FORMAN CHRISTIAN COLLEGE
(A CHARTERED UNIVERSITY)

Compiler Construction - COMP 451 A

Spring 22

Lab 07

Muhammad Sameed Gilani - 231488347

INTRODUCTION:

- **char *fgets(char *str, int n, FILE *stream) → <stdio.h>**
  - str is where the characters will be stored
  - n is the max number of characters to be read
  - File is the pointer to the stream from which the characters will be read

  In the code we use fgets() to store the users input from the terminal into a character array.

  These 3 functions are defined for each production.

- int S();
- int X();
- int Z();

LOGIC:

- In main we first store the input string from the terminal in the char array, expr.
- If S returns 0, the string is simply invalid. If it returns 1 to main and the current char is ‘$’, I.e the string is exhausted. The string is then valid.
- Immediately entering S(); if the first char is ’r’. We increment int count; which we use to iterate the char array. Then we go to X(); if it returns 1 we increment count and check the next char to be ‘d’. Then we return 1 to main. If X(); returns 0, we decrement count back to its original value before entering X(); We now go to Z(). If Z() returns 1 and the next char is ’d’, we return 1 to main, otherwise 0. Meaning the string is invalid.
- In X(); we check the chars according to the productions of X. If the 2 chars are ‘o’ and ‘a’ or they are ‘e’ and ‘a’. we return 1 to S(). Otherwise we decrement count and return 0 to S().
- In Z() we check the chars according to the productions of Z. If the first 2 chars are ‘a’ and ‘I’, we return 1 to S(). Otherwise we decrement count and return 0 to S().
- If X() or Z() follow the productions and return 1 to S(). S () will check the last character to be ‘d’. Then count is incremented to move on to ‘$’. Then we return 1 to main(). If the final char is ‘$’ the string is valid. Otherwise it is invalid.

CODE

```
#include <stdio.h>
#include <string.h>
int S();
int X();
int Z();
char expr[100];
int count,l,countRestore;

int main()
{
    count = 0;
    printf("\nRecursive descent parsing for the following grammar\n");
    printf("\nE->iE'\nE'->+iE'| @\n");
    printf("\nEnter the string to be checked:");

    fgets(expr,100,stdin);
    if(S())
    {
        //count++;
        if(expr[count]=='$'){
            printf("%c",expr[count]);
            printf("\nString is accepted");
        }
        else{
            //printf("%c",expr[count-2]);
            printf("\nString is not accepted");

        }
```

```
}
else{
    printf("%c",expr[count]);

    printf("\nString not accepted");
}
return 0;
}



int S(){

    if(expr[count] == 'r')
    {
     count++;
     //printf("This is X %d\n",X());
     //printf("THIS IS Z %d",Z());
    int valX = X();
    int valZ = Z();
    //printf("%d",valX);
    //printf("%d",valZ);

     if(valX||valZ){
        count++;
        if(expr[count] == 'd'){
            count++;
            return 1;
        }
        else{
```

```
            return 0;
        }
     }


    }

}

int X(){
    //count++;
    if(expr[count] == 'o'){
        count++;
        if(expr[count] == 'a'){
            return 1;
        }
        else{
            count--;
            return 0;
        }
    }

    if(expr[count] == 'e'){
        count++;
        if(expr[count] == 'a'){
            return 1;
        }
        else{
            count--;
```

```
            return 0;
        }
    }

}

int Z(){
    //count++;
    //printf("after Z   %c",expr[count]);

    if(expr[count] == 'a'){
        count++;
        if(expr[count] == 'i'){
            return 1;
        }
        else{
            count --;
            return 0;
        }
    }
    else{
        return 0;
    }
}
```

Sample Outputs: