

PARSER PROJECT: C++ NESTED FOR LOOPS

MASTER OF COMPUTER APPLICATIONS



**DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF DELHI**

SUBMITTED BY:

ADITYA SHIROMANI

AVLEEN KAUR BAKSHI

SUBMITTED TO:

Dr. ANKIT RAJPAL

ACKNOWLEDGEMENT

We would like to give our sincere thanks to our Compiler Design teacher **Dr Ankit Rajpal**, who guided us throughout our project in every possible way with his valuable advice, useful suggestions and relevant ideas that facilitated the completion of this project. Our project would not have been possible without the proper and rigorous guidance of him and we feel honoured and privileged to work under him. We would also like to thank our friends and classmates for their valuable contributions and insights. Finally, we would like to express our heartfelt appreciation to our family for their unwavering support and encouragement.

CONTENTS

	Name	Page No.
1	Lex File(.l) Code	1
2	Yacc File(.y) Code	2
3	y.output File Link	4
4	GOTO Graph Link	4
5	Steps For Execution	4
6	Syntax of Nested For Loops	5
7	Assumptions	6
8	Test Cases	7

1. Lex File (.l) Code

/* This is a lexer file that generates tokens for a C-like programming language. The code starts with %{ and %}, which are used to enclose C code to be copied verbatim into the generated lexer code. The #include directive is used to include the header file y.tab.h, which contains the definitions of the tokens.

The next section defines some regular expressions and macros that will be used to match the tokens. The regular expression Knostar matches any character that is not an asterisk (*), and Pstar matches one or more asterisks. The regular expression id matches any letter or underscore, and digit matches any digit.

The %{ and %} delimiters are used to start and end the rules section. The rules define the regular expressions for the tokens and the actions to be taken when a token is found. Each rule consists of a regular expression and an action. The regular expression is matched against the input, and when a match is found, the corresponding action is executed. The return statement is used to return the token value associated with the matched regular expression.

The first rule matches whitespace and newline characters and ignores them. The second rule matches multiline comments and ignores them. The third rule matches single-line comments and ignores them. The rest of the rules match the different tokens in the language, such as keywords (for, continue, break, int, etc.), identifiers (sequences of letters, digits, and underscores), comparison operators (<, >, <=, >=, ==, and !=), logical operators (|| and &&), and numbers (sequences of digits). The last rule matches any other character that doesn't match any of the previous regular expressions.

The yywrap() function is used to indicate when the end of the input has been reached. In this case, it always returns 1.

Overall, this code is a lexer that matches regular expressions to tokens in a C-like programming language. It is part of a larger program that includes a parser to generate an abstract syntax tree from the tokens. */

```
%{  
    #include <stdio.h>  
    #include "y.tab.h"          // Including the header file with token definitions  
    //extern int yyval;  
}%
```

```
/* Regular expressions and macros for tokens */  
Knostar [^*]*  
Nostarfwd [^*/]  
Pstar [*]+  
id [A-Za-z_]  
digit [0-9]  
reserved int|float|double|char|long|size_t  
  
/* Ignore whitespace and newlines */  
  
%%  
  
[\\t\\n\\ ] ;  
/*{Knostar}{Pstar}({Nostarfwd}{Knostar}{Pstar})*/" ; //Multiline comments  
/*{Knostar}{Pstar}({Nostarfwd}{Knostar}{Pstar})*/ ; //Single line comments  
for return FOR; /* Token definitions */  
continue return CONTINUE;  
break return BREAK;  
exit return EXIT;  
{reserved} return RESERVED;  
{digit}+ return NUM;  
{id}({id}|{digit})* return ID;  
"<" return LT;  
">" return GT;  
"<=" return LE;  
">=" return GE;  
"==" return EQ;  
"!=" return NE;  
"||" return OR;  
"&&" return AND;  
{digit}+("{++}"|"{--}")|("{++}"|"{--}") {digit}+ printf("lvalue is required as operand");  
. {return yytext[0];}  
%%  
  
int yywrap()  
{  
    return 1;  
}
```

2. YACC File(.y)Code

/* This is a flex and bison (GNU yacc) code for a basic C-like language parser. Here is a brief documentation of the code:

The first part of the code contains C preprocessor directives and declarations, including:

#include <stdio.h> and #include <stdlib.h> to include standard C libraries.
int yylex() and int yyerror() function declarations, which are used to perform lexical analysis and error handling.

extern FILE *yyin and extern FILE *yyout declarations, which are used to read input from a file and write output to a file.

The %token section STARTFORine the tokens used in the language. These tokens are used by the parser to recognize the input. Tokens in this code include ID for identifiers, NUM for numbers, and FOR, LE, GE, EQ, NE, OR, AND, CONTINUE, BREAK,EXIT and RESERVED for keywords.

The %right, %left, and %precedence sections STARTFORine the operator precedence of the language. Operators with higher precedence are parsed first. For example, %left '+' '-' defines that the + and - operators have left associativity, which means that the expression a+b-c is equivalent to (a+b)-c. The %right '=' declaration defines that the = operator has right associativity.

The %% section contains the grammar rules for the language. The grammar includes rules for statements (START), loops (LOOP), definitions (STARTFOR), expressions for initialization (EXPR),expressions for condition (CONDITION) and expressions for increment/decrement (EXPR2).

The yyerror function is called when an error occurs during parsing. It prints an error message and exits the program.

The main function opens the input file specified in the command-line arguments, calls yyparse() to begin parsing, and opens a result file for writing the output.

In summary, this code defines a grammar for a basic C-like language and uses flex and bison to perform lexical analysis and parsing. It is meant to demonstrate how to write a simple parser for a programming language. */

```
%{
// Header section: includes and declarations
#include <stdio.h>
#include <stdlib.h>

// Function declarations
int yylex();
int yyerror();
extern FILE *yyin;
extern FILE *yyout;
}%

// Token definitions
%token ID NUM FOR LE GE EQ NE OR AND CONTINUE BREAK EXIT RESERVED

// Operator precedence and associativity
%right '='
%left OR AND
%left LE GE EQ NE LT GT
%left '+' '-'
%left '*' '/'
%left '!' '%'
%left ','

// Start symbol
%%

START : LOOP {printf("FOR Syntax ACCEPTED \n"); exit(0);}

// Grammar rules
LOOP : FOR '(' EXPR ';' CONDITION ';' EXPR2 ')' STARTFOR
    | FOR '(' ';' ';' ')' STARTFOR
    | FOR '(' EXPR ';' ';' ')' STARTFOR
    | FOR '(' ';' CONDITION ';' ')' STARTFOR
    | FOR '(' ';' ';' EXPR2 ')' STARTFOR
    | FOR '(' ';' CONDITION ';' EXPR2 ')' STARTFOR
    | FOR '(' EXPR ';' ';' EXPR2 ')' STARTFOR
    | FOR '(' EXPR ';' CONDITION ';' ')' STARTFOR
    ;
```

```

STARTFOR : ';'
          | '{' '}'
          | '{' BODY '}'
          | EXPR ';'
          | LOOP
          | ID ID ';'
          ;

BODY      : BODY LOOP
          | BODY EXPR ';'
          | LOOP
          | EXPR ';'
          | BREAK ';'
          | CONTINUE ';'
          | EXIT ';'
          | ID ID ';'
          | ';'
          | BODY BREAK ';'
          | BODY CONTINUE ';'
          | BODY EXIT ';'
          ;

EXPR      : ID '=' EXPR
          | EXPR ',' EXPR
          | RESERVED ID '=' EXPR
          | EXPR '+' EXPR
          | EXPR '-' EXPR
          | EXPR '*' EXPR
          | EXPR '/' EXPR
          | EXPR '%' EXPR
          | ID
          | NUM
          ;

```

```

CONDITION :ID '=' EXPR
          | CONDITION LT CONDITION
          | CONDITION GT CONDITION
          | CONDITION LE CONDITION
          | CONDITION GE CONDITION
          | CONDITION EQ CONDITION
          | CONDITION NE CONDITION
          | CONDITION OR CONDITION
          | CONDITION AND CONDITION
          | NUM
          | CONDITION ',' CONDITION
          | ID
          ;

EXPR2     :ID '=' EXPR
          | EXPR2 ',' EXPR2
          | ID '+' '+'
          | ID '-' '-'
          | '+' '+' ID
          | '-' '-' ID
          ;

%%

// Error handling function
int yyerror(char const *s)
{
    printf("\nyyerror %s\n",s);
    exit(1) ;
}

// Main function
int main(int argc,char **argv) {
    yyin = fopen(argv[argc-1],"r"); // Open input file for reading
    yyparse(); // Call parser function
}

```

3. y.output File Link

https://drive.google.com/file/d/1-fuwihOc2hKkckNSK_o5FRw0ZT7bO3ms/view?usp=share_link

For generating y.output file

```
Microsoft Windows [Version 10.0.22621.1413]
(c) Microsoft Corporation. All rights reserved.

C:\GnuWin32\bin>flex forloop.l

C:\GnuWin32\bin>bison -yvd --report=all forloop.y
```

4. GOTO Graph Link

https://drive.google.com/file/d/1xH8qbKegRTRLhWnFidq16b0Y7PznHAS4/view?usp=share_link

For generating y.dot file

```
C:\GnuWin32\bin>flex forloop.l

C:\GnuWin32\bin>bison -yvd --graph --report=all forloop.y

C:\GnuWin32\bin>
```

5. Steps For Execution

We need to give sample.cpp in order to check for loop's validation

```
C:\GnuWin32\bin>flex forloop.l

C:\GnuWin32\bin>bison -dy forloop.y

C:\GnuWin32\bin>gcc lex.yy.c y.tab.c -o ans

C:\GnuWin32\bin>ans.exe sample.cpp
FOR Syntax ACCEPTED

C:\GnuWin32\bin>
```

6. Syntax of Nested For Loops

Nested loop means a [loop statement](#) inside another loop statement. That is why nested loops are also called as “**loop inside loop**“.

Syntax:

```
for (initialization; condition; increment) {  
    for (initialization; condition; increment) {  
        // statement of inside loop  
    }  
    // statement of outer loop  
}
```

Example:

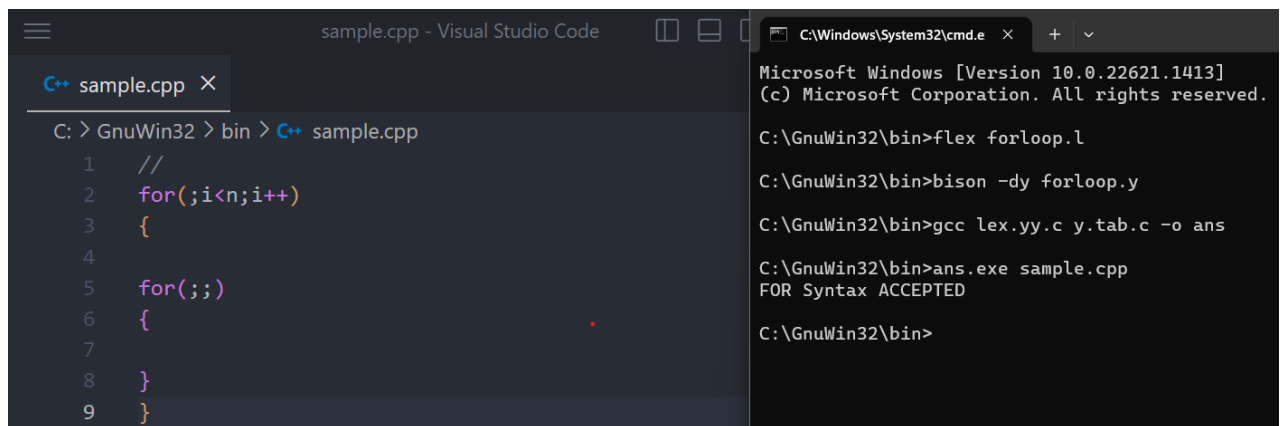
```
// C++ program to display 7 days of 3 weeks  
  
#include <iostream>  
using namespace std;  
  
int main() {  
    int weeks = 3, days_in_week = 7;  
  
    for (int i = 1; i <= weeks; ++i) {  
        cout << "Week: " << i << endl;  
  
        for (int j = 1; j <= days_in_week; ++j) {  
            cout << "    Day:" << j << endl;  
        }  
    }  
  
    return 0;  
}
```


7. Assumptions (DO's & DON'Ts)

- i We ignore everything above for loop and inside its body(just focus on the syntax of for loop).
- ii It will not check for *cout* statements inside the body of for loop.
- iii It will check for initialization and assignment operations in body of for loop.
- iv It would check for keywords such as int, float, double, long, char, string, and size_t reserve words in the initialization part of for loop. No other keyword would be accepted apart from these.
- v It would work for more than one condition in one part of the loop. Example: for(int i=0,j=0;j<10;j++,k++); so this would get accepted.
- vi break, continue and exit are accepted inside body of for loop.
- vii Comments and multiline comments are also accepted inside body of for loop.
- viii Shorthand notations are not handled in this code.
- ix For loop for single statements are accepted without braces.
- x Infinite for loop is accepted.
- xi Loop with either of any argument among three is accepted.
 - a. Example: for(int i=0; ;i++); is accepted
- xii More than 3 semicolons in for loop are not allowed.
- xiii Infinite Nested for loop is accepted.
- xiv Work for single line without braces (it will just search for semicolon).
- xv All valid implementations of nested for loops are accepted.

5. Test Cases

a.)Nested for loop with no Initialization in Outer loop and infinite loop in inner for.



The screenshot shows a Visual Studio Code editor with a file named `sample.cpp` and a Windows Command Prompt window. The code in `sample.cpp` is a nested for loop with no initialization in the outer loop and an infinite loop in the inner loop.

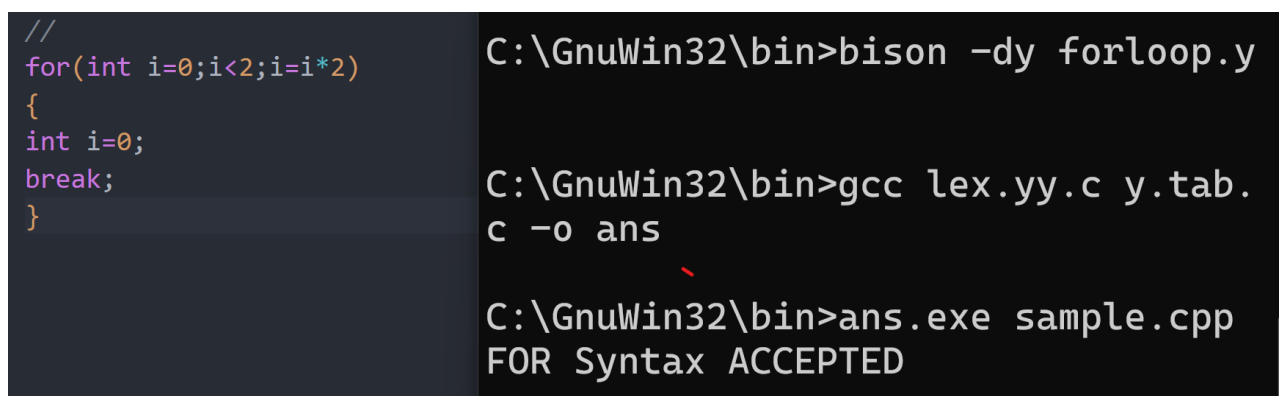
```
1 //
2 for(;;i++)
3 {
4
5     for(;;)
6     {
7
8     }
9 }
```

The Command Prompt shows the following commands and output:

```
C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.22621.1413]
(c) Microsoft Corporation. All rights reserved.

C:\GnuWin32\bin>flex forloop.l
C:\GnuWin32\bin>bison -dy forloop.y
C:\GnuWin32\bin>gcc lex.yy.c y.tab.c -o ans
C:\GnuWin32\bin>ans.exe sample.cpp
FOR Syntax ACCEPTED
C:\GnuWin32\bin>
```

b.)For loop with all 3(initialization,condition and increment/decrement)



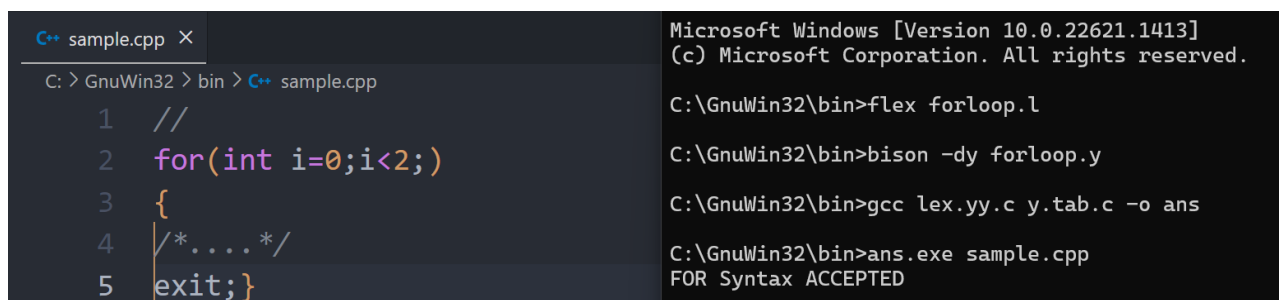
The screenshot shows a Visual Studio Code editor with a file named `sample.cpp` and a Windows Command Prompt window. The code in `sample.cpp` is a for loop with all three components: initialization, condition, and increment/decrement.

```
//
for(int i=0;i<2;i=i*2)
{
int i=0;
break;
}
```

The Command Prompt shows the following commands and output:

```
C:\GnuWin32\bin>bison -dy forloop.y
C:\GnuWin32\bin>gcc lex.yy.c y.tab.c -o ans
C:\GnuWin32\bin>ans.exe sample.cpp
FOR Syntax ACCEPTED
```

c.)for loop with no increment /decrement and using exit inside its body.



The screenshot shows a Visual Studio Code editor with a file named `sample.cpp` and a Windows Command Prompt window. The code in `sample.cpp` is a for loop with no increment/decrement and using `exit` inside its body.

```
1 //
2 for(int i=0;i<2;)
3 {
4     /*...*/
5     exit;}

```

The Command Prompt shows the following commands and output:

```
Microsoft Windows [Version 10.0.22621.1413]
(c) Microsoft Corporation. All rights reserved.

C:\GnuWin32\bin>flex forloop.l
C:\GnuWin32\bin>bison -dy forloop.y
C:\GnuWin32\bin>gcc lex.yy.c y.tab.c -o ans
C:\GnuWin32\bin>ans.exe sample.cpp
FOR Syntax ACCEPTED
```

d.) Infinite for loop with no body

```
C++ sample.cpp X
C: > GnuWin32 > bin > C++ sample.cpp
1 //
2 for(;;);
3

Microsoft Windows [Version 10.0.22621.1413]
(c) Microsoft Corporation. All rights reserved.

C:\GnuWin32\bin>flex forloop.l

C:\GnuWin32\bin>bison -dy forloop.y

C:\GnuWin32\bin>gcc lex.yy.c y.tab.c -o ans

C:\GnuWin32\bin>ans.exe sample.cpp
FOR Syntax ACCEPTED
```

e.) Loop containing multiple statements for initialization and increment/decrement part of for loop.

```
C: > GnuWin32 > bin > C++ sample.cpp
1 //
2 for(int i=0,j=i+1;j<=i;j++,i++)
3 {
4     /*...*/
5 }

C:\GnuWin32\bin>flex forloop.l

C:\GnuWin32\bin>bison -dy forloop.y

C:\GnuWin32\bin>gcc lex.yy.c y.tab.c -o ans

C:\GnuWin32\bin>ans.exe sample.cpp
FOR Syntax ACCEPTED
```

f.) for loop with no initialization and increment/decrement part and having initialization and assignment condition in body of for loop.

```
C: > GnuWin32 > bin > C++ sample.cpp
1 //
2 for(;j<=i;)
3 {int j=0;
4     i=i+1;
5 }

C:\GnuWin32\bin>flex forloop.l

C:\GnuWin32\bin>bison -dy forloop.y

C:\GnuWin32\bin>gcc lex.yy.c y.tab.c -o ans

C:\GnuWin32\bin>ans.exe sample.cpp
FOR Syntax ACCEPTED
```

g.) Checking condition using && operator

```
sample.cpp - Visual Studio Code
C++ sample.cpp X
C: > GnuWin32 > bin > C++ sample.cpp
1 //
2 for(;j<p && p<q;)
3 {int j=0;
4     i=i+1;
5 }

C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.22621.1413]
(c) Microsoft Corporation. All rights reserved.

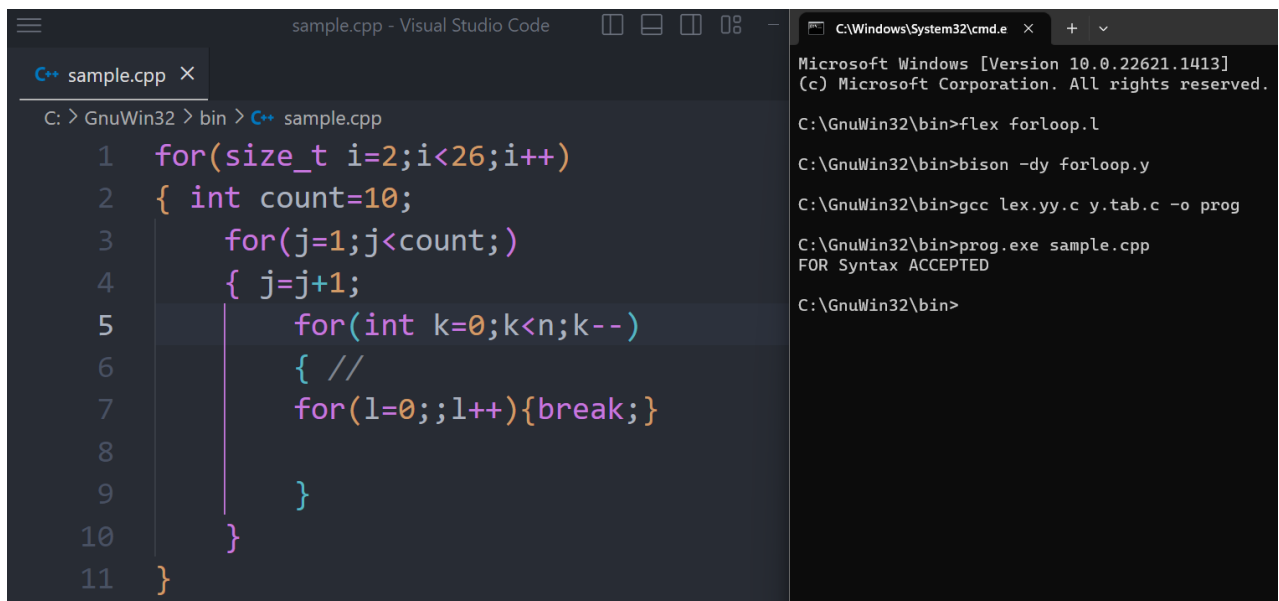
C:\GnuWin32\bin>flex forloop.l

C:\GnuWin32\bin>bison -dy forloop.y

C:\GnuWin32\bin>gcc lex.yy.c y.tab.c -o ans

C:\GnuWin32\bin>ans.exe sample.cpp
FOR Syntax ACCEPTED
```

h.) Nested for loop with all variations



The screenshot shows a Visual Studio Code editor with a file named `sample.cpp` and a Windows Command Prompt window. The code in `sample.cpp` is a nested for loop with three levels of nesting. The first loop iterates `i` from 2 to 25. The second loop iterates `j` from 1 to `count` (which is 10). The third loop iterates `k` from 0 to `n` (which is 10). The code is as follows:

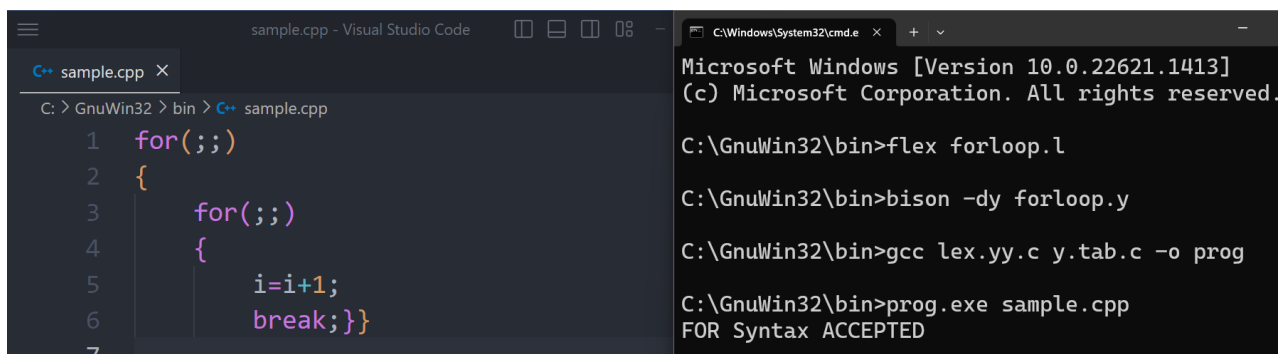
```
1  for(size_t i=2;i<26;i++)
2  { int count=10;
3      for(j=1;j<count;)
4      { j=j+1;
5          for(int k=0;k<n;k--)
6          { //
7              for(l=0;;l++){break;}
8          }
9      }
10 }
11 }
```

The Command Prompt shows the following commands and output:

```
C:\Windows\System32\cmd.e x + v
Microsoft Windows [Version 10.0.22621.1413]
(c) Microsoft Corporation. All rights reserved.

C:\GnuWin32\bin>flex forloop.l
C:\GnuWin32\bin>bison -dy forloop.y
C:\GnuWin32\bin>gcc lex.yy.c y.tab.c -o prog
C:\GnuWin32\bin>prog.exe sample.cpp
FOR Syntax ACCEPTED
C:\GnuWin32\bin>
```

i.) Infinite Nested for loop



The screenshot shows a Visual Studio Code editor with a file named `sample.cpp` and a Windows Command Prompt window. The code in `sample.cpp` is a nested for loop where the first loop is infinite. The code is as follows:

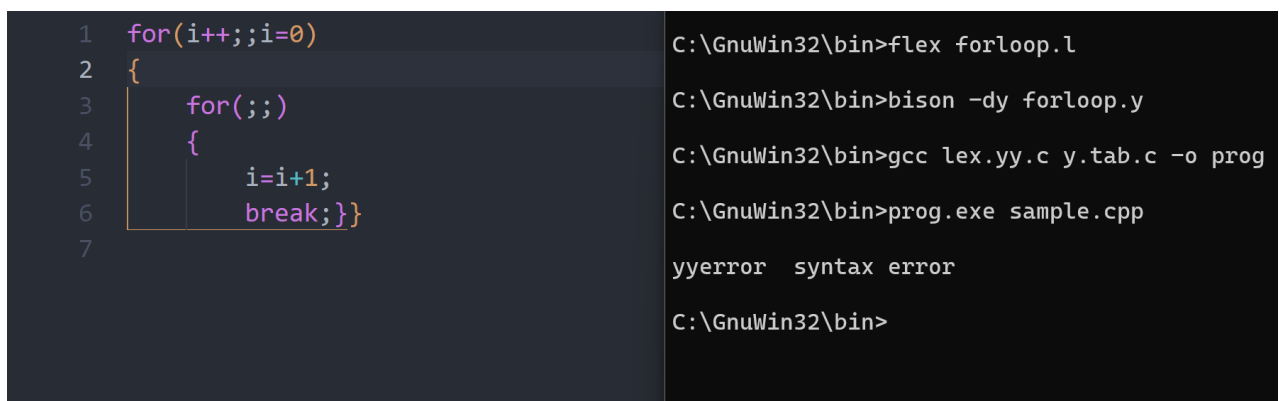
```
1  for(;;)
2  {
3      for(;;)
4      {
5          i=i+1;
6          break;}}
7  }
```

The Command Prompt shows the following commands and output:

```
C:\Windows\System32\cmd.e x + v
Microsoft Windows [Version 10.0.22621.1413]
(c) Microsoft Corporation. All rights reserved.

C:\GnuWin32\bin>flex forloop.l
C:\GnuWin32\bin>bison -dy forloop.y
C:\GnuWin32\bin>gcc lex.yy.c y.tab.c -o prog
C:\GnuWin32\bin>prog.exe sample.cpp
FOR Syntax ACCEPTED
```

j.) Using increment operator in initialization part of the for loop



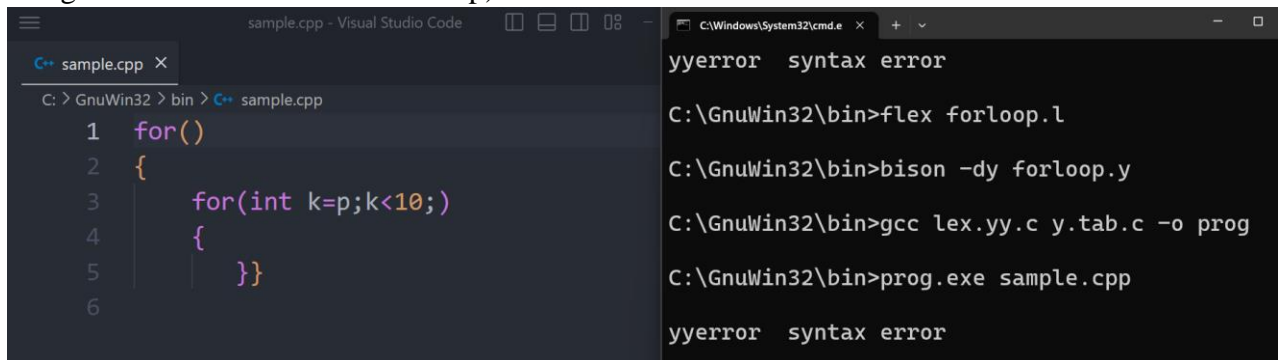
The screenshot shows a Visual Studio Code editor with a file named `sample.cpp` and a Windows Command Prompt window. The code in `sample.cpp` is a nested for loop where the first loop has an increment operator in the initialization part. The code is as follows:

```
1  for(i++;;i=0)
2  {
3      for(;;)
4      {
5          i=i+1;
6          break;}}
7  }
```

The Command Prompt shows the following commands and output:

```
C:\GnuWin32\bin>flex forloop.l
C:\GnuWin32\bin>bison -dy forloop.y
C:\GnuWin32\bin>gcc lex.yy.c y.tab.c -o prog
C:\GnuWin32\bin>prog.exe sample.cpp
yyerror syntax error
C:\GnuWin32\bin>
```

k.) Not defining the initialization, condition and increment/decrement portion of for loop (Not using semicolon inside outer for loop)



The image shows a Visual Studio Code editor window on the left and a Windows command prompt on the right. The editor window displays a C++ file named `sample.cpp` with the following code:

```
1  for()  
2  {  
3      for(int k=p;k<10;)  
4      {  
5      }  
6  }
```

The command prompt window shows the following commands and output:

```
C:\Windows\System32\cmd.e  
yyerror syntax error  
C:\GnuWin32\bin>flex forloop.l  
C:\GnuWin32\bin>bison -dy forloop.y  
C:\GnuWin32\bin>gcc lex.yy.c y.tab.c -o prog  
C:\GnuWin32\bin>prog.exe sample.cpp  
yyerror syntax error
```