

Compiler Design Lab
Group - 3
Project Report
Language : C

Aashrit S	-	AP19110010361
Madhav	-	AP19110010372
Krishna	-	AP19110010379
Chandra	-	AP19110010438
Syed Yaser	-	AP19110010443
Anirudhan	-	AP19110010451
Aditya	-	AP19110010540

Conceptualization and Design

- Data types available in the language
- Syntax of variable declaration
- One decision making statement
- At least two iterative statements
- CFG for at least five constructs in your language
- Design of Parser
- Semantic Actions
- Syntax of target language

Data types available and syntax of variable declaration

Data Types	Example	Description	Syntax
Character	a B	1 byte The most basic data type in C. It stores a single character.	char varname; char varname1,varname2; char varname = 'a';
Integer	1 -2	2 byte As the name suggests, an int variable is used to store an integer.	int varname; int varname1,varname2; int varname = 0;
Float	10.3	4 bytes It is used to store decimal numbers (numbers with floating point value) with single precision.	float varname; float varname1,varname2; float varname = 3.1;
Double	361.14159 8.25000	8 bytes It is used to store decimal numbers (numbers with floating point value) with double precision.	double varname; double varname1,varname2; double varname = 3.14159;

Decision Making Statements

Decision making statements decide the order of execution of statements based on a particular condition (test condition).

For example, the “**if**” statement executes a set of statements in case the condition is true.

Syntax of If statement in C :	Sample C Code: (If statement)
<pre>If (condition) { Statements }</pre>	<pre>// code to check if a number // is even if (x % 2 == 0) { printf(“ %d is even ”, x); }</pre>

Similarly we have **if-else statement** :

In case the “if” condition is false, statements under the else block are executed.

Syntax of if- else statement in C :	Sample C Code: (If-else statement)
<pre>If (condition) { Statements }else { statements }</pre>	<pre>// prints odd if number is // not even if (x % 2 == 0) { printf(“ %d is even ”, x); } else { printf(“ %d is odd”,x); }</pre>

Iterative Statements

For loop

It is used to repeat a specific block of code a known number of times.

```
int main() {  
    int i;  
    for (i=0; i<10; i++)  
    {  
        printf("%d", i);  
    }  
    return 0;  
}
```

Here **i** is initialized to 0. The text expression **i<10** is evaluated. Since 0 less than 10 evaluates to true, the body of the “for loop” is executed. This will print the **i** value on screen. Now the **i++** update statement is executed. This will print values of **i** on screen. It will update the values till it reaches 10. When **i** value reaches 10, the test condition will be false, and the “for loop” terminates.

While Loop:

A while loop in C programming repeatedly executes a target statement as long as a given condition is true.

The statements may be a single statement or a block of statements. The condition may be any expression, and true is any nonzero value. The loop iterates while the condition is true. When the condition becomes false, the program control passes to the line immediately following the loop.

Execution of While Loop:

1. Control falls into the while loop.
2. The flow jumps to Condition
3. Condition is tested.
 - a. If Condition yields true, the flow goes into the Body.
 - b. If Condition yields false, the flow goes outside the loop

4. The statements inside the body of the loop get executed.
5. Updation takes place.
6. Control flows back to Step 2.
7. The while loop has ended and the flow has gone outside.

Syntax of While loop in C :	Sample C Code: (While loop)
<pre>while (condition) { statement(s); }</pre>	<pre>#include <stdio.h> int main() { // initialize iterator int i = 1; // test expression while (i < 11) { printf("Hello World\n"); // update expression i++; } return 0; }</pre> <hr/> <p>The above program will try to print “Hello World” 10 times.</p>

Phases of Compiler design

- Lexical Analysis
- Syntax Analysis
- Parser Design
- Semantic Analysis
- Code Generation

Lexical Analysis

The input program is broken down into tokens (identifiers, operators, numbers, keywords, punctuators). A symbol table is created with the list of tokens obtained from the input. Lexical Analysis is done using the Flex tool to generate tokens.

Syntax Analysis

CFG for constructs in C Language

<Operator> $\rightarrow + \mid - \mid * \mid / \mid \%$

<Relational Operator> $\rightarrow < \mid > \mid <= \mid >= \mid == \mid !=$

<Logical Operator> $\rightarrow \&\& \mid || \mid !$

<Identifier> $\rightarrow <alpha>A$

$A \rightarrow (<alpha> \mid <digit>) A \mid \epsilon$

<digit> $\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

<num> $\rightarrow <digit> (<num> \mid \epsilon)$

<exp> $\rightarrow (E \mid e) <num>$

<alpha> $\rightarrow a \mid b \mid c \mid d \mid e \mid f \mid g \mid h \mid i \mid j \mid k \mid l \mid m \mid n \mid o \mid$

$p \mid q \mid r \mid s \mid t \mid u \mid v \mid w \mid x \mid y \mid z \mid$

$A \mid B \mid C \mid D \mid E \mid F \mid G \mid H \mid I \mid J \mid K \mid L \mid M \mid N \mid O \mid$

P | Q | R | S | T | U | V | W | X | Y | Z

<Identifiers> → <Identifier> | <Identifier>,<Identifiers>

<Variable Declaration> → <Type><Identifiers>;

<Type> → int | float | char | double

<Statement> → <Identifier>=<Expression>;

<Expression> → <Identifier> | <Number> | <Expression> <Operator> <Expression>

<Condition> → <Identifier> | <Number> | <Condition> <Relational Operator>

<Condition> | <Condition> <Logical Operator> <Condition>

<Statement> → { <Statement List> }

<Statement List> → <Statement> | <Statement><Statement List>

<If Condition> → if (<Condition>)<Statement>

<If-Else Condition> → <If Condition>else<Statement>

Parser Design

The compiler will make use of a Look Ahead Left-to-Right Parser.

The parsing will be done with the help of YACC. It will be using the procedure of a CFG for each Terminal and NT and it will choose the correct expansion rule.

In this step, the tokens generated after lexical analysis are examined to construct syntactically correct sentences and the syntax errors present are printed out.

Example:

Parsing the following code block

```
if (h > 0) {  
    f = 1;  
    return f;  
}
```

will produce the following Abstract Syntax Tree (AST)

Color key



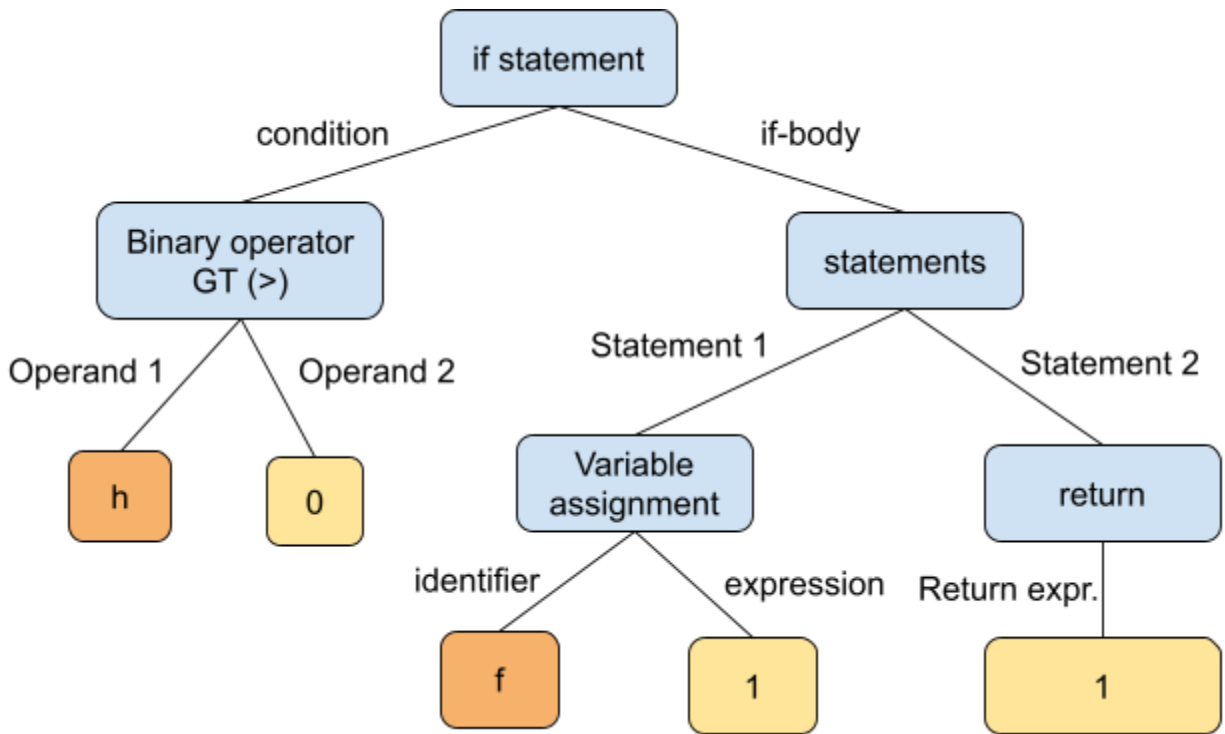
constructs



variable



constant



Semantic Analysis and Actions

The job of ensuring that statements are valid and that their meaning is clear and consistent with the way in which control structures and data types are supposed to be used.

It includes the following features:

- Type checking: check if variables are assigned values of compatible types. Otherwise perform a casting operation as required. Also check for operation on valid data types.
- Label checking: check if further instances of identifiers are already declared. Otherwise throw an error.
- Control-Flow checking: check if control structures are arranged meaningfully. In case of presence of valid but useless conditionals, raise warning. Otherwise throw error.

The parsed output generated in the above step is examined for semantic errors and the abstract syntax tree is printed. Semantic errors like type checking, undeclared variables, multiple declarations of variables are verified.

Syntax of Target Language

INC COUNT ; Increment the memory variable COUNT

MOV TOTAL, 48 ; Transfer the value 48 in the memory variable TOTAL

ADD AH, BH ; Add the content of the BH register into the AH register

AND TEST1, 128 ; Perform AND operation on the variable TEST1 and 128

ADD MARKS, 10 ; Add 10 to the variable MARKS

MOV AL, 10 ; Transfer the value 10 to the AL register

XCHG ; Exchange a byte or word between the specified source and the destination

XCHG AX, BX /* Swap the data in AX and BX */

IN ;Loads the AX register with the contents of the input port specified as the source.

IN AX, Port8

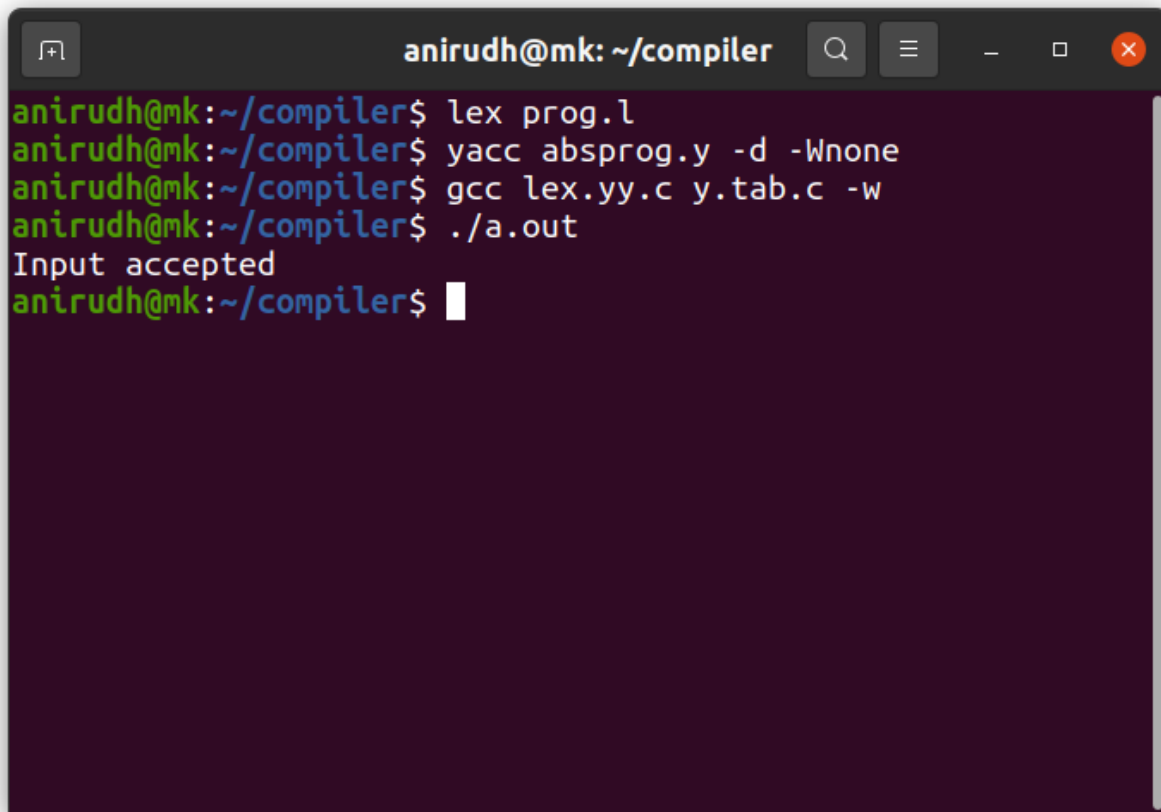
PUSH ;We use it to push word into the stack segment.

PUSH CX

POP ; We use it to pop a word from the stack and store it in the provided location.

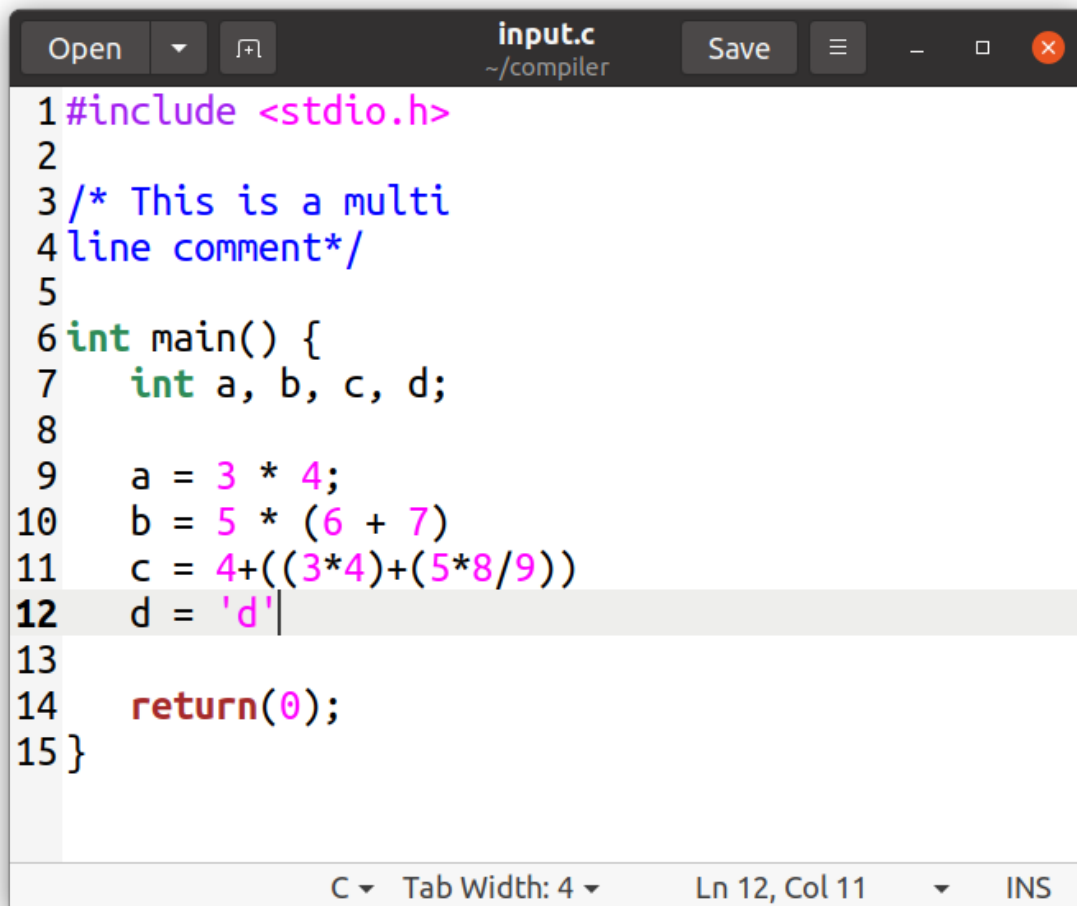
POP CX

USAGE (Instructions to run the code)

A terminal window with a dark purple background and a title bar that reads "anirudh@mk: ~/compiler". The terminal shows a series of commands and their outputs. The commands are: "lex prog.l", "yacc absprog.y -d -Wnone", "gcc lex.yy.c y.tab.c -w", and "./a.out". The output of the last command is "Input accepted". The prompt "anirudh@mk:~/compiler\$" is followed by a white cursor.

```
anirudh@mk:~/compiler$ lex prog.l
anirudh@mk:~/compiler$ yacc absprog.y -d -Wnone
anirudh@mk:~/compiler$ gcc lex.yy.c y.tab.c -w
anirudh@mk:~/compiler$ ./a.out
Input accepted
anirudh@mk:~/compiler$
```

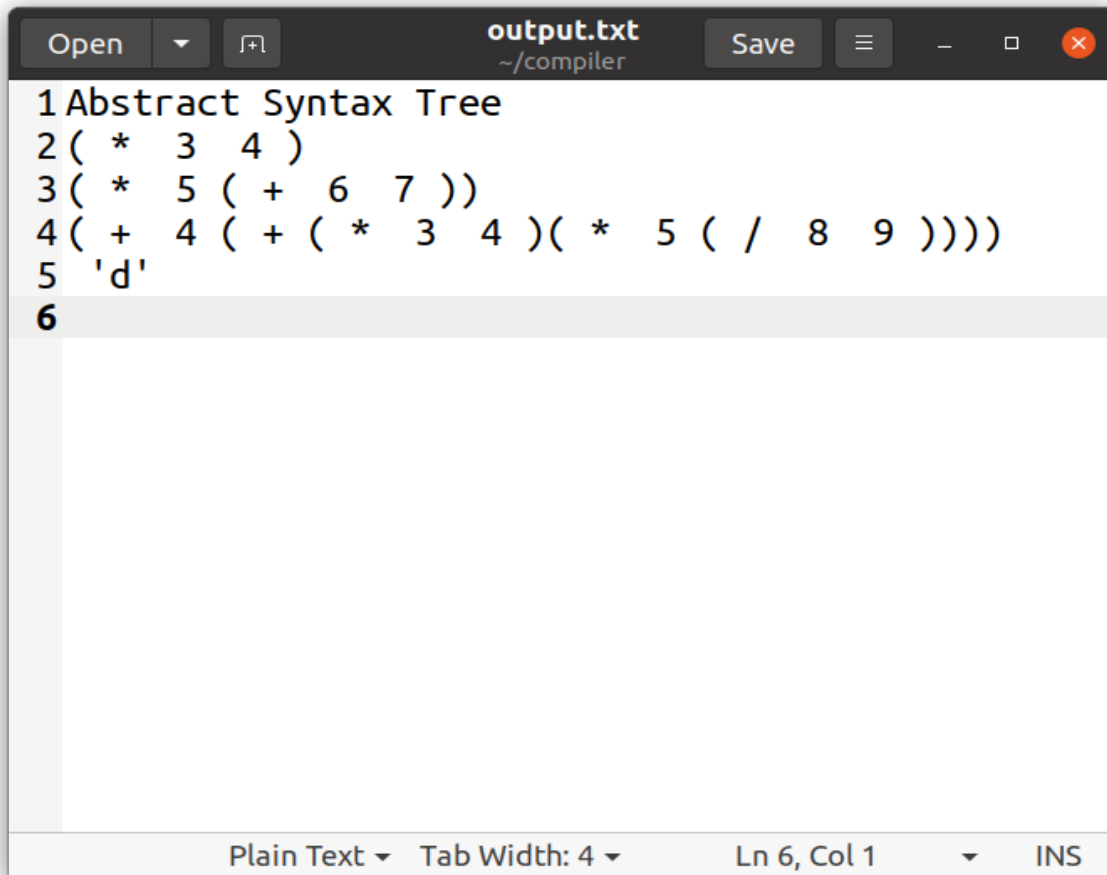
INPUT



```
1 #include <stdio.h>
2
3 /* This is a multi
4 line comment*/
5
6 int main() {
7     int a, b, c, d;
8
9     a = 3 * 4;
10    b = 5 * (6 + 7)
11    c = 4+((3*4)+(5*8/9))
12    d = 'd'|
13
14    return(0);
15 }
```

C ▾ Tab Width: 4 ▾ Ln 12, Col 11 ▾ INS

OUTPUT



```
1 Abstract Syntax Tree
2 ( * 3 4 )
3 ( * 5 ( + 6 7 ) )
4 ( + 4 ( + ( * 3 4 ) ( * 5 ( / 8 9 ) ) ) )
5 'd'
6
```

Plain Text ▾ Tab Width: 4 ▾ Ln 6, Col 1 ▾ INS