# Data Structures and Algorithms ( 3 )

Instructor: Ming Zhang

Textbook Authors: Ming Zhang, Tengjiao Wang and Haiyan Zhao

Higher Education Press, 2008.6 (the "Eleventh Five-Year" national planning textbook)

https://courses.edx.org/courses/PekingX/04830050x/2T2014/

# Chapter 3 Stacks and Queues

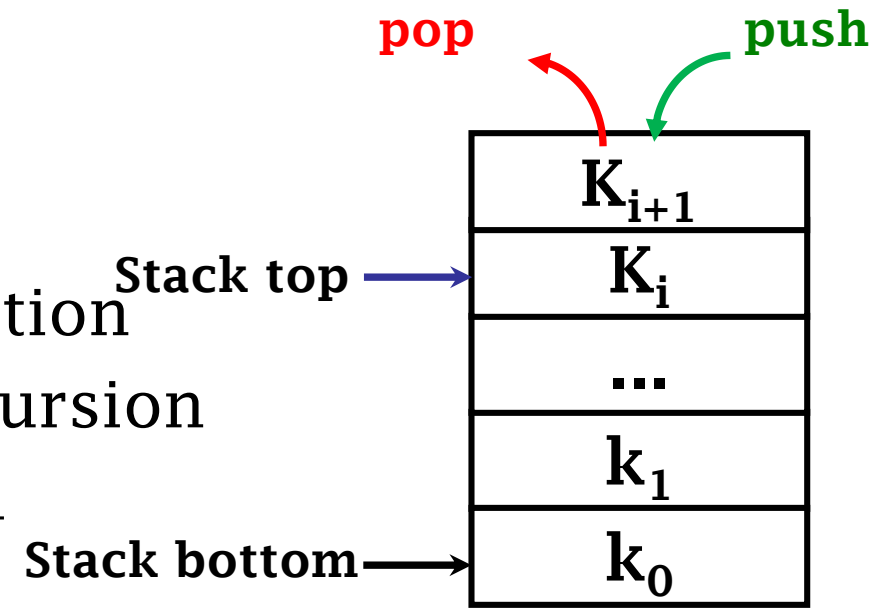
- Stacks
- Applications of stacks
  - Implementation of Recursion using Stacks
- Queues

# Linear lists with limited operation

- **Stack**
  - Operation are permitted only on one end
- **Queue**
  - Operation are permitted only on two ends

# Definition of stack

- **Last In First Out**
  - A linear list with limited access port
- **Main operation**
  - push、pop
- **Applications**
  - Expression evaluation
  - Elimination of recursion
  - Depth-first search

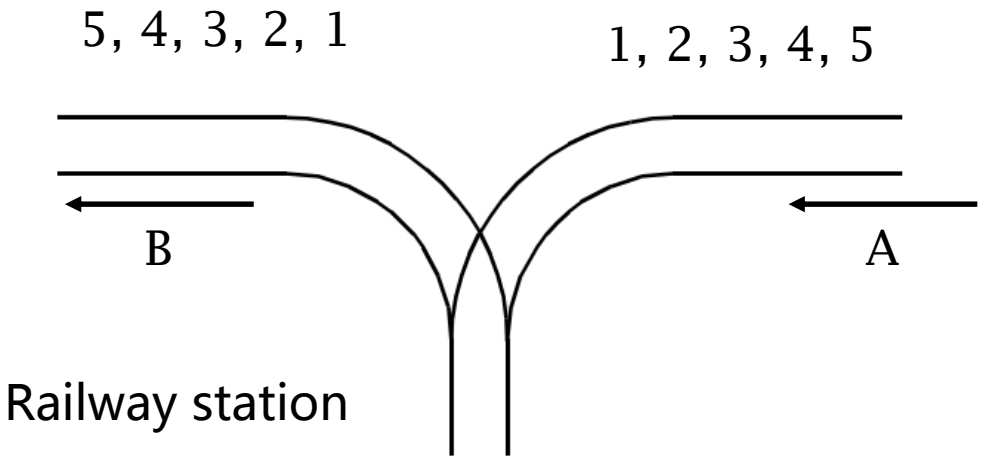


# Abstract data type of stacks

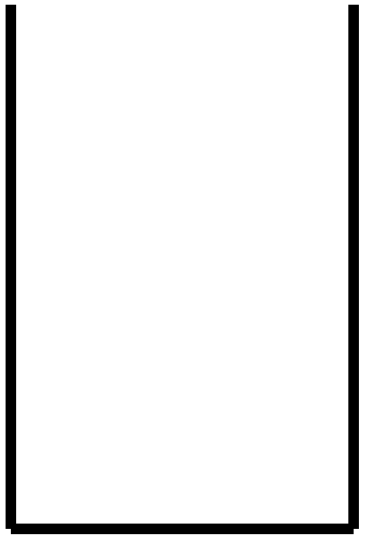
```
template <class T>
class Stack {
public:                    // Operation set of stacks
   void clear();           // Change into an empty stack
   bool push(const T item);
              // push item into the stack , return true if succeed, otherwise false
   bool pop(T& item);
             // pop item out of the stack ,  return true if succeed, otherwise false
   bool top(T& item);
             // read item at the top of the stack, return true if succeed, otherwise false
   bool isEmpty();      // If the stack is empty return true
   bool isFull();       // If the stack is full return true
};
```

# Railway station problem

- Judge whether the trains go out of the station in the right order?
  - http://poj.org/problem?id=1363
- N trains numbered as 1,2,….,n go into the train in order , given an arrangement , judge whether the trains go out of the station in the right order?



# Use legal reconstruction to find conflicts

# Question

- If the order of the item pushed into the stack is 1,2,3,4 , then what is the order of the item popped out of the stack ?
- There is an original input sequence 1 , 2 , … , n , you are required to get the output sequence of $p_1$ , $p_2$ , … , $p_n$ (They are a permutation of 1 , 2 , … , n)using a stack. If there exists subscript i , j , k , which meet the condition that i<j<k and $P_j<P_k<P_i$ , then whether the output sequence is legal or not ?

# Implementation of stacks

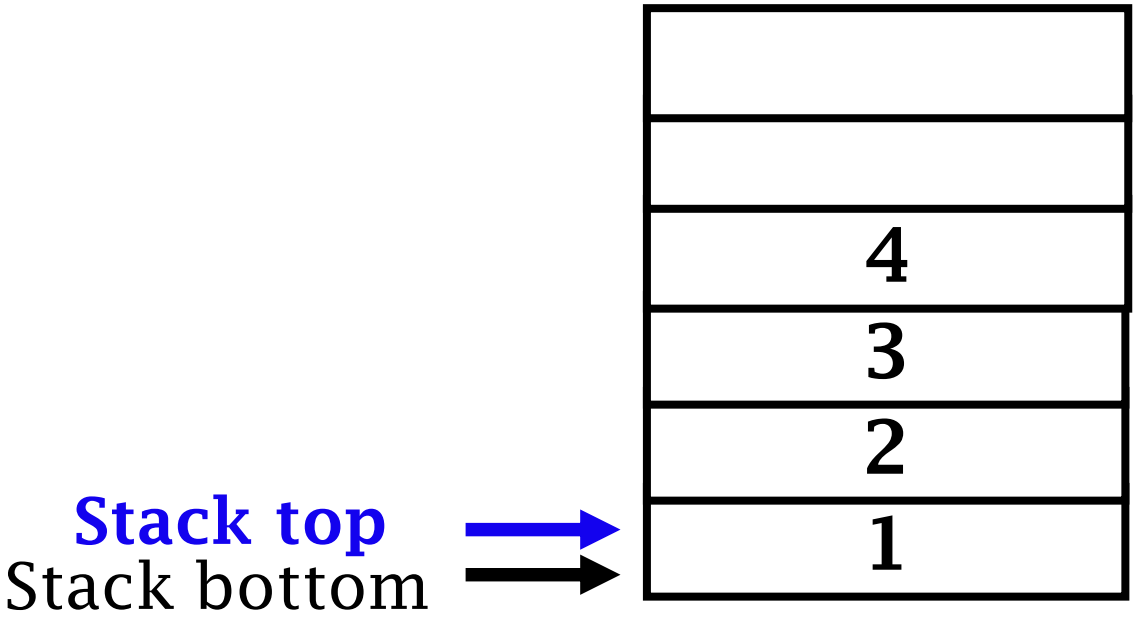
- **Array-based Stack**
  - Implemented by using vectors , is a simplified version of sequential list
    - The size of the stack
  - The key point is to make sure which end as the stack top
  - Overflow, underflow problem
- **Linked Stack**
  - Use single linked list for storage , in which the direction of the pointer is from stack top down

# The class definition of Array-based Stack

```
template <class T> class arrStack : public Stack <T> {
private:                  // storage of Array-based Stack
   int mSize;            // the number of elements that the stack can have at most
   int top;               // stack top , should be small than mSize
   T *st;                // array to put stack element
public:                   // implementation of the operation of the Array-based Stack
   arrStack(int size) {
      // creates an instance of Array-based Stack with given size
      mSize = size; top = -1; st = new T[mSize];
   }
   arrStack() {// creates an instance of Array-based Stack
      top = -1;
   }
   ~arrStack() { delete [] st; }
   void clear() { top = -1; }  // clear the stack
}
```

# Array-based Stack

- The index of the last element pushed into the stack is 4 , followed by 3,2,1 in order



# Overflow of Array-based Stack

- **Overflow**
  - When you perform push operation on a full stack (that already has maxsize elements), overflow will occur.
- **Underflow**
  - When you perform pop operation on an empty stack, underflow will occur.

# Push

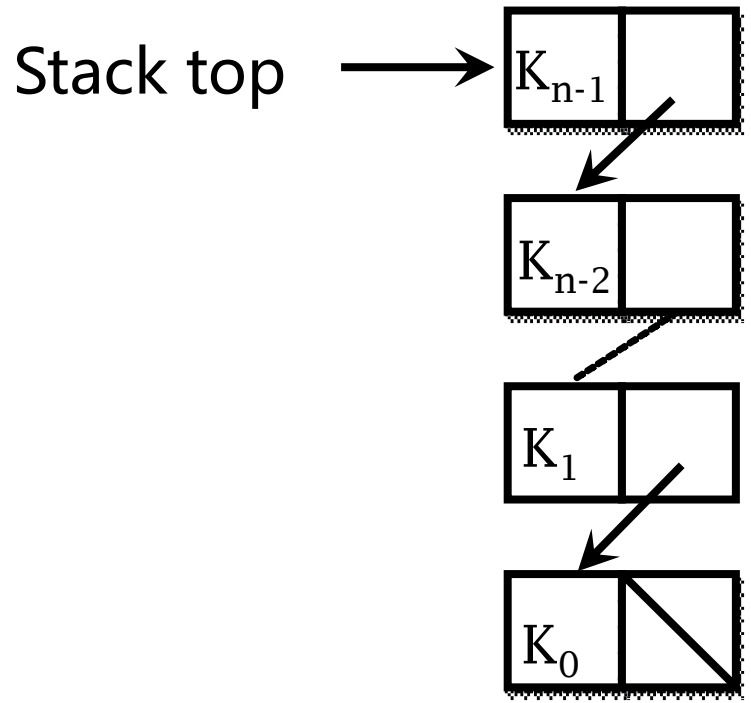
```
bool arrStack<T>::push(const T item) {
    if (top == mSize-1) {
// the stack has been full
        cout <<  "Stack overflow" << endl;
        return false;
    } else { //push new element into the stack and
modify the pointer of the stack top
        st[++top] = item;
        return true;
    }
}
```

# Pop

```
bool arrStack<T>::pop(T & item) {  // pop
    if (top == -1) {      // the stack is empty
        cout << " The stack is empty, you can’t
pop "<< endl;
        return false;
    } else {
        // Get top value and decrease top by 1
        item = st[top--];
        return true;
    }
}
```

# Definition of Linked Stack

- Use single linked list for storage
- The direction of the pointer is from stack top down



# Construction of Linked Stack

```
template <class T> class lnkStack : public Stack <T> {
private:                    // storage for linked stack
    Link<T>* top;
    //Pointer which points to the stack top
    int size; // the number of elements that the stack can
have at most
public:// implementation of the operation of the linked Stack
    lnkStack(int defSize) {  // constructed function
        top = NULL; size = 0;
    }
    ~lnkStack() {           // destructor function
        clear();
    }
}
```

# Push

```
// implementation of push operation of linked stack
bool lnksStack<T>:: push(const T item) {
    Link<T>* tmp = new Link<T>(item, top);
    top = tmp;
    size++;
    return true;
}

Link(const T info, Link* nextValue)  {
        // constructed function with 2 parameters
        data = info;
        next = nextValue;
}
```

# Pop

```
// implementation of pop operation of linked stack
bool lnkStack<T>:: pop(T& item) {
    Link <T> *tmp;
    if (size == 0) {
        cout << " The stack is empty, you can't pop"<< endl;
        return false;
    }
    item = top->data;
    tmp = top->next;
    delete top;
    top = tmp;
    size--;
    return true;
}
```

# Comparison of Array-based Stack and Linked Stack

- **Time efficiency**
  - All operations only take constant time
  - Array-based Stack and Linked Stack have almost the same time efficiency
- **Space efficiency**
  - The length of an Array-based Stack is fixed
  - The length of a Linked Stack is variable, with extra structural cost

# Comparison of Array-based Stack and Linked Stack

- In real applications , Array-based Stack is more widely used than Linked Stack
  - It is easy for Array-based Stack to perform relative replacement according to the position of stack top , quickly position and read the internal element
  - The time taken for Array-based Stack to read internal element is O(1). And the Linked stack has to walk along the chain of pointers, and is slower than Array-based Stack . It takes O(k) time to read the kth element.
- In general, the stack does not allow the internal operation, can only operate in the stack top

# Question : functions about stack in STL

- Top function gets the element of the stack top and returns the result back to the user
- Pop function pops a element out of the stack top ( if the stack is not empty )
  - Pop function is just an operation and doesn't return the result
  - pointer = aStack.pop() ? Error !
- Why does STL divide these two operations ? Why not provide ptop ?

# Applications of stacks

- Characteristic of stacks : **last-in first-out**
  - Embodies the transparency between elements
- Commonly used to deal with data which has recursive structure
  - **DFSevaluate the expression**
  - Subroutine / function call management
  - **Elimination of recursion**

# Evaluate the expression

- Recursive definition of expressions
  - The basic symbol set : {0 , 1 , ... , 9 , + , - , * , / , ( , ) }
  - Grammar set : {<expression> , <term> , <factor> , <constant> , <digit> }
- The infix expression 23+(34*45)/(5+6+7)
- Postfix expression 23 34 45 * 5 6 + 7 + / +

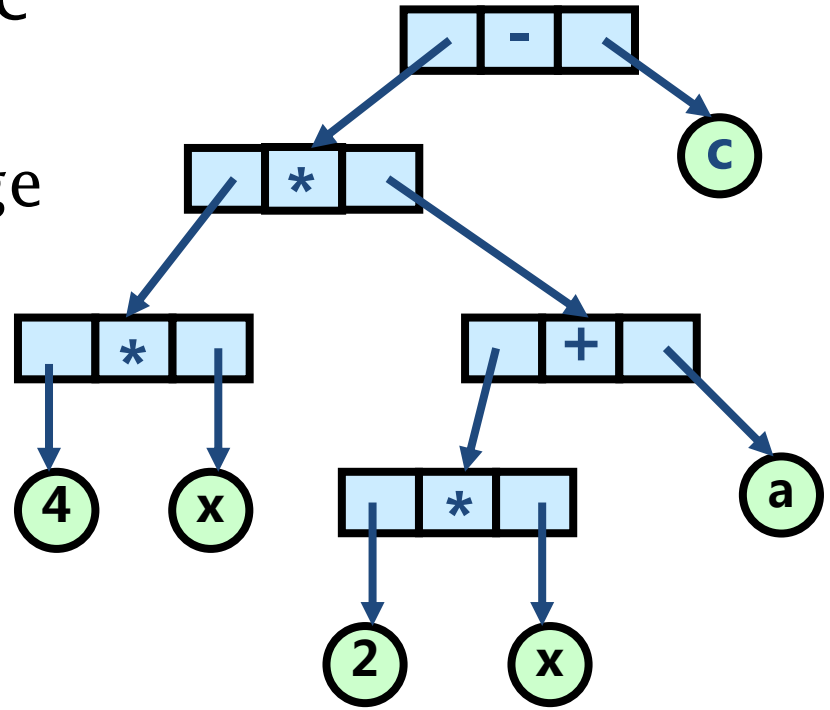
# Infix expression

- **Infix expression**

  4 * x * (2 * x + a) – c

  – Operator in the middle

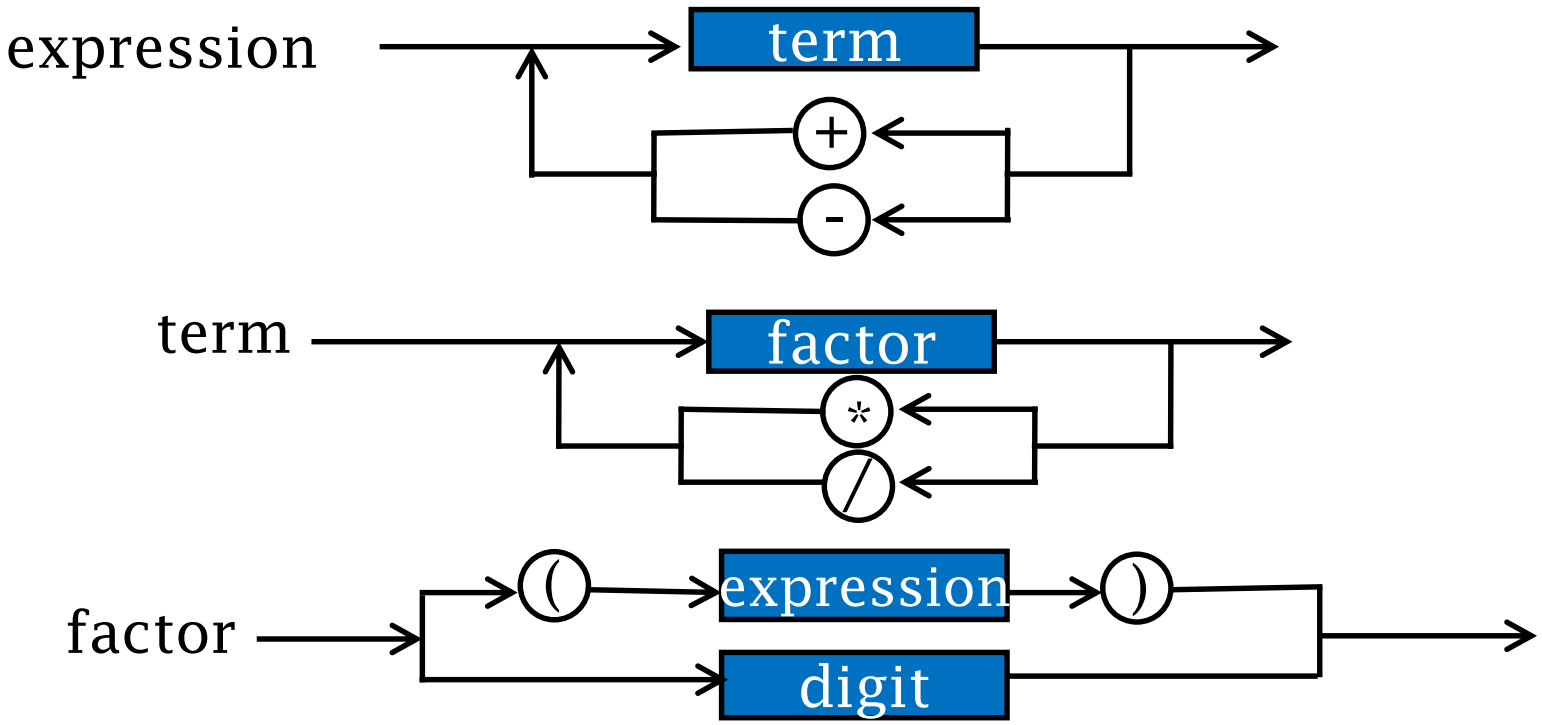
  – Need brackets to change the priority



# Syntax formula for infix expression

<expression> ::= <term> <term> +
| <term> <term> -
| <term>
<term> ::= <factor> < factor > *
| < factor > < factor > /
| < factor >
< factor > ::= < constant >
< constant > ::= <digit>
| <digit> <constant>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

# Graphical representation for expression recursion

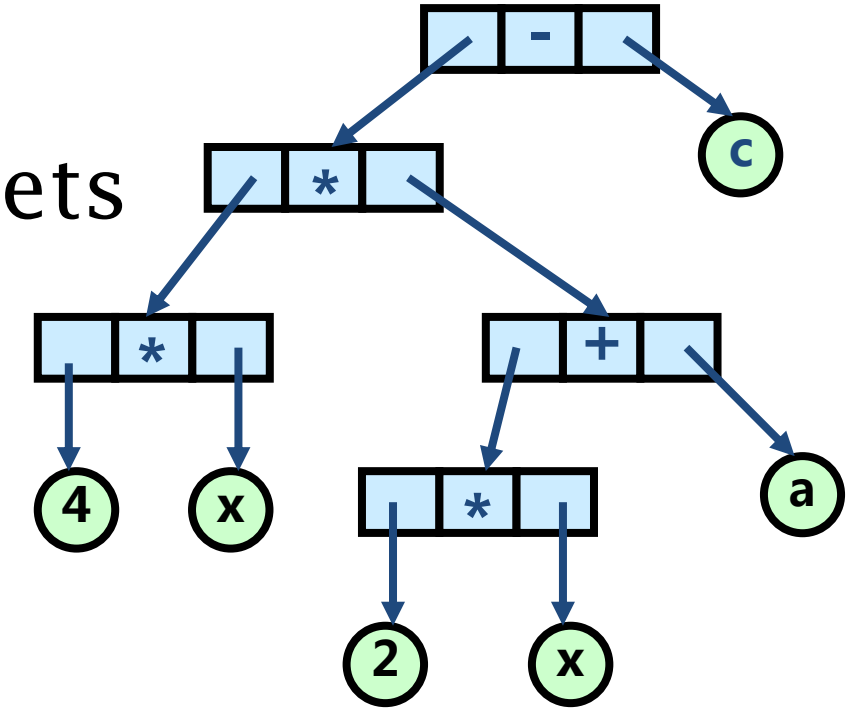


# Postfix expression

- **Postfix expression**

  4 x * 2 x * a + * c -

  - Operators behind
  - No need for brackets



# Postfix expression

<expression> ::= <term> <term> +
| <term> <term> -
| <term>
<term> ::= <factor> < factor > *
| < factor > < factor > /
| < factor >
< factor > ::= < constant >
< constant > ::= <digit>
| <digit> <constant>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

# Evaluating a postfix expression

- 23 34 45 * 5 6 + 7 + / + = ?

Calculation characteristics ?

The main differences between infix and postfix expression ?

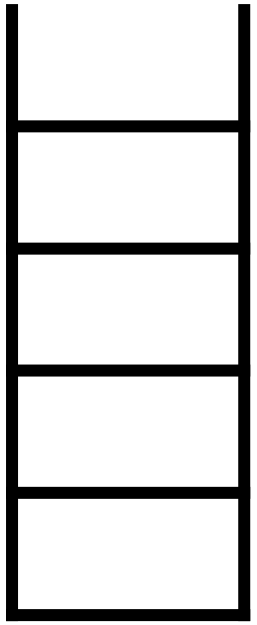
23 + 34 * 45 / ( 5 + 6 + 7 ) = ?

23 34 45 * 5 6 + 7 + / + = ?

**postfix expression to be handled :**

23 34 45 * 5 6 + 7 + / +

**change of the stack state**



| | |
|---|---|
| | 1180 |

**calculation result**

# Evaluating a postfix expression

- Loop : read symbol sequences of expressions ( assume "=" as the end of the input sequence ) , and analyze one by one according to the element symbol read
  1. When an operand is met , push
  2. When an operator is met, pop twice and get two operands, calculate them using the operator. And finally push the result into the stack.
- Continue the process above until the symbol " = " is met , then the value of the stack top is the value of the input expression

# The class definition of postfix calculator

```
class Calculator {
private:
    Stack<double> s;//the stack is used for pushing and storing operands
    // push two operands opd1 and opd2 from the stack top
    bool GetTwoOperands(double& opd1,double& opd2);
    // get two operands, and calculate according to op
    void Compute(char op);
public:
    Calculator(void){} ;
    // creates calculator instance and construct a new stack
    void Run(void);   // read the postfix expression, ends when meet "="
    void Clear(void); // clear the calculator to prepare for the next calculation
};
```

# The class definition of postfix calculator

```
template <class ELEM>
bool Calculator<ELEM>::GetTwoOperands(ELEM& opnd1, ELEM& opnd2) {
    if (S.IsEmpty()) {
        cerr << "Missing operand!" <<endl;
        return false;
    }
    opnd1 = S.Pop(); // right operator
    if (S.IsEmpty()) {
        cerr << "Missing operand!" <<endl;
        return false;
    }
    opnd2 = S.Pop(); // left operator
    return true;
}
```

# The class definition of postfix calculator

```
template <class ELEM> void Calculator<ELEM>::Compute(char op) {
    bool result;  ELEM operand1, operand2;
    result = GetTwoOperands(operand1, operand2);
    if (result == true)
        switch(op) {
            case '+' : S.Push(operand2 + operand1); break;
            case '-' : S.Push(operand2 - operand1); break;
            case '*' : S.Push(operand2 * operand1); break;
            case '/' : if (operand1 == 0.0) {
                    cerr << "Divide by 0!" << endl;
                    S.ClearStack();
                } else S.Push(operand2 / operand1);
                break;
        }
    else S.ClearStack();
}
```

# The class definition of postfix calculator

```
template <class ELEM> void Calculator<ELEM>::Run(void) {
    char c; ELEM newoperand;
    while (cin >> c, c != '=') {
        switch(c) {
            case '+': case '-': case '*': case '/':
                Compute(c);
                break;
            default:
                cin.putback(c); cin >> newoperand;
                Enter(newoperand);
                break;
        }
    }
    if (!S.IsEmpty())
        cout << S.Pop() << endl;          // print the final result
}
```

# Question

- 1. Stack is usually implemented by using single linked list. Can we use doubly linked list? Which is better ?

- 2. Please summarize the properties of prefix expression, as well as the evaluation process.

# Data Structures and Algorithms

## Thanks

the National Elaborate Course (Only available for IPs in China)
http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/

**Ming Zhang, Tengjiao Wang and Haiyan Zhao**
**Higher Education Press, 2008.6 (awarded as the "Eleventh Five-Year" national planning textbook)**