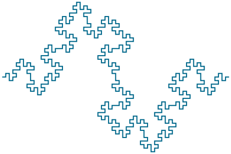
Fundamental Data Structures

Part 2: Stacks, Queues

Phung Hua Nguyen
HCMC University of Technology



April 6, 2020

OUTLINE

CONCEPTS

LIST ADT

IMPLEMENTATION
- Array-based
- Linked

SPECIAL LISTS
- Stack
- Queue

SUMMARY

STACK

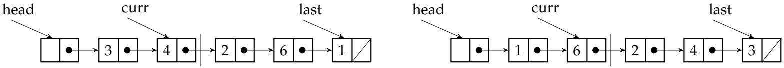
- special list with restricted access
- insert/remove just in one end
- First In Last Out

STACK APPLICATION

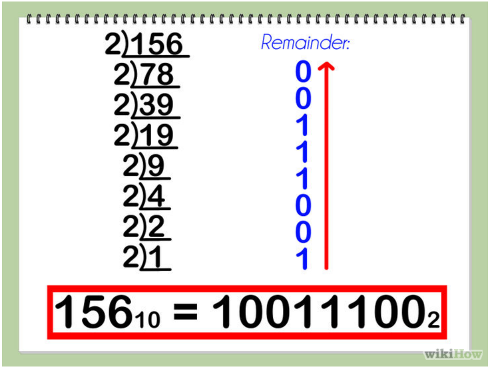
- Reverse data items
  - Reverse a list
  - Converse Decimal to Binary
- Parsing
- Postponement of processing data items
  - Infix to Postfix Transformation.
  - Evaluate a Postfix Expression.
- Backtracking
  - Goal Seeking Problem.
  - Knight's Tour.
  - Exiting a Maze.
  - Eight Queens Problem.
- ...

STACK APPLICATION: REVERSE DATA ITEMS

- Reverse a list



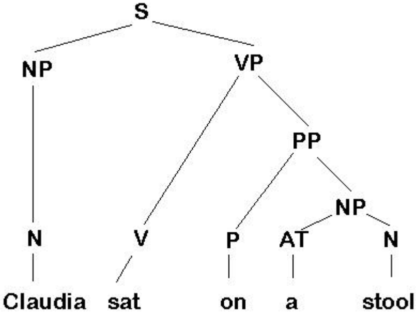
- Converse Decimal to Binary



$156_{10} = 10011100_2$

STACK APPLICATION: PARSING

$$\begin{array}{rcl} S & \rightarrow & NP\ VP \\ NP & \rightarrow & N \mid PN \mid AT\ N \\ VP & \rightarrow & V \mid V\ PP \\ PP & \rightarrow & P\ NP \end{array}$$

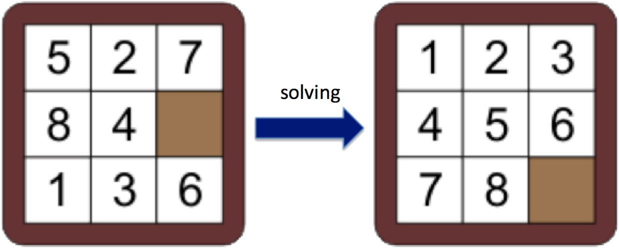


STACK APPLICATION: POSTPONEMENT OF PROCESSING DATA ITEMS

- Infix to Postfix Transformation
  a + b * (c - d) + e ⇒ a b c d - * + e +
- Evaluate a Postfix Expression
  10 3 4 + * 5 + ⇒ 75

STACK APPLICATION: BACKTRACKING

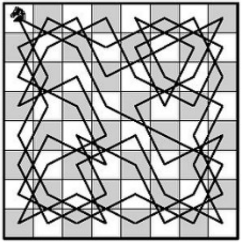
- Goal Seeking Problem.



- Knight’s Tour.
- Exiting a Maze.
- Eight Queens Problem.

STACK APPLICATION: BACKTRACKING

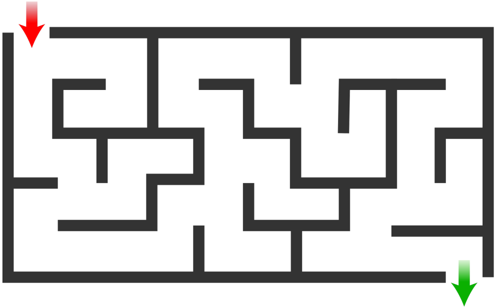
- Goal Seeking Problem.
- Knight's Tour.



- Exiting a Maze.
- Eight Queens Problem.

STACK APPLICATION: BACKTRACKING

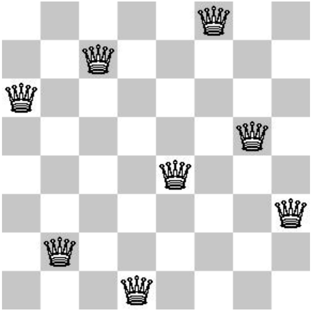
- Goal Seeking Problem.
- Knight's Tour.
- Exiting a Maze.



- Eight Queens Problem.

STACK APPLICATION: BACKTRACKING

- Goal Seeking Problem.
- Knight's Tour.
- Exiting a Maze.
- Eight Queens Problem.



WHAT CAN ACTIVITIES BE DONE ON A STACK?

- *push* an element on top of the stack
- *pop* the element at the top out of the satck
- *top*: get value of the top element of the stack
- *length* of the stack
- *isEmpty*: check if the stack is empty
- *isFull*: check if the stack is full
- *clear* the stack

STACK ADT

```
template <typename T>
class Stack {
public:
    Stack(){}
    ~Stack(){}

    virtual void push(const T& it) = 0;
    virtual T pop() = 0;
    virtual const T& top()const = 0;
    virtual int length() const = 0;
    virtual bool isEmpty() const = 0;
    virtual bool isFull() const = 0;
    virtual void clear() = 0;
}
```

Stack Implementation

- Array-based
- Linked

ARRAY-BASED STACKS

```
template <typename T>
class AStack: public Stack<T> {
private:
    int maxSize;
    int top;
    T∗ listArray;
public:
    AStack(int size = defaultSize){
        maxSize = size;
        top = 0;
        listArray = new T[maxSize];}
    ~AStack(){
        delete[] listArray;
    }
    ...
 };
```

Push a new element onto the stack

```
void push(const T& ele) {
    Assert(top < maxSize, "Stack is full");
    listArray[top++] = ele;
}
```

LINKED STACKS

```
template <typename T>
class LStack: public Stack<T> {
    private:
        Link<T>* top;
        int size;
    public:
    LStack() {
        top = NULL;
        size = 0;
    }
    ~LStack() {clear();}
    ...
};
```

Push a new element onto a linked stack

```
void push(const T& ele) {
    top = new Link<T>(ele,top);
    size++;
}
```

QUEUE

b14624 www.fotosearch.com

- a special list with restricted access
- insert in one end and remove in the other end
- First In First Out (FIFO)

QUEUE APPLICATION

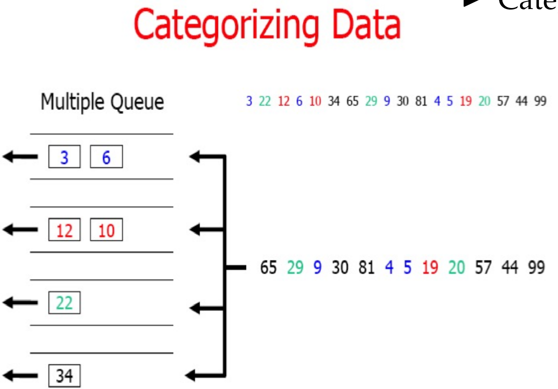
- Client-Server Model
  - Single-Server Model

QUEUE APPLICATION

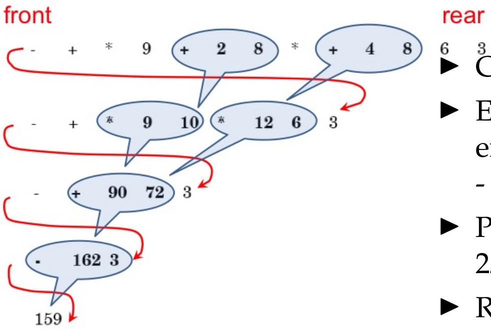
- Client-Server Model
  - Single-Server Model
  - Multi-Server Model

QUEUE APPLICATION

- Client-Server Model
  - Single-Server Model
  - Multi-Server Model
- Categorizing data



QUEUE APPLICATION



- Client-Server Model
  - Single-Server Model
  - Multi-Server Model
- Categorizing data
- Evaluate a **prefix** expression
  - + * 9 + 2 8 * + 4 8 6 3
- Polynomial Arithmetic
  $2x^2 + 4x^5$
- Radix Sort
- …

WHAT CAN ACTIVITIES BE DONE ON A QUEUE?

- *enqueue* an element at the rear of the queue
- *dequeue* an element at the front out of the queue
- *front*: get value of the element in the front of the queue
- *length* of the queue
- *isEmpty*
- *isFull*
- *clear* the queue

Queue ADT

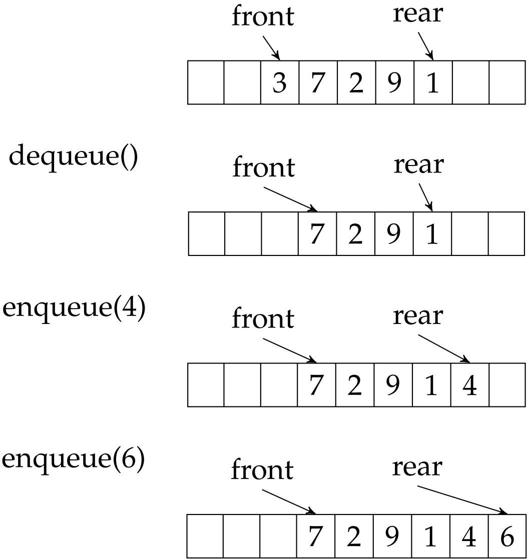
```
template <typename T>
class Queue {
public:
    Queue(){}
    ~Queue(){}

    virtual void enqueue(const T& element) = 0;
    virtual T dequeue() = 0;
    virtual const T& front() const = 0;
    virtual int length() const = 0;
    virtual bool isEmpty() const = 0;
    virtual bool isFull() const = 0;
    virtual void clear() = 0;
}
```

QUEUE IMPLEMENTATION

- Array-based
- Linked

ARRAY-BASED QUEUE



front rear

| | | 3 | 7 | 2 | 9 | 1 | | |

dequeue()

front rear

| | | | 7 | 2 | 9 | 1 | | |

enqueue(4)

front rear

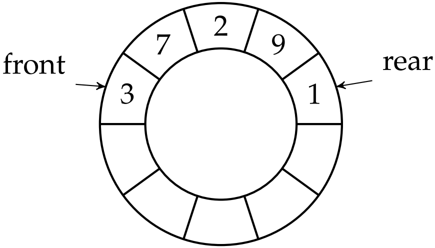
| | | | 7 | 2 | 9 | 1 | 4 | |

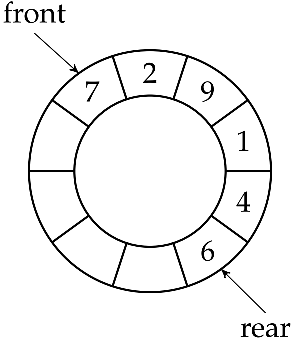
enqueue(6)

front rear

| | | | 7 | 2 | 9 | 1 | 4 | 6 |

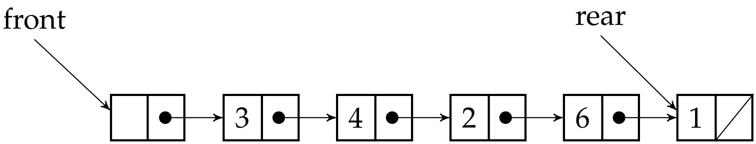
ARRAY-BASED QUEUE





Read book (page 132) for more details.

LINKED QUEUE



SUMMARY

- List is a data structure whose each element has a unique successor.
- Stack is a special list where insertions/deletions just occur in one end
- Queue is a special list where insertions occur in one end and deletions in the other end.