Name	Roll No.	Section	DoP
Vidhan Dahatkar	166	B	

Practical 4

Aim –
Develop, Implement and execute a program using YACC/Bison tool to recognize all strings ending with b preceded by n a's using the grammar a n b (note: input n value), also create DFA of given grammar using JFLAP.

Theory -

Deterministic Finite Automaton (DFA)

A **Deterministic Finite Automaton (DFA)** is a finite state machine used to recognize strings that conform to a set of predefined rules. In a DFA, there is a finite number of states, each of which has transitions based on input symbols. The automaton reads the input string one symbol at a time, transitioning between states accordingly. There is a starting state from which the machine begins its computation, and a set of accepting states that define when the input string is accepted. The DFA operates by checking whether a string matches the pattern it has been designed to recognize. For instance, in the context of the grammar "a^n b", where strings consist of n as followed by a b, the DFA would have a state for each a it reads, followed by a final state that processes the b.

JFLAP (Java Formal Languages and Automata Package)

JFLAP (Java Formal Languages and Automata Package) is a software tool that facilitates the design and simulation of automata, such as DFAs and NFAs, as well as grammars like context-free and regular grammars. With JFLAP, users can create DFAs by manually defining states and transitions, and then simulate how the automaton processes various input strings to determine if they are accepted or rejected. In addition to simulating automata, JFLAP can also visualize and simulate the behavior of different types of grammars, including context-free grammars (CFGs). In this scenario, JFLAP serves as a useful tool for visualizing and testing a DFA that recognizes strings of the form a^n b, helping users better understand the structure and behavior of the automaton as it processes input strings.

Program –

parser.y

```
%{
#include <stdio.h>
#include <stdlib.h>
int n;  // Will store the number of 'a's we expect
int count = 0;  // Counts the 'a's we actually see
```

```
void yyerror(const char *s);
int yylex();
%}

%token A B

%%
input:    /* empty */
        | input string '\n'   { count = 0; }  // Reset counter after each line
        ;

string:   pattern   { printf("Valid string: a^%db\n", n); }
        | error     { printf(""); yyerrok; }
        ;

pattern:  a_part B  { if (count != n) yyerror("Incorrect number of 'a's"); }
        ;

a_part:   /* empty */
        | a_part A  { count++; }
        ;
%%

void yyerror(const char *s) {
}

int main() {
   printf("Enter the value of n: ");
   scanf("%d", &n);
   printf("Now enter strings to validate (each ending with newline):\n");
   printf("(Press Ctrl+D to exit)\n");
   return yyparse();
}

/* Simple lexer */
int yylex() {
   int c = getchar();

   if (c == 'a') return A;
   if (c == 'b') return B;
   if (c == '\n') return '\n';
   if (c == EOF) return 0;

   return c;  // Return other characters as-is (will cause parse error)
}
```

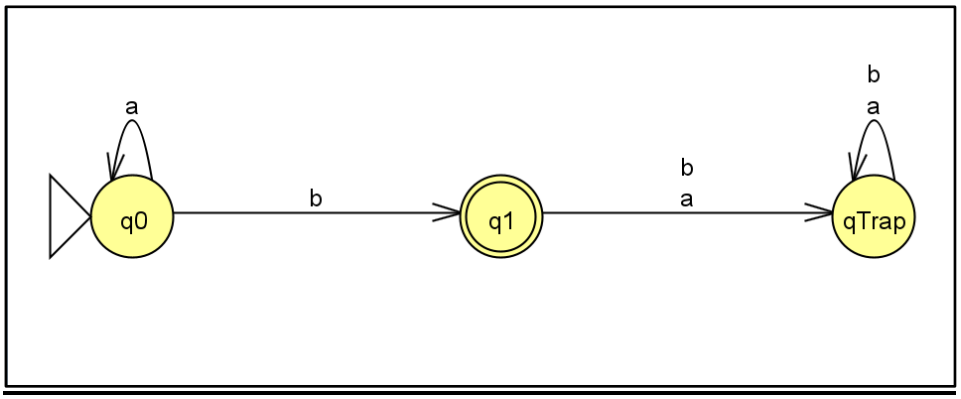
Output -

```
C:\Users\vidha\OneDrive\Desktop\LP Pract codes>bison -d parser.y

C:\Users\vidha\OneDrive\Desktop\LP Pract codes>gcc parser.tab.c -o parser

C:\Users\vidha\OneDrive\Desktop\LP Pract codes>parser
Enter the value of n: 3
Now enter strings to validate (each ending with newline):
(Press Ctrl+D to exit)
aaab
Valid string: a^3b
```

DFA –



Results-

Input	Result
aaab	Accept
aab	Accept
aaaaaab	Accept
aaaab	Accept
aaaaaaaaaaab	Accept
aaa	Reject
aaaaaaabaa	Reject
aaaabba	Reject

Conclusion –

In this practical, we learned how to use Lex and YACC to recognize strings of the form a^nb, where $n \geq 0$. We implemented lexical and syntax rules to validate user input and handle errors. We also designed a DFA in JFLAP to visualize the pattern recognition process. Through this, we understood the working of compiler tools and the construction of automata for regular languages.