CPSC 323, Assignment No.2 Name: Richard Gresham

(CFG of some statements in C++) Name: Sean McCarthy

1. Consider the following grammar:

**S🡪aSbB | A | c**

**A🡪cA | c**

**B🡪 d | A**

Trace the grammar to determine which of the following words are accepted or rejected?

1. accbc (use parsing tree)

ii. acccdd (use left-most-derivation)

2. Given the following CFG:

S🡪 I = E

E🡪E+T | E-T | T

T🡪T\*F | T/F | F

F🡪( E ) | I

I 🡪 a | b

Use parsing tree to trace the grammar and decide which of the following statements are accepted

or rejected

1. a=a\*( b- a\*a)

ii. b=a\*b – b\*(a + b)

1. Find the language of the following grammars:

|  |  |
| --- | --- |
| * 1. S🡪aS | bB | aA |λ   B🡪 bB | aA  A🡪aA | bA | λ | * 1. S🡪aS | bA |λ   A🡪aA | bX |λ  X🡪aX | bX |λ |

a.

b.

1. Find a CFG for each of the following languages

1. L=a\* + b\*
2. L=a\*b\*c\*
3. L=ab\* + ba\*+c
4. Find a CFG for



|  |  |
| --- | --- |
| 1. ***Switch statement*** in C++   switch ( <condition> )  {  case v1: <statements> break;  case v2: <statements> break;  ……  default : <statements> break;  }  Where vi are int or char type | 1. ***Do-while loop*** in C++   do{  <statements>  }while (<condition> ) |

# ------------------------------------------------------------------

#             Group names: Gresham, Richard and McCarthy, Sean

#             Assignment: No. 2

#             Due Date: Febuary 9, 2022

#             Purpose: this program reads an expression in postfix form

#             evaluates the expression and displays its values

# -----------------------------------------------------------------------

def main():

    """This is my stack program"""

    looprun = True

    wordstorage = (

        {}

    )  # saves dictionary so if the user continues, variables he used is saved.

    while looprun:

        # loop of program so it can continue at user request.

        print("please enter in your postfix expression.")

        postfix = input("Enter your expression: ")

        s1 = postfixevaluation(postfix, wordstorage)

        if s1 != False:

            print(s1)

        ask = input("CONTINUE(y/n)? ")

        if ask == "n" or ask == "N":

            looprun = False

def number\_check(value):

    # checks a value to see if it finds a float if not return false.

    try:

        float(value)

        return True

    except ValueError:

        return False

def postfixevaluation(exp, wordstorage):

    # this evaluates a postfix expression and turns it into infix.

    stackops = []

    stackvalues = []

    token = ""

    for i in range(0, len(exp)):

        # This for loops creates a list of tokens from user input requires space between each token.

        if exp[i] == " ":

            # print(token)

            stackops.append(token)

            token = ""

        if exp[i] != " ":

            token += exp[i]

            if exp[i] == "$":

                stackops.append(token)

                token = ""

    while len(stackops) != 0:

        # this while loop checks each token and does the appropriate actions via if else if statements.

        value = stackops.pop(0)

        if number\_check(value):

            # checks if it's a float or int.

            if value.isdigit():

                value = int(value)

                stackvalues.append(value)

            else:

                value = float(value)

                stackvalues.append(value)

        elif value == "+":

            s1 = stackvalues.pop()

            s2 = stackvalues.pop()

            s3 = s2 + s1

            stackvalues.append(s3)

        elif value == "-":

            s1 = stackvalues.pop()

            s2 = stackvalues.pop()

            s3 = s2 - s1

            stackvalues.append(s3)

        elif value == "\*":

            s1 = stackvalues.pop()

            s2 = stackvalues.pop()

            s3 = s2 \* s1

            stackvalues.append(s3)

        elif value == "/":

            s1 = stackvalues.pop()

            s2 = stackvalues.pop()

            s3 = s2 / s1

            print(s3)

            stackvalues.append(s3)

        elif value == "^" or value == "\*\*":

            s1 = stackvalues.pop()

            s2 = stackvalues.pop()

            s3 = s2 \*\* s1

        elif value == "%":

            s1 = stackvalues.pop()

            s2 = stackvalues.pop()

            s3 = s2 % s1

        elif value == "$":

            return stackvalues.pop()

        elif value in wordstorage:

            stackvalues.append(wordstorage[value])

        else:

            # if none of these passes it meant the problem is a variable.

            numerator = input(

                "User please enter in a numeric value for word " + value + ": "

            )

            wordstorage[value] = int(

                numerator

            )  # stores variable and value into dictionary.

            stackvalues.append(wordstorage[value])  # appends the value to the stack.

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Text

Description automatically generated