Module-3: Data Structures

Dr. P. Gayathri
Associate Professor
SCOPE, VIT University

Abstract Data Type (ADT)

- ADT = object + operations
- An **ADT** describes an object together with a set of operations
- Example: Integer
- Operations – addition, subtraction, multiplication, division and modulus
- There is no rule telling us which operations must be supported for each ADT.
- This is a decision made for the given problem.

Example

- Example - Abstract stack
- Could be defined by three operations:
  - push, that inserts some data item onto the structure
  - pop, that extracts an item from it (with the constraint that each pop always returns the most recently pushed item)
  - peek, that allows data on top of the structure to be examined without removal.

Typical ADTs:

- Lists
- Stacks
- Queues
- Trees
- Heaps
- Graphs

Stacks & Queues

Stacks:	LIFO Last In First Out
Queues:	FIFO First In First Out

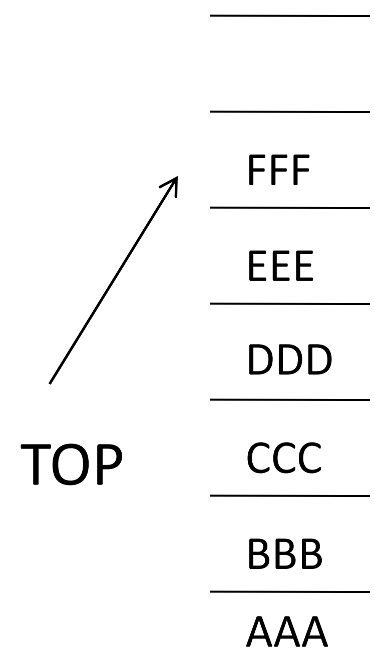
STACKS - LIFO

- A stack is a linear structure in which items are added or removed only at one end.
- Three everyday examples of such a structure
  - Stack of plates
  - Stack of papers
  - Stack of folded towels
- In particular the last item to be added to stack is the first item to be removed

Operations On Stack

- PUSH: is the term to insert an element into a stack
- POP: is the term to delete an element from a stack

- Example: Suppose the following 6 elements are pushed in order onto an empty stack
- AAA, BBB, CCC, DDD, EEE, FFF
- This means:
  - EEE cannot be deleted before FFF is deleted,
  - DDD cannot be deleted before EEE and FFF is deleted and so on.



- **PUSH (ITEM)**

  1. If TOP == MAX, then Print OVERFLOW
  2. Else Set TOP = TOP + 1
  3. Set STACK [TOP] = ITEM.

- **POP ()**

  1. If TOP == 0, then Print UNDERFLOW
  2. Else Set ITEM = STACK[TOP]
  3. Set TOP = TOP - 1
  4. Print ITEM

Stack applications

- Expression evaluation
- Balancing parenthesis / symbols

Arithmetic Expressions

- Precedence Level
  - Highest — Exponentiation ($\uparrow$)
  - Next Highest — Multiplication (*) and Division (/)
  - Lowest — Addition (+) and subtraction (-)

- Infix Notation

  A + B
- Prefix Notation

  + AB
- Postfix or Suffix Notation (Reverse Polish Notation)

  AB +

Evaluation of Expression

- The Computer Usually Evaluates an Arithmetic expression written in infix notation into steps

1. First converts the expression to postfix notation
2. Evaluates the postfix expression

- Stack is the Main Tool that is Used to Accomplish given Task.

Q: A + (B * C - (D / E ↑ F) * G) * H

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Symbol	STACK	Expression P
(1) A		A
(2) +	+	A
(3) (	+ (	A
(4) B	+ (	A B
(5) *	+ (*	A B
(6) C	+ (*	A B C
(7) -	+ (-	A B C *
(8) (	+ (- (	A B C *
(9) D	+ (- (	A B C * D
(10) /	+ (- (/	A B C * D
(11) E	+ (- (/	A B C * D E

Q: A + (B * C - (D / E $\uparrow$ F) * G) * H

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Symbol	**STACK**	**Expression P**
(12) $\uparrow$	+ (- (/ $\uparrow$	A B C * D E
(13) F	+ (- (/ $\uparrow$	A B C * D E F
(14))	+ (-	A B C * D E F $\uparrow$ /
(15) *	+ (- *	A B C * D E F $\uparrow$ /
(16) G	+ (- *	A B C * D E F $\uparrow$ / G
(17))	+	A B C * D E F $\uparrow$ / G * -
(18) *	+ *	A B C * D E F $\uparrow$ / G * -
(19) H	+ *	A B C * D E F $\uparrow$ / G * - H
(20)		A B C * D E F $\uparrow$ / G * - H * +

P: A B C * D E F $\uparrow$ / G * - H * +

Postfix Expression Evaluation

- Q: 5 * (6 + 2) – 12 / 4
- P : 5 , 6 , 2 , + , * , 12 , 4 , / , -

	Symbol Scanned	STACK
1.	5	5
2.	6	5,6
3.	2	5,6,2
4.	+	5,8
5.	*	40
6.	12	40,12
7.	4	40, 12, 4
8.	/	40, 3
9.	-	37

infix to postfix Algorithm: POLISH (Q, P)

1. Create empty stack
2. Scan Q from left to right and repeat step 3 to step 6 for each element of Q until the STACK is empty.
3. If an operands is encountered, add it to P
4. If a left parenthesis is encountered, push it onto STACK
5. If an operator X is encountered then:
    a) Repeatedly POP from STACK and add to P each operator (on the top of STACK) which has the same precedence as or higher precedence than X
    b) Add X to STACK

    [END of IF Structure]
6. If a right parenthesis is encountered then:
    a) Repeatedly POP from STACK and add to P each operator (on the top of STACK) until a left parenthesis is encountered.
    b) Remove the left parenthesis

    [END of IF Structure]

    [END of STEP 2 loop]
7. EXIT

Postfix Expression Evaluation Algorithm

This Algorithm Finds the VALUE of an Arithmetic Expression P Written in Postfix Notation.

1. Scan P from left to right and repeat step 3 and 4 for each element of P until “end of expression” is encountered.
2. If an operand is encountered, put it in STACK.
3. If an operator X is encountered then:
   a) Remove the two top elements of STACK
   b) Evaluate T2 X T1
   c) Place the result of (b) back on STACK.
4. [End of IF Structure]
5. [End of STEP1 loop]
6. Set VALUE equal to the top element on STACK.
7. Exit

Prefix conversion and evaluation

- Refer notebook

Balancing parenthesis / symbols Algorithm

1. Make an empty stack
2. Read the characters in the expression until end of expression
3. If char is an opening symbol, push it on to the stack
4. If it is a closing symbol, then

   if the stack is empty report an error

   otherwise, pop the stack. If the symbol popped is not the corresponding opening symbol, then report an error

5. At the end of expression, if stack is not empty, report an error.

Stacks in Recursion

- Recursion – Function calling itself

Example: factorial calculation

if n==5, then n! would be 5! = 5*4*3*2*1=120

5!=5 * 4 * 3 * 2 * 1 but 4*3*2*1 is really 4!

So:5! = 5* 4! but 4! is just 4* 3! and so on.

Stack in Recursive call

```
int fact(int n){
   int result;            //stores result of function
   if(n==1 || n==0){    //check for base case
      result=1;           //if n is 1 or 0 result is 1
   }
   else{               //else it is recursive case
      result=n * fact(n-1);  //result is n * (n-1)!
   }
   return result;
}
int main(void){
  int aa = fact(4);
}
```

<u>Stack content</u>

2*fact(1) - RA
3*fact(2) - RA
4*fact(3) - RA
fact(4) – RA

QUEUES - FIFO

- Queue is a linear structure in which Deletions can take place at one end only, called the Front and Insertions can take place only at other end, called the Rear.

- Three everyday examples of such a structure
  - Waiting Automobiles for Fuel
  - People Waiting in Line at Bank
  - Programs with the same Priority

AAA	BBB	CCC	DDD

AAA Front
DDD Rear

Representation of Queues

- FRONT: Containing the Location of the Front Element
- REAR: Containing the Location of the Rear Element
- Deletion: FRONT := FRONT + 1
- Insertion: REAR := REAR + 1
- After N Insertions, the Rear Element of the Queue will Occupy QUEUE [N]
- Enqueue is the term used to denote insertion and dequeue is the term used to denote deletion operation

Representation of Queues Cont…

FRONT- 1	AAA	BBB	CCC	DDD	…….	…….	……
REAR - 4	1	2	3	4	5	6	N

FRONT- 2		BBB	CCC	DDD	…….	…….	……
REAR - 4	1	2	3	4	5	6	N

FRONT- 2		BBB	CCC	DDD	EEE	FFF	……
REAR - 6	1	2	3	4	5	6	N

FRONT- 3			CCC	DDD	EEE	FFF	……
REAR - 6	1	2	3	4	5	6	N

ENQ (ITEM)

1. If REAR = N
   Then Print OVERFLOW
2. Set REAR=REAR+1
3. Set QUEUE[REAR]=ITEM
4. If FRONT= 0 //queue is initially empty
   Then Set FRONT=1

DEQ ()

1. If FRONT=0, then print UNDERFLOW
2. Else Set ITEM=QUEUE[FRONT]
3. If FRONT=REAR, then //Queue has only one element]
   Set FRONT=0 and REAR=0
   Else Set FRONT=FRONT+1
4. Print ITEM

Queue applications

- Printer
- Device connected to a network
- Client-server communication

Circular Queue

- Queue – Disadvantage is once if rear has reached the (end)max position, we cannot insert an element into queue though the number of elements is lesser than the max capacity of queue
- Solution – Circular queue – whenever front or rear reaches the end of the queue, it is wrapped around to the beginning.

Full queue condition

(Rear == max and front ==1) or (rear+1 == front)

ENQ (ITEM)

1. If (Rear == max and front ==1) or (rear+1 == front)
   Then Print OVERFLOW
2. If rear == MAX
   rear = 1
   else rear = rear +1
3. Set QUEUE[rear]=ITEM
4. If FRONT= 0 //queue is initially empty
   Then Set FRONT=1

DEQ ()

1. If FRONT=0, then print UNDERFLOW
2. Else Set ITEM=QUEUE[FRONT]
3. If FRONT=REAR, then //Queue has only one element]
   Set FRONT=0 and REAR=0
   Else if (FRONT = MAX) Then set Front = 1
       else Set FRONT=FRONT+1
4. Print ITEM