Name – Bhargav Shamuvel Gurav	**PRN** – 2041009
Class – L.Y. B-Tech (Computer)	**Batch** – B1
Course Code – CO406U	**Course Name -** CDL

Practical no. 9

Aim: Write a C program to check whether a string belongs to a grammar or not.

Theory :
To check whether a given string belongs to a grammar or not, you can use a technique called "parsing." Parsing is the process of analyzing the structure of a string to determine if it conforms to a particular grammar. There are several methods and algorithms for parsing, and the choice of which one to use depends on the grammar type.

Here are the general steps to check if a string belongs to a grammar:

1. Define the Grammar:
Start by defining the context-free grammar (CFG) or the formal grammar that describes the language in question. The grammar should include a set of production rules that define how valid strings are formed in the language.

2. Choose a Parsing Technique:
Select an appropriate parsing technique based on the complexity and nature of the grammar. Common parsing techniques include Recursive Descent Parsing, LL Parsing, LR Parsing, and more.

3. Create a Parser:
Implement a parser based on the chosen parsing technique. The parser's job is to take the string and the grammar and determine if the string can be derived from the grammar by applying the production rules.

4. Parsing the String:
Feed the input string to the parser, which will attempt to break down the string into a sequence of tokens and apply the grammar rules. If the parser successfully parses the entire string and reaches an accepting state, the string is considered part of the grammar. If the parser cannot parse the string or encounters an error, the string is not in the language of the grammar.

5. Error Handling:
If the parser encounters an error or cannot continue parsing, it should provide error messages or report that the string does not belong to the grammar.

Here's a simplified example using a simple grammar and a recursive descent parser in Python:

```
# Define a simple grammar for a language that generates strings of the form "a^n b^n" where n is a non-negative integer.
# Grammar:
# S -> 'a' S 'b'
# S -> ε (empty string)

def parse(input_string):
    index = 0

    def match(char):
        nonlocal index
        if index < len(input_string) and input_string[index] == char:
            index += 1
            return True
        return False

    def S():
        if match('a') and S() and match('b'):
            return True
        return True  # ε production

    return S() and index == len(input_string)

# Example usage:
input_string = "aaabbb"
result = parse(input_string)
if result:
    print("String belongs to the grammar.")
else:
    print("String does not belong to the grammar.")
```

In this example, we have a simple grammar for strings of the form "a^n b^n," and the `parse` function uses recursive descent parsing to check if the input string belongs to this grammar.

For more complex grammars, you might need more advanced parsing techniques and possibly parser generator tools like Bison, Yacc, or ANTLR. These tools can generate parsers from a formal grammar specification.

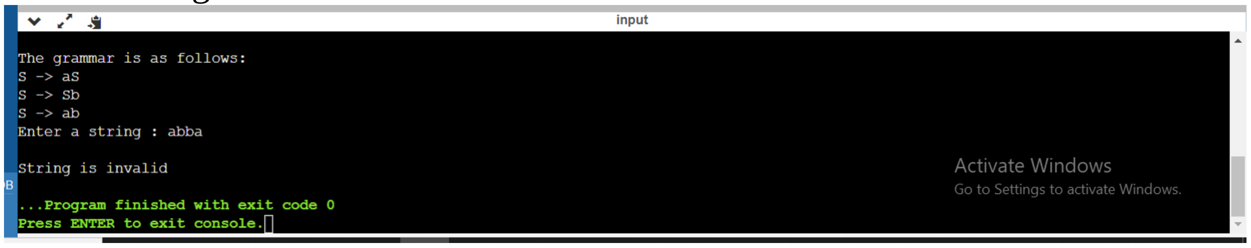
Program Code:

```
#include<stdio.h>
void main()
{
   char str[10];
   int n=1;
   printf("\nThe grammar is as follows:\nS -> aS\nS -> Sb\nS -> ab\n");
   printf("Enter a string : ");
   scanf("%s", str);
   if(str[0]!='a')
        {       printf("\nString is invalid because of incorrect first character");
                exit(0);

        }
        while(str[n]=='a')
           n++;
        if ( str[n] != 'b')
              {printf("\nString is invalid");
                exit(0);
              }
        n++;
        while (str[n]=='b')
              n++;
        if (str [n] != '\0')
        {       printf("\nString is invalid");
                exit(0);
        }
        printf("\nstring is valid");
        getch();
}
```

Output:

Invalid String - abba

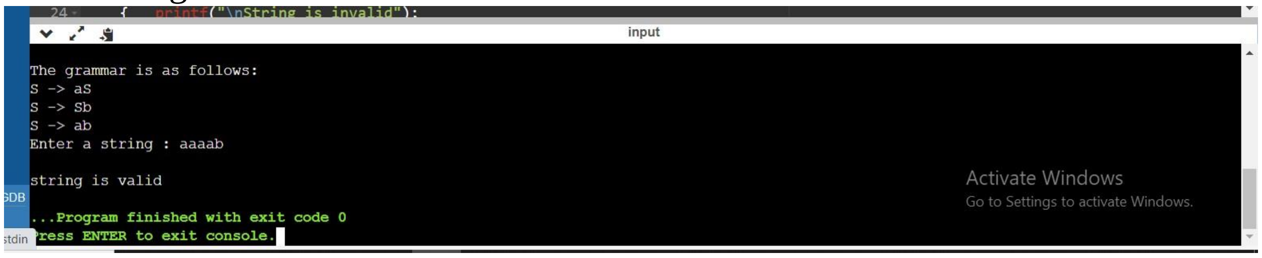


```
The grammar is as follows:
S -> aS
S -> Sb
S -> ab
Enter a string : abba

String is invalid

...Program finished with exit code 0
Press ENTER to exit console.
```

Valid String – aaaab

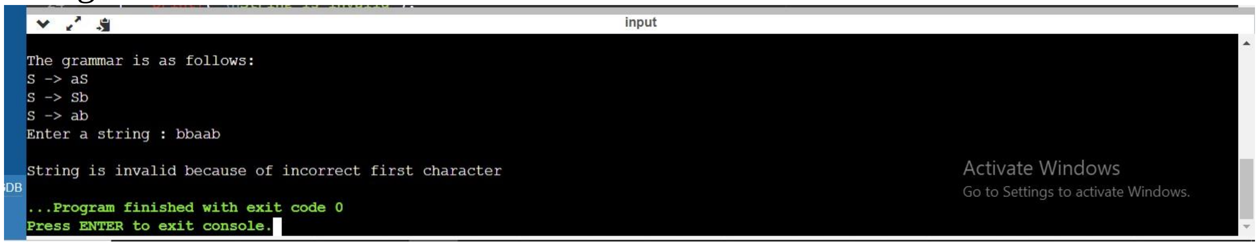


```
The grammar is as follows:
S -> aS
S -> Sb
S -> ab
Enter a string : aaaab

string is valid

...Program finished with exit code 0
Press ENTER to exit console.
```

String invalid due to first character – bbaab



```
The grammar is as follows:
S -> aS
S -> Sb
S -> ab
Enter a string : bbaab

String is invalid because of incorrect first character

...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion : In this practical we learnt how the strings are determined if they belong the particular grammar or not.