Programming Principle & Algorithm

Class- BCA IIIrd Semester

Dr. Dharm Raj Singh
Assistant Professor, (HOD)
Department of Computer Application
JagatpurP. G. College, Varanasi
Mobile No. 9452070368, 7275887513
Email- dharmrajsingh67@yahoo.com

Outline

1. **MODULE 2**

unit 1 : Stacks

- operations on stack
  - (i) Insert Operation
  - (ii) Remove Operation
  - (iii) Application of stack
- infix, postfix and prefix expressions

unit 2: Queues

- operations on queue
  - (i) Insert Operation
  - (ii) Remove Operation
  - (iii) Application of queue

Stacks : An Overview

- Introduction:

  Stack : an ordered list in which all insertions and deletions are made at one end, called the top of the stack.

- Objective:

  At the end of the Module, students should be able to appreciate,

  1. Introduction to Stack
  2. Why use Stack
  3. Operations Performed in Stack
  4. Push Operation
  5. Pop Operation
  6. Application of Stack

Introduction to Stack

- A stack is used to provide temporary storage space for values. It is defined as a data structure which operates on a first in, last out basis. Its uses a single pointer (index) to keep track of the information in the stack.
- The basic operations associated with a stack are,
  - insert (push) an item onto the stack
  - remove (pop) an item from the stack
- The following diagram shows an empty stack of four locations. It looks just like an array, and it has an index pointer pointing to the beginning of the stack (called the TOP of the stack).

```
+--------+ <------- Stack Pointer
+--------+
+--------+
```

Why use Stacks

- Stacks provide a very efficient storage structure for manipulating data that is accessed according to the LIFO method. Stacks are often used as temporary storage to facilitate the solution of a problem.
- Stacks are restricted linear lists of data where additions and deletions are made only at one end, called the "top" of the stack. The restrictions imposed on stacks allow us to specialize their implementation with arrays or linked lists, in ways that can increase efficiency.

Operations Performed in Stack

- **Pushing items onto the stack**

  The stack pointer is considered to be pointing to a free (empty) slot in the stack. A push operation thus involves copying data into the empty slot of the stack, then adjusting the stack pointer to point to the next free slot.

- **Removing items from the stack**

  To remove an item, first decrement (subtract 1 from) the stack pointer, then extract the data from that position in the stack.

Push Operation

- Initialization

  TOS = -1

  MAXSIZE = 100

- Push Operation

```
Push(x){
    TOS ++;
    If  (TOS > MAXSIZE)
    Display "Stack overflow"
    Else
      Stack(TOS)= x
    End if
}
```

Pop Operation

- Pop Operation

```
Pop()
{
   If (TOS<0)
     Display “Stack underflow”
   Else
     Return (Stack(TOS--))
   End if
}
```

Application of Stacks

- Can you think of some applications for Stacks ?
  - Reversing a string
  - Evaluating an expression
  - Recursion

Stacks : Summary

- A stack is used to provide temporary storage space for values. It is defined as a data structure which operates on a first in, last out basis. Its uses a single pointer (index) to keep track of the information in the stack.
- Stacks provide a very efficient storage structure for manipulating data that is accessed according to the LIFO method. Stacks are often used as temporary storage to facilitate the solution of a problem.

What are infix, postfix and prefix expressions?

- **INFIX:-**

  An infix expression is a single letter, or an operator, proceeded by one infix string and followed by another infix string.
  A
  A + B
  (A + B) + (C – D)
- **PREFIX:-**

  A prefix expression is a single letter, or an operator, followed by two prefix strings. Every prefix string longer than a single variable contains an operator, first operand and second operand

  A
  + A B
  + + A B – C D
- **POSTFIX:-**

  A postfix expression (also called Reverse Polish Notation) is a single letter or an operator, preceded by two postfix strings. Every postfix string longer than a single variable contains first and second operands followed by an operator.

  A
  A B +
  A B + C D –

Algorithm for Postfix to Infix Conversion

- Initialize a string containing postfix expression.
- Create a stack s of type string.
- Traverse from the start to end of the string and check if the current character is an operand push it as a string in the stack.
- Else pop the two top characters from the stack and concatenate them as SECOND CHARACTER + CURRENT OPERATOR + FIRST CHARACTER. Push the string back into the stack.
- Return the top of the stack.

Postfix expression : ABC/-AD/E-*

Expression	Stack
ABC/-AD/E-*	A
BC/-AD/E-*	B, A
C/-AD/E-*	C, B, A
/-AD/E-*	(B/C), A
-AD/E-*	(A-(B/C))
AD/E-*	A, (A-(B/C))
D/E-*	D, A, (A-(B/C))
/E-*	(A/D), (A-(B/C))
E-*	E, (A/D), (A-(B/C))
-*	((A/D)-E), (A-(B/C))
*	((A-(B/C))*((A/D)-E))

Therefore, Infix expression : ((A-(B/C))((A/D)-E))*

Algorithm for Prefix to Postfix Conversion

- Initialize a string containing prefix expression.
- Create a stack s of type string.
- Traverse from the last character to first of the string and check if the current character is an operator pop the two top characters from the stack and concatenate them as a single string with current operator after these both. Push the string back into the stack.
- Else if the current character is not an operator, push it as a string in the stack.
- Return the top of the stack.

Prefix expression : *-A/BC-/ADE

Expression	Stack
*-A/BC-/ADE	E
*-A/BC-/AD	D, E
*-A/BC-/A	A, D, E
*-A/BC-/	AD/, E
*-A/BC-	AD/E-
*-A/BC	C, AD/E-
*-A/B	B, C, AD/E-
*-A/	BC/, AD/E-
*-A	A, BC/, AD/E-
*-	ABC/-, AD/E-
*	ABC/-AD/E-*

*Therefore, Postfix expression : ABC/-AD/E-**

Queue : An Overview

- Introduction:

  Queue : an ordered list in which all insertions take place on one end, the rear of the queue, while all the deletions take place at the other end, the front of the queue.

- Objective:

  At the end of the Module, students should be able to appreciate,

  1. Introduction to Queue
  2. Why use Queue
  3. Operations Performed in Queue
  4. Insert Operation
  5. Remove Operation
  6. Application of queue

Introduction to Queues

- A queue is defined as a data structure which holds a series of items to be processed on a first in first out basis (though some queues can be sorted in priority). The operations associated with a queue are,
- initialize the queue
- insert an item in the queue
- remove an item from the queue
- count the number of empty slots in the queue
- count the number of items in the queue

The following diagram shows an empty queue. It is identified as a series of ten locations,

and two pointers 1 and 10 named front and rear. These two pointers keep track of where the front and rear of the queue is.

```
 1     2   3   4    5   6   7     8  9    10
 +---+---+---+---+---+---+---+---+---+---+
 |    |     |   |   |    |    |   |   |    |   |
+---+---+---+---+---+---+---+---+---+---+
```

Why use Queues

The purpose of a queue is to provide some form of buffering.

Typical uses of queues in computer systems are,

- process management
- buffer between the fast computer and a slow printer

Operations performed in Queue

- First in First out – FIFO
  - Insert () : Insert into a queue
  - Remove () : Remove from a queue
- Overflow
- Underflow
- Initialization
    - Front = 0
    - Rear = -1
    - MAXSIZE =10

Insert Operation

- Example

```
Insert(x)
{
    rear ++;
    if (rear == MAXSIZE)
     display “Queue size reached”
    else
     Queue[rear]=x
    End if
}
```

Remove Operation

- Example

```
int Remove()
{
    if (front > rear)
      display “Queue is empty”
      return null
    else
      return (front ++)
}
```

Application of Queues

- Some applications of Queues are,
  - Cache Memory Management schemes
    - FIFO, LRU, etc
  - In any real life situation like queues in counters, airport flight takes off!!

Queues : Summary

- A queue is defined as a data structure which holds a series of items to be processed on a first in first out basis (though some queues can be sorted in priority).
- Typical uses of queues in computer systems are,
  - **process management**
  - **buffer between the fast computer and a slow printer**

Exercise

1. What do you understand by stack overflow and underflow?
2. Differentiate between an array and a stack.
3. How does a stack implemented using a linked list differ from a stack implemented using an array?
4. Differentiate between peek() and pop() functions.
5. Why are parentheses not required in postfix/prefix expressions?
6. What is a priority queue? Give its applications.
7. Explain the concept of a circular queue? How is it better than a linear queue?
8. Why do we use multiple queues?
9. Draw the queue structure in each case when the following operations are performed on an empty queue.

   (a) Add A, B, C, D, E, F

   (b) Delete two letters

   (c) Add G   (d) Add H

   (e) Delete four letters   (f) Add I
10. Write a program to implement a priority queue.
11. 9. Write a program to create a queue from a stack.
12. 10. Write a program to create a stack from a queue.

References

- Patel, Mayank. *Data Structure and Algorithm With C*. Educreation Publishing, 2018.
- E.Horowitz and S.Sahani, “Fundamentals of Data structures”, Galgotia Book source Pvt. Ltd., 2003.
- R.S.Salaria, “Data Structures & Algorithms”, Khanna Book Publishing Co. (P) Ltd.,2002.
- Y.Langsam et. Al., “Data Structures using C and C++”, PHI, 1999.
- Bergin, Joseph A. *Data Abstraction: The Object-Oriented Approach Using C++/Book and Disk*. McGraw-Hill, Inc., 1994.
- Samet, Hanan. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006.

Declaration

Dr. Dharm Raj Singh
Assistant Professor, (HOD)
Department of Computer Application
JagatpurP. G. College, Varanasi

Thanks