ADT Stack

Biswajit Prasad
Assistant Professor
Department of Computer Science
Maharaja Manindra Chandra College
Calcutta 700 003

Definitions

- **Objects**

  A finite sequence / chain of elements of the same type, where elements can be inserted or deleted from only one end.

- **Operations**
- **s_create (s)**

  Function to initialize an empty stack
- **s_dispose** (s)

  Function to dispose stack space held by s

Definitions…

- **s_empty (s) boolean**

  Function returns true if stack s is empty.

- **s_full (s) boolean**

  Function returns true if stack s is full.

Definitions…

- **push (s,e)**

  Function to place an item with e's value on the top of the stack s.

- **pop (s)**

  Function to return the top item of the stack;

  The item is removed from the stack.

- Stack is a Last-In-First-Out (LIFO) structure.

Some Example Problems

1. Duplicate the top element on a stack.
2. Print the elements of a stack .
   a) From top to bottom
   b) From bottom to top
   i. Stack becoming empty
   ii. Stack unchanged

3. Print the data from a file of integer in the reverse order.
4. Write a boolean function to return true if two stacks are equal.

Array implementation of stack

```
# define                102
MAXST                   4
# define TRUE      1
# define  FALSE    0
struct     stack_t       {
        int    top ;
        T    val [MAXST] ;
   } ;

typedef  struct  stack_t  * stack ;
```

Implementation…

```
stack    s_create ( )
 {  stack     p;
    if ((p = (stack) malloc(sizeof (struct stack_t)))
                                                == NULL)
    {
    Memory allocation error;  exit(0);
 }
 p->top = -1 ;  return p;
 }void s_dispose (stack        s)
 { if (s != NULL)
        free(s);
 }
```

Implementation …

```
int  s_empty (stack s)
 {
   return (s->top == -1) ;
 }

 int   s_full (stack s)
 {
   return ( s->top == MAXST-1);
 }
```

Implementation…

```
void push (stack s , T  e)
 {  if   (!(s_full(s)))
    s->val [ ++( s->top) ] = e;
    else stack overflow error;
 }

 T    pop (stack s )
 {  if   (!(s_empty(s)))
    return s->val [(s->top) -- ] ;
     else stack underflow error;
 }
```

Stack Implementation using Linked List

- s_create → init_l
- s_empty → empty_l
- push → insert_front
- pop → delete_front

A Classic Example

Arithmetic Expression Evaluation

Representation of arithmetic expressions

1. Infix Notation

   Operator appears in between operands .

   a + b

2. Prefix Notation

   Operator appears before operands .

   + a b

3. Postfix (or Reverse Polish) Notation

   Operator appears after operands .

   a b +

Infix Evaluation

In order to evaluate infix expression we have to assign priority to operators.

Operator	priority
** , u+ ,u- , not	4
*, / , div , mod ,and	3
+ , - , or	2
< , <= , = , >= , >	1

what about brackets?

Manual Conversion of Infix to Postfix

A * B / C

A / B ** C + D * E – A * C

<u>For Fully Parenthesized Expressions</u>

(((A / (B ** C)) + (D * E)) – (A * C))

or ABC ** / DE *+ AC *-

Postfix Evaluation

```
{   x = get_next_token(e) ;  while ( x != sentinel)
{
                                          if (is_operand(x))
                                                push (s,x);
    else {
              pop required no. of operands ;
                 perform operation on them ;  push
                 result on stack ;
    }
    x = get_next_token (e);
}
}
```

- The stack contains the value of the expression.

Conversion of Infix to Postfix

Observations

1. Operands are in the same order.

2. Operators are rearranged in the order they are evaluated.

3. Operators follow their operands.

4. Brackets are deleted.

Conversion…

- Store the operators in a stack till the right moment, then unstack them and output .

•Priorities are assigned to the operators in-stack and in-coming :

Symbol	ISP	ICP
)	-	-
**	3	4
* , /	2	2
+ , -	1	1
(	0	4

- Operators are taken out from stack as long as isp >= icp.

Conversion Algorithm

```
{   initialize stacks ;  push sentinel          '#' ;
x = get_new_token (e) ;
while ( x != '#' )
{
      if ( is_operand (x) )
                  output (x) ;  else          if (x == ')' )
                              {
                              while (( y = pop (s)) != '(' )
                                          output (y) ;
          }
          else
```

Conversion Algorithm…

```
{
                while (isp ( y = pop(s)) >= icp (x))
                        output (y) ;
        push (y) ;
        push (x);
        }       /*      end else */  x =
    get_next_token (e) ;
      }      /*      end      while   */  while (( y =
pop (s)) != '#')  output (y) ;
      output ('#');
    }        */end   algo     */
```

Conclusion

- Stack is a very useful data structure. Most of the modern computers and microprocessors provide a hardware stack. Even there are stack-oriented computer architectures.

- A very important application of stack is to implement recursive subroutine call / return mechanism.

- The scope rule and block-structure can also be implemented using stack.

- Stacks are used in the development of Compilers, System Programs, Operating Systems and in many elegant application algorithms.

ADT Queue

Objects

A finite sequence / chain of elements of the same type, where elements enter from the rear end and exit from the front end.

The front item has been in the queue the longest, and the rear item entered the queue most recently.

Operations

- **init_q (q)**
  Function to initialize an empty queue q.

- **dispose_q (q)**
  Function to dispose the memory space held by q

- **empty_q (q)**
  Boolean function to return true if the queue q is empty.

- **full_q (q)**
  Boolean function to return true if the queue q is full.

Operations

- **init_q (q)**
  Function to initialize an empty queue q.

- **dispose_q (q)**
  Function to dispose the memory space held by q

- **empty_q (q)**
  Boolean function to return true if the queue q is empty.

- **full_q (q)**
  Boolean function to return true if the queue q is full.

Operations…

- **enqueue (q,c)**
  Function to place an item with e's value into the queue q at the rear.

- **dequeue (q)**
  Function to take the front item out of the queue q.

- Queue is a FIRST-IN-FIRST-OUT (FIFO) structure.

A Few Example Problems

1. Append a queue p at the end of a queue q.
2. Add the elements of a queue and return the sum.

3.Print queue a) Forward b) reverse

Queue becoming empty ii. Queue unchanged

4. Boolean function equal_q (q1, q2); queue should remain unchanged.
5. Reverse a queue.
6. Procedure Replace (q, e, x) to replace every occurrence of element e in a queue with the value of x.

Implementation of Queue using Array

- Simple array implementation is not elegant and wasteful of memory.

- A circular array is a better option.

Implementation

```
#define   MAXQ    1024
#define   TRUE    1
#define   FALSE   0
struct  queue_t {
    int  front, rear ;
    int  count ;
    T    val [MAXQ] ;
};
typedef  struct queue_t  * queue ;
```

Implementation…

```
queue  init_q ( )
{
      queue q;
      if ((q = (queue) malloc(sizeof (struct queue_t)))
                                              == NULL)
      {
      Memory allocation error;  exit(0);
      }
      q->front = 1 ;  q->rear = 0 ;  q->count = 0 ;
      return q;
      }

void dispose_q (queue  q)
      { if (q != NULL)
            free(q);
      }
```

Implementation …

```
int empty_q (queue q)
{
        return ( q->count == 0) ;
}



int  full_q (queue  q)
{
        return (q->count == MAXQ) ;
}
```

Implementation…

```
void enqueue ( queue q , T  e )
 {    if (!(full_q(q)))
      {q->rear = ( q->rear + 1) % MAXQ;
      q->val [ q->rear ] = e ;
      (q->count) ++ ;}
      else Queue full error;
  }

 T  dequeue ( queue  q)
 {
      T         x ;
      if (!(empty_q(q)))
      {x = q->val [q-> front ] ;
      q->front = (q->front + 1) % MAXQ;
      (q->count) -- ;
      return x;}
      else Queue empty error;
 }
```

Implementation of Queue using Linked List

- Use a circular linked list with tail pointer

- init_q → init_l
- empty_q → empty_l

- enqueue → insert_after (tail); advance tail
- dequeue → delete_after (tail);

Application of Queue

- A major application of queue is in simulation [see Kruse for example] .

- In operating systems, queues are used for process management, I/O request handling etc.
  Example: Print queue of DOS, Message queue of Unix IPC.

- Queues are also used in some elegant algorithms like graph algorithms.

Types of Queue

- A special kind of queue is a double-ended queue, where the enqueue and dequeue operations can be performed at both front and rear.

- Another special kind of queue is a priority queue . An element with a higher priority can overtake another with a lower priority. But elements of the same priority are treated FIFO.