03-10-2023 (DAY 5)

H.TNO:(2105A4120)

**Siamese method: Approach**

1.Start with an empty 3\*3 grid

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

2.Place the number 1 in the top row and the middle column(row 1,column 2)

|  |  |  |
| --- | --- | --- |
|  | 1 |  |
|  |  |  |
|  |  |  |

3.Increment the number by 1 and move diagonally up and to the right

|  |  |  |
| --- | --- | --- |
|  | 1 |  |
|  |  |  |
|  |  | 2 |

4.continue increasing and moving diagonally, in this case, wrap around to the bottom-left corner and place the number 3

|  |  |  |
| --- | --- | --- |
|  | 1 |  |
| 3 |  |  |
|  |  | 2 |

5.keep incrementing and moving diagonally, in this case, wrap around to the top-left corner and place the number 4

|  |  |  |
| --- | --- | --- |
|  | 1 |  |
| 3 |  |  |
| 4 |  | 2 |

6.place the number 5. Continue incrementing and moving diagonally, wrapping around to the top-right corner

|  |  |  |
| --- | --- | --- |
|  | 1 |  |
| 3 | 5 |  |
| 4 |  | 2 |

7. place the number 6. Continue incrementing and moving diagonally, wrapping around to the bottom-left corner

|  |  |  |
| --- | --- | --- |
|  | 1 | 6 |
| 3 | 5 |  |
| 4 |  | 2 |

8. place the number 7. Continue incrementing and moving diagonally, wrapping around to the top-right corner

|  |  |  |
| --- | --- | --- |
|  | 1 | 6 |
| 3 | 5 | 7 |
| 4 |  | 2 |

9. place the number 8. Continue incrementing and moving diagonally, wrapping around to the bottom-right corner

|  |  |  |
| --- | --- | --- |
| 8 | 1 | 6 |
| 3 | 5 | 7 |
| 4 | 9 | 2 |

1.Write a python program of a magic square

def generateSquare(n):

# 2-D array with all

# slots set to 0

magicSquare = [[0 for x in range(n)]

for y in range(n)]

# initialize position of 1

i = n / 2

j = n - 1

# Fill the square by placing values

num = 1

while num <= (n \* n):

if i == -1 and j == n: # 3rd condition

j = n - 2

i = 0

else:

# next number goes out of

# right side of square

if j == n:

j = 0

# next number goes

# out of upper side

if i < 0:

i = n - 1

if magicSquare[int(i)][int(j)]: # 2nd condition

j = j - 2

i = i + 1

continue

else:

magicSquare[int(i)][int(j)] = num

num = num + 1

j = j + 1

i = i - 1 # 1st condition

# Printing the square

print ("Magic Square for n =", n)

print ("Sum of each row or column",n \* (n \* n + 1) / 2, "\n")

for i in range(0, n):

for j in range(0, n):

print('%2d ' % (magicSquare[i][j]),end = '')

# To display output

# in matrix form

if j == n - 1:

print()

n=int(input("Number of rows of the Magic Square:"))

generateSquare(n)

OUTPUT:

Number of rows of the magic square:3

Magic square for n=3

Sum of each row or column:15.0

2 7 6

9 5 1

4 3 8

**Parenthesis:**

1.C program:

#include<stdio.h>

int main()

{

char expression[50];

int x=0, i=0;

printf("\nEnter an expression");

scanf("%s", expression);

while(expression[i]!= '\0')

{

if(expression[i]=='(')

{

x++;

}

else if(expression[i]==')')

{

x--;

if(x<0)

break;

}

i++;

}

if(x==0)

{

printf("Expression is balanced");

}

else

{

printf("Expression is unbalanced");

}

return 0;

}

OUTPUT:

Enter an expression

(a+{b+c]+d}e)

Expression is balanced

**2.Parenthesis python program:**

def are\_parentheses\_balanced(expression):

stack = []

opening\_brackets = "({["

closing\_brackets = ")}]"

for char in expression:

if char in opening\_brackets:

stack.append(char)

elif char in closing\_brackets:

if not stack:

return False # Unmatched closing bracket

top\_of\_stack = stack.pop()

if opening\_brackets.index(top\_of\_stack) != closing\_brackets.index(char):

return False # Mismatched opening and closing bracket

return not stack # True if the stack is empty (all parentheses matched)

# Example usage:

expression = "{[()()]}"

if are\_parentheses\_balanced(expression):

print("Parentheses are balanced.")

else:

print("Parentheses are not balanced.")

OUTPUT:

Parentheses are balanced

3. **Parenthesis python program:**

def printParenthesis(str, n):

if n > 0:

\_printParenthesis(str, 0, n, 0, 0)

return

def \_printParenthesis(str, pos, n, open, close):

if close == n:

for i in str:

print(i, end="")

print()

return

else:

if open > close:

str[pos] = '}'

\_printParenthesis(str, pos + 1, n, open, close + 1)

if open < n:

str[pos] = '{'

\_printParenthesis(str, pos + 1, n, open + 1, close)

n = 3

string = [""] \* 2 \* n

printParenthesis(string, n)

OUTPUT:

{}{{}}

{{}}{}

{{}{}}

{{{}}}