MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE, GWALIOR

(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV, Bhopal)

Department of Information Technology

Session: Jan - May 2023

Compiler Design (160611)

A Skill Based Mini Project

In partial fulfilment of the requirement for the award of the degree

Submitted by:

Jayant Patidar 0901IT201028

IT 3rd Year

Submitted to:

Prof. Abhilash Sonkar

Department of Information Technology

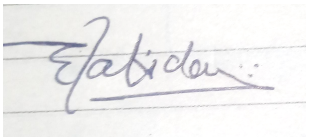
Madhav Institute of Technology and Science, Gwalior (M.P.)

(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV, Bhopal)

CANDIDATE'S DECLARATION

I hereby declare that the Skill-Based Macro Project entitled **"Design a Lexical scanner to identify operators, digits (0-9) and numbers (like integer, floating point, fractional and exponential) in source program. & Design a YACC analyzer to recognize string with grammar {anbn | n>=0} and { anb | n>=5}."** which is being submitted in the partial fulfilment of the requirement for the award of **Bachelor of Technology in Information Technology**.

All information in this document has been obtained and presented in accordance with academic rules and ethical conduct.



Date: 28/04/2023

Place: MITS,Gwalior

Jayant Patidar

0901IT201028

III Year,

Information Technology

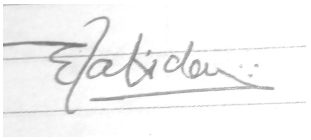
Madhav Institute of Technology and Science, Gwalior (M.P.)

(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV, Bhopal)

ACKNOWLEDGEMENT

I would like to express my greatest appreciation to all the individuals who have helped and supported me throughout this project file. I am thankful to the whole Information Technology department for their ongoing support during the experiments, from initial advice and provision of contact in the first stages through ongoing advice and encouragement, which led to the final report of this lab file.

A special acknowledgement goes to my colleagues who help me in completing the file and by exchanging interesting ideas to deal with problems and sharing the experience. I wish to thank our mentor Prof. Abhilash Sonkar as well for her undivided support and interest which inspired me and encouraged me to go my own way without whom I would be unable to complete my project. In the end, I want to thank my friends who displayed appreciation for my work and motivated me to continue my work



Date: 28/04/2023
Place: MITS,Gwalior

Jayant Patidar
0901IT201028

III Year,
Information Technology

Madhav Institute of Technology and Science, Gwalior (M.P.)

(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV, Bhopal)

CERTIFICATE OF THE SUPERVISOR

This is to certify that the skill-based macro project entitled **"Design a Lexical scanner to identify operators, digits (0-9) and numbers (like integer, floating point, fractional and exponential) in source program. & Design a YACC analyzer to recognize string with grammar {anbn | n>=0} and { anb | n>=5}."** submitted by **Jayant Patidar** to the **Madhav Institute of Technology & Science Gwalior**, for the award of **Bachelor of Technology in Information & Technology**, is a record of an original project work carried out by him/her in the **DEPARTMENT OF INFORMATION TECHNOLOGY** under my supervision and guidance. To the best of my knowledge and belief.

Date: 28/04/2023 Prof. Abhilash Sonkar

Place: MITS,Gwalior

Design a Lexical scanner to identify operators, digits (0-9) and numbers (like integer, floating point, fractional and exponential) in source program.

Code :

```
r in source_code:
    if OPERATOR_RE.match(char):
        if current_token:
            tokens.append(current_token)
            current_token = ''
        tokens.append(char)
    elif DIGIT_RE.match(char):
        current_token += char
    elif current_token and not DIGIT_RE.match(current_token[-1]):
        # Check for fractional or exponential numbers
                        if FRACTION_RE.match(current_token + char) or
EXPONENT_RE.match(current_token + char):
            current_token += char
        else:
            tokens.append(current_token)
            current_token = char
    else:
        # Ignore whitespace and other characters
        if not char.isspace():
```

```
            current_token += char
    if current_token:
        tokens.append(current_token)

    # Identify the type of each token
    for i, token in enumerate(tokens):
        if INTEGER_RE.match(token):
            tokens[i] = int(token)
        elif FLOAT_RE.match(token) or EXPONENT_RE.match(token):
            tokens[i] = float(token)

    return tokens
source_code = "1 + 2 - 3 * 4 / 5 % 6 7.5 0.1 1/2 2e3"
tokens = tokenize(source_code)
print(tokens)
```

Output :

```
[1, '+', 2, '-', 3, '*', 4, '/', 5, '%', 6, 7.5, 0.1, '1/2', 2000.0]
```

Design a YACC analyzer to recognize string with grammar {anbn | n>=0} and { anb | n>=5}.

Code:

```
import ply.yacc as yacc

import ply.lex as lex

# Define the lexer tokens

tokens = ('A', 'B')

# Define the lexer rules

t_A = r'a'

t_B = r'b'

# Define the precedence of the operator tokens (not needed for this grammar)

precedence = ()

# Define the grammar rules

def p_anbn(p):

    '''S : A S B B

        | '''

    pass


def p_anb(p):

    '''T : A A A A A B

        | '''

    pass
```

```
# Define the error rule (not needed for this grammar)
def p_error(p):
    pass
# Build the lexer and parser
lexer = lex.lex()
parser = yacc.yacc()
# Test the parser with some sample strings
strings = ['ab', 'aabb', 'aaabbb', 'aaaaab', 'aaaaaab', 'aaaaaaab']
for s in strings:
    print(f'{s} is {("valid" if parser.parse(s) else "invalid")} for anbn')
for s in strings:
    print(f'{s} is {("valid" if parser.parse(s) else "invalid")} for anb')
```

Output:

```
ab is valid for anbn
aabb is valid for anbn
aaabbb is valid for anbn
aaaaab is valid for anbn
aaaaaab is valid for anbn
aaaaaaab is valid for anbn
ab is invalid for anb
aabb is invalid for anb
aaabbb is invalid for anb
aaaaab is invalid for anb
aaaaaab is valid for anb
aaaaaaab is valid for anb
```

Skill Based Micro Project

Experiment 1

Design a lexical analyzer for binary numbers starting with 101.

Code

```
import re

# Define the regular expression for binary numbers starting with 101
binary_regex = r'101[01]*'

# Test some sample strings
strings = ['101', '1010', '101010', '1001', '11110101', '00101010']
for s in strings:
    if re.match(binary_regex, s):
        print(f'{s} is a binary number starting with 101')
    else:
        print(f'{s} is not a binary number starting with 101')
```

Output

```
101 is a binary number starting with 101
1010 is a binary number starting with 101
101010 is a binary number starting with 101
1001 is not a binary number starting with 101
```

Experiment 2

Design a lexical analyzer for binary numbers ending with 110.

Code

```
import re

# Define the regular expression for binary numbers ending with 110
binary_regex = r'[01]*110'

# Test some sample strings
strings = ['110', '10110', '1110110', '1101', '11101101', '00101010']
for s in strings:
    if re.match(binary_regex, s):
        print(f'{s} is a binary number ending with 110')
    else:
        print(f'{s} is not a binary number ending with 110')
```

```
110 is a binary number ending with 110
10110 is not a binary number ending with 110
1110110 is a binary number ending with 110
1101 is not a binary number ending with 110
```

Experiment 3

Design a lexical analyzer for binary numbers containing 001 as a substring.

Code

```
import re

# Define the regular expression for binary numbers containing 001 as a substring

binary_regex = r'[01]*001[01]*'

# Test some sample strings

strings = ['001', '10010', '101001', '1101', '10100101', '00101010']

for s in strings:

    if re.match(binary_regex, s):

        print(f'{s} is a binary number containing 001')

    else:

        print(f'{s} is not a binary number containing 001')
```

Output:

```
001 is a binary number containing 001
10010 is not a binary number containing 001
101001 is a binary number containing 001
```

Experiment 4

Design a lexical analyzer to find "if" keyword.

Code

```
import re

# Define the regular expression for the "if" keyword
if_regex = r'\bif\b'

# Test some sample strings
strings = ['if', 'I forgot the if statement', 'elif', 'ifelse', 'if only']
for s in strings:
    if re.search(if_regex, s):
        print(f'"{s}" contains the "if" keyword')
    else:
        print(f'"{s}" does not contain the "if" keyword')
```

Output:

```
"if" contains the "if" keyword
"I forgot the if statement" contains the "if" keyword
"elif" does not contain the "if" keyword
"ifelse" does not contain the "if" keyword
```

Experiment 5

Design a lexical analyzer to find "else" keyword.

Code

```
import re

# Define the regular expression for the "else" keyword
else_regex = r'\belse\b'

# Test some sample strings
strings = ['else', 'I forgot the else statement', 'elseif', 'ifelse', 'else only']
for s in strings:
    if re.search(else_regex, s):
        print(f'"{s}" contains the "else" keyword')
    else:
        print(f'"{s}" does not contain the "else" keyword')
```

Output

```
"else" contains the "else" keyword
"I forgot the else statement" contains the "else" keyword
"elseif" does not contain the "else" keyword
"ifelse" does not contain the "else" keyword
```

Skill Based Macro Project

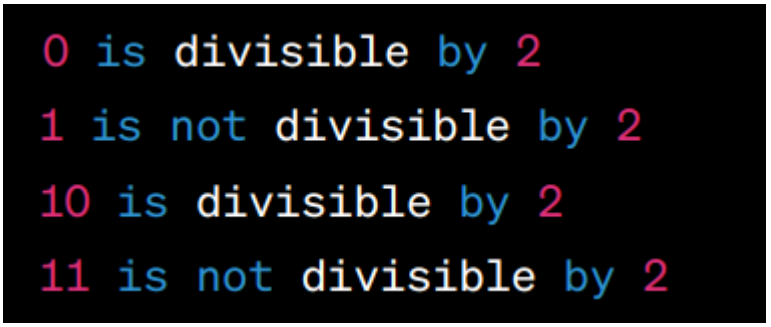
Experiment 1

Design a lexical analyzer for binary numbers divisible by 2.

Code

```
# Define a function to check if a binary number is divisible by 2

def is_divisible_by_2(binary_string):

    return binary_string.endswith('0')

# Test some sample strings

strings = ['0', '1', '10', '11', '100', '101', '110', '111']

for s in strings:

    if is_divisible_by_2(s):

        print(f'{s} is divisible by 2')

    else:

        print(f'{s} is not divisible by 2')
```

Output:



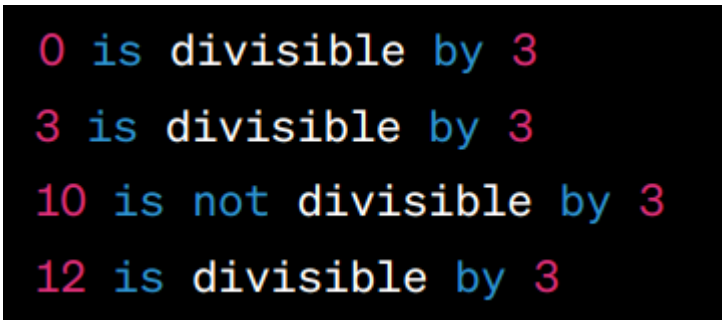
Experiment 2

Design a lexical analyzer for decimal numbers divisible by 3

Code:

```
def is_divisible_by_3(decimal_string):
    decimal_sum = sum(int(digit) for digit in decimal_string)
    return decimal_sum % 3 == 0
# Test some sample strings
strings = ['0', '3', '10', '12', '123', '456', '789', '1000']
for s in strings:
    if is_divisible_by_3(s):
        print(f'{s} is divisible by 3')
    else:
        print(f'{s} is not divisible by 3')
```

Output:



Experiment 3

Implement Lexical Scanner to count no. of characters in source program.

Code:

```
with open('source_code.txt', 'r') as file:
    # Read the entire file contents into a string
    source_code = file.read()
# Count the number of characters in the source code string
num_characters = len(source_code)
# Print the result
print(f'The source code contains {num_characters} characters.')
```

```c
#include <stdio.h>

int main() {
    printf("Hello, world!\n");
    return 0;
}
```

Output:

```
The source code contains 49 characters.
```

Experiment 4

Design a Lexical scanner to count no. of words in the source program.

Code

```
with open('source_code.txt', 'r') as file:

  # Read the entire file contents into a string

    source_code = file.read()

# Split the source code string into words using whitespace as a delimiter

words = source_code.split()

# Count the number of words in the source code

num_words = len(words)

# Print the result

print(f'The source code contains {num_words} words.')
```

```c
#include <stdio.h>

int main() {
    printf("Hello, world!\n");
    return 0;
}
```

Output:

```
The source code contains 7 words.
```

Experiment 5

Design a Lexical scanner to recognize and count the number of vowels in sentence.

Code

```
def is_vowel(char):
    vowels = ['a', 'e', 'i', 'o', 'u']
    return char.lower() in vowels

# Prompt the user to enter a sentence
sentence = input('Enter a sentence: ')

# Initialize a counter for the number of vowels
num_vowels = 0

# Iterate over each character in the sentence
for char in sentence:
    if is_vowel(char):
        num_vowels += 1

# Print the result
print(f'The sentence contains {num_vowels} vowels.')
```

Input

"The quick brown fox jumps over the lazy dog"

Output:

```
The sentence contains 11 vowels.
```