DS ASSIGNMENT 3

Name-Sumit Sunil Koundanya PRN-2019BTEIT00023 Class-SYIT

Q.1) What is linear data structure? What is stack?

Data structure is a particular way of organizing data in a computer so that it can be used effectively. The structure in which data is arranged in linear sequence is called linear data structure. Data items can be traversed in a single run and elements are accessed or placed in contiguous memory location.

**Stack** :- The stack is a linear data structure which follows a particular order in which operation are performed. The order may be LIFO(Last In First Out) or FILO (First In Last out) .The stack is a sequence of items that are accessible at only one end of sequence.

* LIFO order says that the element which is inserted at the last in the stack will be first one to be removed.
* FILO order says that the element which is inserted at the First in the stack will be Last one to be removed.

Q.2) Explain with example the different operations on stack?

Followings are the three main operations on stack-

* **Push:**Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
* **Pop:** Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.
* **Peek or Top:** Returns top element of stack.
* **isEmpty :**Returns true if stack is empty, else false.

Following examples illustrate the operation on stack-

Push D push E Pop Pop

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Top=C | Top=D | Top=E | Top=D | Top=C |
|  |  | E |  |  |
|  | D | D | D |  |
| C | C | C | C | C |
| B | B | B | B | B |
| A | A | A | A | A |

Q.3) What are the application of the stack ?

Application of stacks:-

1) Reversing Strings- A simple application of stack is reversing strings. Once all the characters of string are pushed onto stack, they are popped one by one. Since the character last pushed in comes out first , subsequent pop operation results in reversal of the string.

2) Balancing Parenthesis- stacks are also used to check whether a given arithmetic expression containing a nested parenthesis is properly parenthesized. For each left parenthesis bracket there is corresponding closing symbol . If not, we can balance it using stack.

3) Infix to Postfix Conversion- To convert infix expression to postfix expression, we will use the stack data structure. By scanning the infix expression from left to right, when we will get any operand, simply add them to the postfix form, and for the operator and parenthesis, add them in the stack maintaining the precedence of them.

4) Infix to Prefix Conversion- At first infix expression is reversed. Note that for reversing the opening and closing parenthesis will also be reversed. so we need to convert opening parenthesis to closing parenthesis and vice versa.

After reversing, the expression is converted to **Postfix**form by using infix to postfix algorithm. After that again the postfix expression is reversed to get the prefix expression.

5) Redo-Undo features at many places like editors, photoshop

6) Forward and Backward features in web browsers

Q.4) Write an algorithm to convert infix expression to postfix expression using stack and trace with example showing the content of stack.

Algorithm: – To convert infix to postfix expression using stack.

Pre: - Formula is infix notation that has been edited to ensure that there not be any syntactical error.

Post: - postfix formula has been formatted as a string.

Let, X is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression Y .

1. Push “(“onto Stack, and add “)” to the end of X.
2. Scan X from left to right and repeat Step 3 to 6 for each element of X until the Stack is empty.
3. If an operand is encountered, add it to Y.
4. If a left parenthesis is encountered, push it onto Stack.
5. If an operator is encountered ,then:
   1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) which has the same precedence as or higher precedence than operator.
   2. Add operator to Stack.  
      [End of If]
6. If a right parenthesis is encountered ,then:
   1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) until a left parenthesis is encountered.
   2. Remove the left Parenthesis.  
      [End of If]  
      [End of If]
7. END.

Example-

Infix Expression- ( (A+B) \* (C-D))

|  |  |  |  |
| --- | --- | --- | --- |
| Sr.No | Expression | Stack | Postfix |
| 1 | ( | ( |  |
| 2 | ( | (( |  |
| 3 | A |  | A |
| 4 | + | ((+ | A |
| 5 | B | ((+ | AB |
| 6 | ) | ( | AB+ |
| 7 | \* | (\* | AB+ |
| 8 | ( | (\*( | AB+ |
| 9 | C | (\*( | AB+C |
| 10 | - | (\*(- | AB+C |
| 11 | D | (\*(- | AB+CD |
| 12 | ) | (\* | AB+CD- |
| 13 | ) |  | AB+CD-\* |

Postfix Expression- AB + CD -\*

Q.5) Write an algorithm to evaluate postfix expression using stack explain with example.

Algorithm postFixEvaluate(expr)

This algorithm evaluates a postfix expression and returns its value.

Pre- a valid expression Post- Postfix value computed.

1. createStack(stack)
2. loop(for each character)
3. if (character is operand)

a. pushStack (stack,character)

1. else

a. popStack (stack,opr2)

b. popStack (stack,opr1)

c. operator=character

d.set value to calculate (opr1,operator,opr2)

e. pushStack(stack,value)

1. end if

3. end loop

4. popStack (stack,result)

5. return(result)

End postFixEvaluate

Example –

Postfix Expression – 6 3 + 5 3 - \* \0

|  |  |  |  |
| --- | --- | --- | --- |
| Sr.No | Value | Remark | Stack |
| 1 | 6 | Operand, push | 6 |
| 2 | 3 | Operand, push | 3  6 |
| 3 | + | Operator, pop top two operand and push computed result in stack | 9 |
| 4 | 5 | Operand, push | 5  9 |
| 5 | 3 | Operand, push | 3  5  9 |
| 6 | - | Operator, pop top two operand and push computed result in stack | 2  9 |
| 7 | \* | Operator, pop top two operand and push computed result in stack and  as we get null character final result stored in stack | **18** |

Q.6) what is backtracking ? Explain with Examples.

Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time (by time, here, is referred to the time elapsed till reaching any level of the search tree). **Backtracking** can be defined as a general algorithmic technique that considers searching every possible combination in order to solve a computational problem.

Example - For example, consider the SudoKo solving Problem, we try filling digits one by one. Whenever we find that current digit cannot lead to a solution, we remove it (backtrack) and try next digit. This is better than naive approach (generating all possible combinations of digits and then trying every combination one by one) as it drops a set of permutations whenever it backtracks. Backtracking is another stack use found in applications such as computer gaming, decision analysis, and expert systems.

Q.7) imagine we have two empty stack of integers s1 and s2 .Draw a picture of each stack after the following operation:

11

9

5

7

3

9

7

5

3

3

5

3

pushStack(s1,3) pushStack(s1,5) pushStack(s1,7) pushStack(s1,9) pushStack(s1,11)

7

5

3

pushStack(s1,13)

13

11

9

7

5

3

Loop not emptyStack(s1) - below operation continue till s1 become empty

1. popStack(s1,x)

2. pushStack(s2,x)

Pop 13 push 13

|  |  |
| --- | --- |
|  |  |
| 11 |  |
| 9 |  |
| 7 |  |
| 5 |  |
| 3 | 13 |

S1 S2

|  |  |
| --- | --- |
|  |  |
|  |  |
| 9 |  |
| 7 |  |
| 5 | 11 |
| 3 | 13 |

Pop 11 push 13

S1 S2

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
| 7 | 9 |
| 5 | 11 |
| 3 | 13 |

Pop 9 push 9

S1 S2

Pop 7 push 7

|  |  |
| --- | --- |
|  |  |
|  |  |
|  | 7 |
|  | 9 |
| 5 | 11 |
| 3 | 13 |

S1 S2

Pop 5 push 5

|  |  |
| --- | --- |
|  |  |
|  | 5 |
|  | 7 |
|  | 9 |
|  | 11 |
| 3 | 13 |

S1 S2

Pop 3 push 3

|  |  |
| --- | --- |
|  | 3 |
|  | 5 |
|  | 7 |
|  | 9 |
|  | 11 |
|  | 13 |

S1 S2

As s1 become empty and all numbers in s1 get copied to s2 in reverse direction we complete the process.

End loop.

Q.8) Using manual transformation, write the following infix expression in their postfix and prefix forms

a) Infix Expression:- **D – B + C**

Postfix Expression:- **DB – C+**

Prefix Expression:- **- D + BC**

b) Infix Expression:- **A \* B + C \* D**

Postfix Expression:- **AB \* CD \* +**

Prefix Expression:- **+ \* AB \* CD**

c) Infix Expression:- **(A+B) \* C – D \* F + C**

Postfix Expression:- **AB + C \* DF \* - C +**

Prefix Expression:- **- \*ABC + \* DFC**

d) Infix Expression:- **(A – 2 \* (B + C) – D \* E ) \* F**

Postfix Expression:- **A2BC + \* - DE \* - F \***

Prefix Expression:- **\* - A - \* 2 + BC \* DEF**

Q.9) Determine the value of the following Postfix expression when variable have the following values: A=2 , B=3, C=4, D=5

Values are inserted from left to right in the below stack (as shown in table) and rightmost value in stack became top of the stack

a. ABCD \* - +

By putting above values the postfix expression become 2 3 4 5 \* -

|  |  |
| --- | --- |
| Value | Stack |
| 2 | 2 |
| 3 | 2 3 |
| 4 | 2 3 4 |
| 5 | 2 3 4 5 | 4 \* 5 = 20 |
| \* | 2 3 20 | 3 – 20 = -17 |
| - | 2 -17 | 2 + (-17) = -15 |
| + | -15 |  | **Result = (-15)** |

Result of expression A B C D \* - + is (-15)

b. DC \* BA + -

By putting above values the postfix expression become 5 4 \* 3 2 + -

|  |  |
| --- | --- |
| Values | Stack |
| 5 | 5 |
| 4 | 5 4 |
| \* | 20 | 5 \*4 =20 |
| 3 | 20 3 |
| 2 | 20 3 2 |
| + | 20 5 | 3 + 2 = 5 | |
| - | 15 | 20 – 5 =15 | | **Result = 15** |

Result of expression D C \* B A + - is 15.