

**Don Bosco Institute of Technology, Kulra(W)**

**Department of Computer Engineering**

**CSL601: System Programming and Compiler Construction  
Lab2022-23**

---

<b>Experiment No.:</b>	07
<b>Experiment Title:</b>	Intermediate code Generator  Task:  Implement Intermediate Code Generation and convert infix expression to postfix expression.(Yacc parser)
<b>Student Name</b>	Siddhanth Naidu
<b>Roll No.</b>	43
<b>Objectives :</b>	1) To understand parser generator tool : YACC and Intermediate code generation phase of compiler

<b>Theory /Algorithm :</b>	<p>Intermediate code era may be a stage within the compiler plan handle where the compiler takes the source code and changes over it into an halfway representation that's more appropriate for advance investigation and optimization. The prepare of middle of the road code generation involves several steps, counting lexical examination, language structure examination, and semantic examination.</p> <p>After the AST is generated, the compiler can begin the middle code era stage. This stage includes changing over the AST into a more machine-independent representation, regularly utilizing three-address code or quadruples. The intermediate code era stage includes a few errands, counting expression assessment, control-flow investigation, and code optimization. Expression assessment includes changing over expressions into three-address code or quadruples.</p> <p>Code optimization includes changing the middle of the road code to make strides execution or decrease code size. Once the middle code is produced, it can be assist optimized utilizing strategies such as consistent collapsing, quality reduction, and loop optimization. In conclusion, middle code era is an fundamental stage within the compiler plan prepare. It includes changing over the AST into a more machine-independent representation utilizing three-address code or quadruples. The middle code can at that point be assist optimized to progress execution or diminish code estimate, making it simpler for the compiler to create productive machine code.</p>
--------------------------------	---

--	--

<b>Program Code</b>	<pre> <b>scan.l</b>  %{  #include "y.tab.h" extern  int yylval;  %}  %%  [0-9]+ {yylval=atoi(yytext); return NUMBER;}  \n      return 0; [ \t] ;   *yytext;  .        return  %%   int yywrap(){      return 1;  }  <b>postfix.y</b>      1&gt;  %{  #include &lt;stdio  %}  %token NUMBER </pre>
---------------------	---

	<pre> %left  '+' '-'  %left  '*' '/'  %right t  NEGATIVE  %%  S:      E     {printf("\n");} ; </pre>
--	--

	<pre> E:   E '+' E {printf("+");}            E '*' E {printf("*");}            E '-' E {printf("-");}            E '/' E {printf("/");}            '(' E ')'            '-' E %prec NEGATIVE {printf("-");}            NUMBER    {printf("%d", yylval);}        ;  %%  int main(){ printf("\nEnter infix expression  =&gt; "); yyparse();  }  int yyerror (char *msg) { return printf  ("Error: %s\n", msg);  } </pre>
<b>Input to the Program:</b>	<pre> 1. 1+2 / 7 2. 9 - 4 * 3 /7 + 5 3. 1 2 + 3 4. 12 - 3 5. 1 + + 2 </pre>

<p><b>Output of the program:</b></p>	<pre> [15:34] ❏ Bash 84ms ❏ ~/Projects/Sem6/spcc/exp7 &gt; ./a.out  Enter infix expression =&gt; 1+2 / 7 127/+ [15:35] ❏ Bash 15s 123ms ❏ ~/Projects/Sem6/spcc/exp7 &gt; █  [15:35] ❏ Bash 107ms ❏ ~/Projects/Sem6/spcc/exp7 &gt; ./a.out  Enter infix expression =&gt; 9 - 4 * 3 /7 + 5 943*7/-5+ [15:36] ❏ Bash 37s 867ms ❏ ~/Projects/Sem6/spcc/exp7 &gt; █  [15:36] ❏ Bash 94ms ❏ ~/Projects/Sem6/spcc/exp7 &gt; ./a.out  Enter infix expression =&gt; 1 2 + 3 1 Error: syntax error [15:36] ❏ Bash 6s 442ms ❏ ~/Projects/Sem6/spcc/exp7 &gt; █  [15:37] ❏ Bash 87ms ❏ ~/Projects/Sem6/spcc/exp7 &gt; ./a.out  Enter infix expression =&gt; 12 - 3 123- [15:37] ❏ Bash 3s 923ms ❏ ~/Projects/Sem6/spcc/exp7 &gt; █  [15:38] ❏ Bash 83ms ❏ ~/Projects/Sem6/spcc/exp7 &gt; ./a.out  Enter infix expression =&gt; 1 + + 2 1Error: syntax error [15:39] ❏ Bash 8s 663ms ❏ ~/Projects/Sem6/spcc/exp7 &gt; █ </pre>
<p><b>Outcome of the Experiment:</b></p>	<p>In this experiment these two tools were used to create a compiler to convert arithmetic expressions written in human readable infix form to These expressions should also be valid for any programming language.</p>

	In this experiment these two tools were used to create a compiler to convert arithmetic expressions written in human readable infix form to These expressions should also be valid for any programming language.
<b>Reference:</b>	<a href="#">class notes</a>

---

Course in-charge-Mayura  
Gavhane