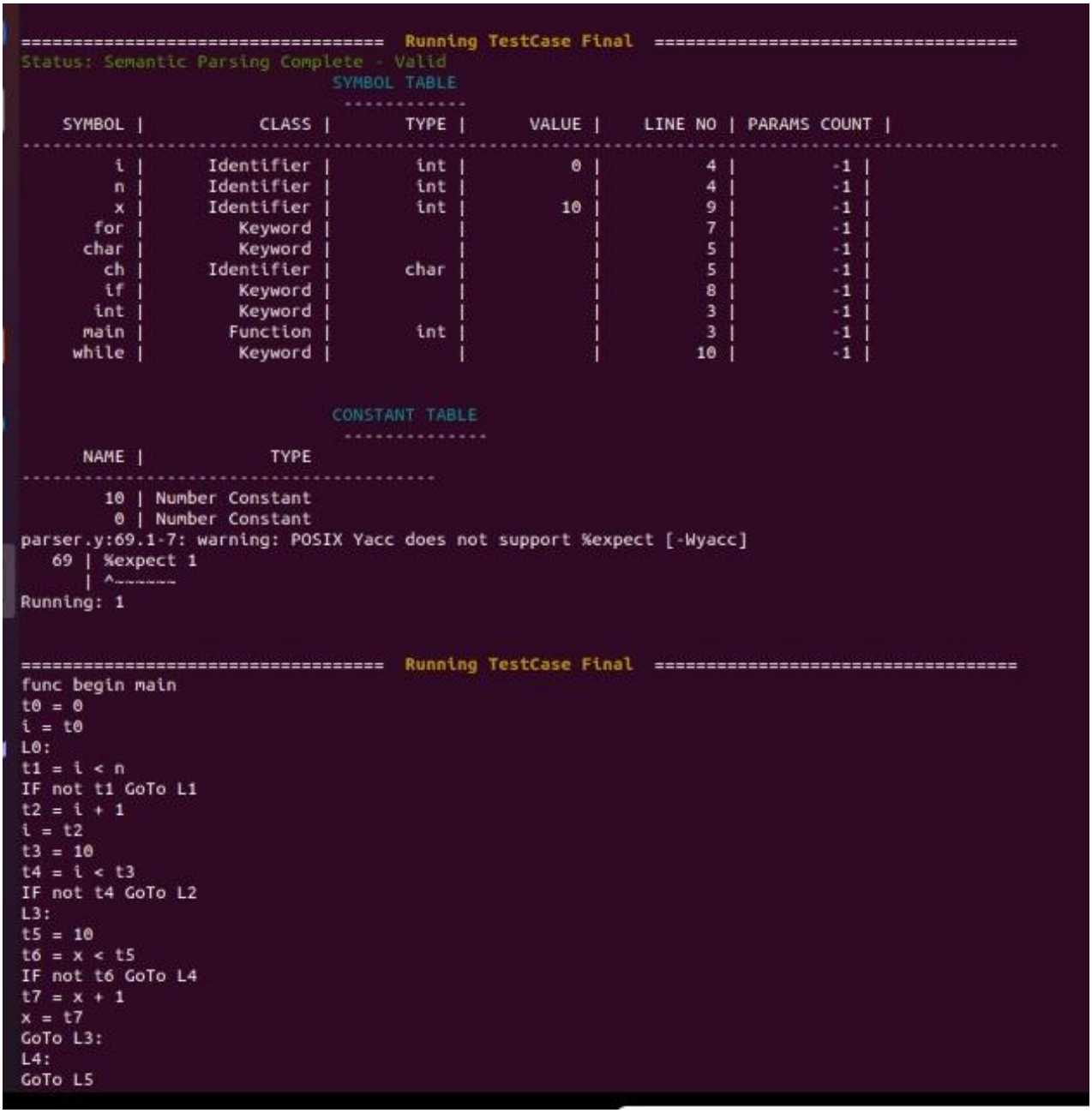
Input Program:

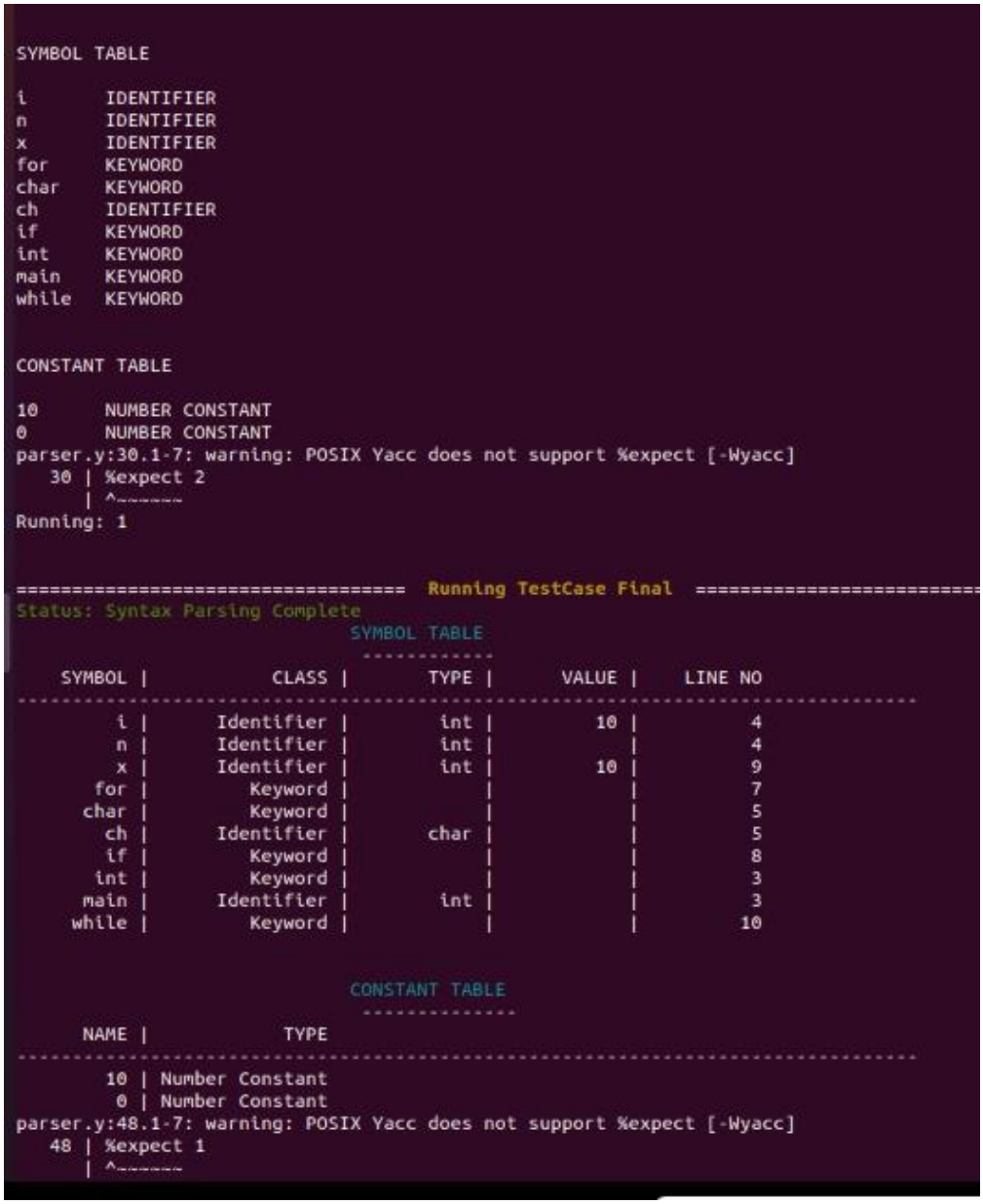
We took the following program to demonstrate the full compilation process along with the outputs of each phase.

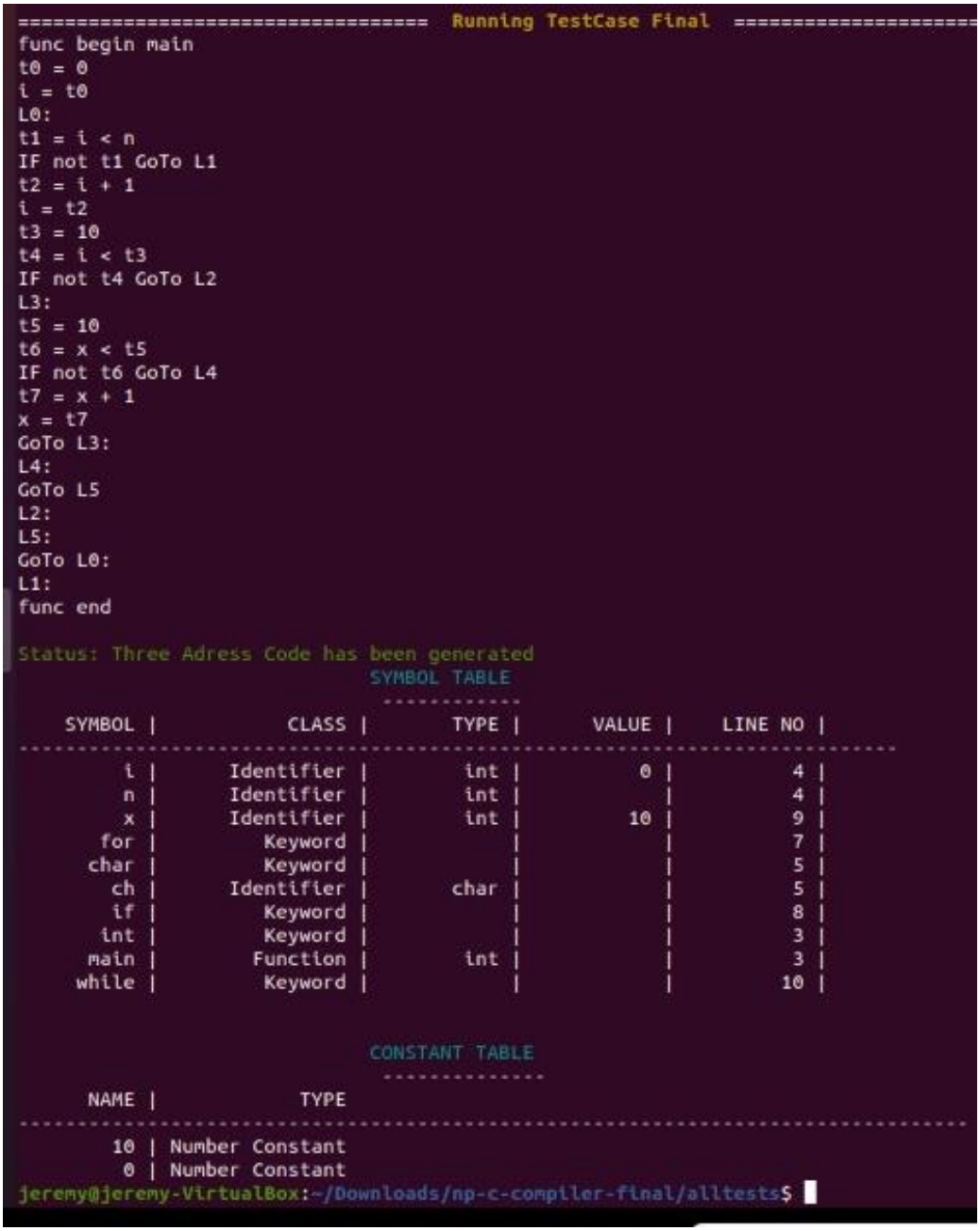
```
#include<stdio.h>

int main(){
    int n, i;
    char ch;//Character Datatype

    for(i=0;i<n;i++){
        if(i<10){
            int x;
            while(x<10){
                x++;
            }
        }
    }
}
```

Lexical Analyzer:




Running TestCase Final
Status: Semantic Parsing Complete - Valid
SYMBOL TABLE
SYMBOL | CLASS | TYPE | VALUE | LINE NO | PARAMS COUNT |
i | Identifier | int | 0 | 4 | -1 |
n | Identifier | int | | 4 | -1 |
x | Identifier | int | 10 | 9 | -1 |
for | Keyword | | | 7 | -1 |
char | Keyword | | | 5 | -1 |
ch | Identifier | char | | 5 | -1 |
if | Keyword | | | 8 | -1 |
int | Keyword | | | 3 | -1 |
main | Function | int | | 3 | -1 |
while | Keyword | | | 10 | -1 |
CONSTANT TABLE
NAME | TYPE
10 | Number Constant
0 | Number Constant
parser.y:69.1-7: warning: POSIX Yacc does not support %expect [-Wyacc]
69 | %expect 1
Running: 1
Running TestCase Final
func begin main
t0 = 0
i = t0
L0:
t1 = i < n
IF not t1 GoTo L1
t2 = i + 1
i = t2
t3 = 10
t4 = i < t3
IF not t4 GoTo L2
L3:
t5 = 10
t6 = x < t5
IF not t6 GoTo L4
t7 = x + 1
x = t7
GoTo L3:
L4:
GoTo L5


SYMBOL TABLE
i IDENTIFIER
n IDENTIFIER
x IDENTIFIER
for KEYWORD
char KEYWORD
ch IDENTIFIER
if KEYWORD
int KEYWORD
main KEYWORD
while KEYWORD
CONSTANT TABLE
10 NUMBER CONSTANT
0 NUMBER CONSTANT
parser.y:30.1-7: warning: POSIX Yacc does not support %expect [-Wyacc]
30 | %expect 2
Running: 1
Running TestCase Final
Status: Syntax Parsing Complete
SYMBOL TABLE
SYMBOL | CLASS | TYPE | VALUE | LINE NO
i | Identifier | int | 10 | 4
n | Identifier | int | | 4
x | Identifier | int | 10 | 9
for | Keyword | | | 7
char | Keyword | | | 5
ch | Identifier | char | | 5
if | Keyword | | | 8
int | Keyword | | | 3
main | Identifier | int | | 3
while | Keyword | | | 10
CONSTANT TABLE
NAME | TYPE
10 | Number Constant
0 | Number Constant
parser.y:48.1-7: warning: POSIX Yacc does not support %expect [-Wyacc]
48 | %expect 1


========== Running TestCase Final ==========
func begin main
t0 = 0
i = t0
L0:
t1 = i < n
IF not t1 GoTo L1
t2 = i + 1
i = t2
t3 = 10
t4 = i < t3
IF not t4 GoTo L2
L3:
t5 = 10
t6 = x < t5
IF not t6 GoTo L4
t7 = x + 1
x = t7
GoTo L3:
L4:
GoTo L5
L2:
L5:
GoTo L0:
L1:
func end
Status: Three Adress Code has been generated
SYMBOL TABLE
SYMBOL | CLASS | TYPE | VALUE | LINE NO |
i | Identifier | int | 0 | 4 |
n | Identifier | int | | 4 |
x | Identifier | int | 10 | 9 |
for | Keyword | | | 7 |
char | Keyword | | | 5 |
ch | Identifier | char | | 5 |
if | Keyword | | | 8 |
int | Keyword | | | 3 |
main | Function | int | | 3 |
while | Keyword | | | 10 |
CONSTANT TABLE
NAME | TYPE
10 | Number Constant
0 | Number Constant
jeremy@jeremy-VirtualBox:~/Downloads/np-c-compiler-final/alltests$