Chapter 3

Stack

By: S. Hassan Adelyar

Stack

- A **stack** allows access to only **one data item**, the **last item inserted**.
- This **capability** is useful in many **programming** situations.
- Most **microprocessors** use a **stack-based** architecture.
- **Internet web browsers** store the addresses of **recently** visited sites on a stack.
- Test **editors** usually provide an **undo mechanism** that cancels recent edition.

Stack

- **A simple array-based implementation**
- The **array implementation** of a stack is both simple and efficient, and is widely used in a variety of **computing applications**.
- One of the benefits of using a stack to implement method invocation is that it allows programs to use **recursion**.

```
Public static long factorial (long n)
{
    if (n <= 1)
        return 1;
    else
        return n*factorial(n-1);
}
```

Stack

```
class stackx
    {
    private int maxSize;
    private long[] stackArray;
    private int top;

    public stackx(int s)
    {
        maxSize = s;
        stackArray = new long[maxSize];
        top = -1;
    }
```

Stack

```
public void push(long j)
    {
        stackArray[++top] = j;
    }

    public long pop()
    {
        return stackArray[top--];
    }
```

Stack

```
public long peek()
    {
        return stackArray[top];
    }

    public boolean isempty()
    {
        return (top == -1);
    }
 public boolean isfull()
    {
        return (top == maxSize-1);
    }

}
```

Stack

```
class stackapp
{
    public static void main(String[] args)
    {
    stackx thestack = new stackx(10);
    thestack.push(20);
    thestack.push(40);
    thestack.push(60);
    thestack.push(80);

    while(!thestack.isempty() )
    {
        long value = thestack.pop();
        System.out.print(value);
        System.out.print(" ");
    }
    System.out.println(" ");

    }
}
```

Balanced Parentheies

Stack

- Stack can be used to evaluate arithmetic expressions.
- Each of the left parenthesis has a corresponding right parenthesis .
- There is never an occurrence of a right parenthesis that cannot be matched with a corresponding left parenthesis.
- The stack works by keeping unmatched left parenthesis. Every time the Algorithm encounters a right parenthesis, the corresponding left parenthesis is deleted from the stack.
- If the parenthesis in the input match correctly, things work out perfectly, and the stack is empty at the end of the input line.

Balanced Parentheses

Stack

```
class stackapp
{
    public static void main(String[] args)
    {
    boolean getans;
        String exptest;
        exptest = "({2,4}{})";
      getans = isBalanced(exptest);
      if (getans)
          System.out.print("Balanced");
      else
          System.out.print("Not Balanced");

    }
```

Balanced Parentheses

Stack

```
public static boolean isBalanced(String expression) {
        final char LEFT_NORMAL = '(';
        final char RIGHT_NORMAL = ')';
        final char LEFT_CURLY = '{';
        final char RIGHT_CURLY = '}';
        final char LEFT_SQURE = '[';
        final char RIGHT_SQURE = ']';

        CharStack store = new CharStack(15);
        int i;
        boolean failed = false;
        for (i = 0; !failed && (i< expression.length()); i++)
        {
            switch (expression.charAt(i))
            {
                case LEFT_NORMAL:
                case LEFT_CURLY:
                case LEFT_SQURE:
                    store.push(expression.charAt(i));
                    break;
```

Stack

```
case RIGHT_NORMAL:
                if (store.isempty() || (store.pop() != LEFT_NORMAL))
                    failed = true;
                break;
            case RIGHT_CURLY:
                if (store.isempty() || (store.pop() != LEFT_CURLY))
                failed = true;
                break;
            case RIGHT_SQURE:
                if (store.isempty() || (store.pop() != LEFT_SQURE))
                    failed = true;
                break;
          }
       }
       return (store.isempty() && !failed);
    }
}
```

Stack

```
class CharStack
   {
   private int maxSize;
   private long[] stackArray;
   private int top;

   public CharStack(int s)
   {
      maxSize = s;
      stackArray = new long[maxSize];
      top = -1;
   }
```

Stack

```
public void push(long j)
    {
        stackArray[++top] = j;
    }

    public long pop()
    {
        return stackArray[top--];
    }
public long peek()
    {
        return stackArray[top];
    }
```

Stack

```
public boolean isempty()
    {
        return (top == -1);
    }
 public boolean isfull()
    {
        return (top == maxSize-1);
    }
}
```