C#程序设计

第8章　泛型类与异常处理

杨琦
西安交通大学
计算机教学实验中心
http://ctec.xjtu.edu.cn

本章目标

- 掌握泛型概念和使用
- 了解异常处理机制

- 8-1 泛型
- 8-2 异常处理机制

8-1泛型

- 泛型（Generic） 允许您延迟编写类或方法中的编程元素的数据类型的规范，直到实际在程序中使用它的时候。
- 换句话说，泛型允许您编写一个可以与任何数据类型一起工作的类或方法。

- 通过数据类型的替代参数编写类或方法的规范。

- 当编译器遇到类的构造函数或方法的函数调用时，它会生成代码来处理指定的数据类型。
- 泛型的定义方法为：
  - class <类名> <T> {
  - ……
  - };

【例8-1】定义一个交换两个数据的泛型方法

- 输入和输出
- d1=5.2, d2=3.3
- str1=pku, str2=xjtu

【例8-1】

```
1. using System;
2. class My {
3.     public static void Swap<T>(ref T a, ref T b)
4.     {
5.         T temp;
6.         temp = a;
7.         a = b;
8.         b = temp;
9.     }
```

【例8-1】

```
1.  static int Main()
2.      {
3.          double d1 = 3.3, d2 = 5.2;
4.          string str1 = "xjtu", str2 = "pku";
5.          My.Swap<double>(ref d1, ref d2);
6.          Console.WriteLine("d1={0}, d2={1}", d1, d2);
7.          My.Swap<string>(ref str1, ref str2);
8.          Console.WriteLine("str1={0}, str2={1}", str1, str2);
9.          return 0;
10.     }
11. }
```

8-1泛型

【例8-2】求两个数据最大值的泛型类

```
1.  using System;
2.  class AnyType<T> {
3.      T x, y;
4.      public AnyType(T a, T b)    {
5.          x = a;
6.          y = b;
7.      }
8.      public T Max()     {
9.  int t=System.Collections.Comparer.Default.Compare(x, y);
10.         return t>0?x:y;
11.     }
12. }
```

【例8-2】定义一个任意类类型AnyType

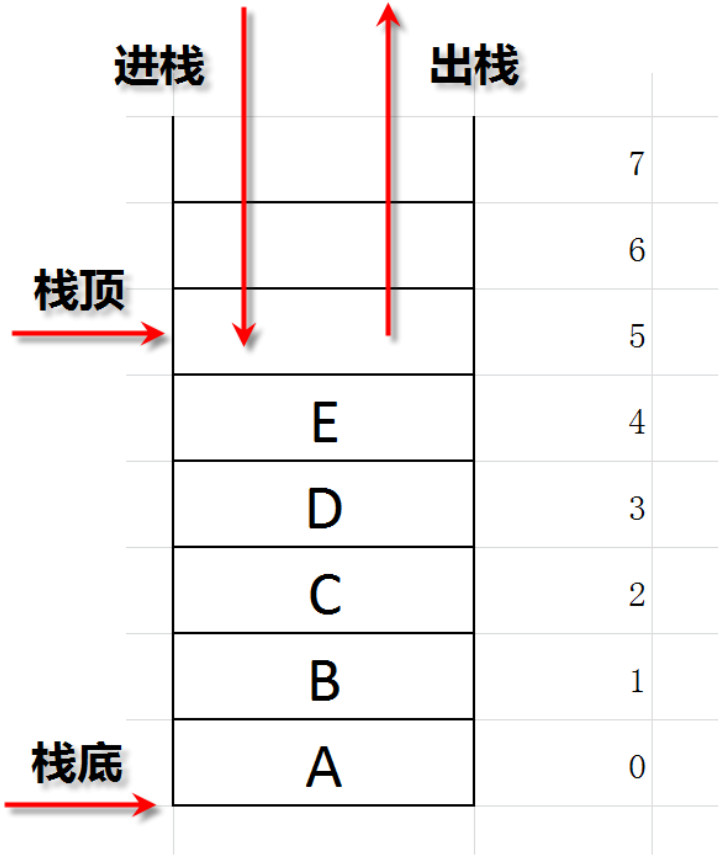
```
1.  class My {
2.      static int Main()    {
3.          AnyType<int> i = new AnyType<int>(1, 2);
4.          AnyType<double> d = new AnyType<double>(1.5, 2.7);
5.          AnyType<char> c = new AnyType<char>('a', 'b');
6.          AnyType<string> s = new AnyType<string>("Hello", "template");
7.          Console.WriteLine("整型类： " + i.Max());
8.          Console.WriteLine("双精度类： " + d.Max());
9.          Console.WriteLine("字符类： " + c.Max());
10.         Console.WriteLine("字符串类： " + s.Max());
11.         return 0;
12.     }
13. }
```

【例8-2】

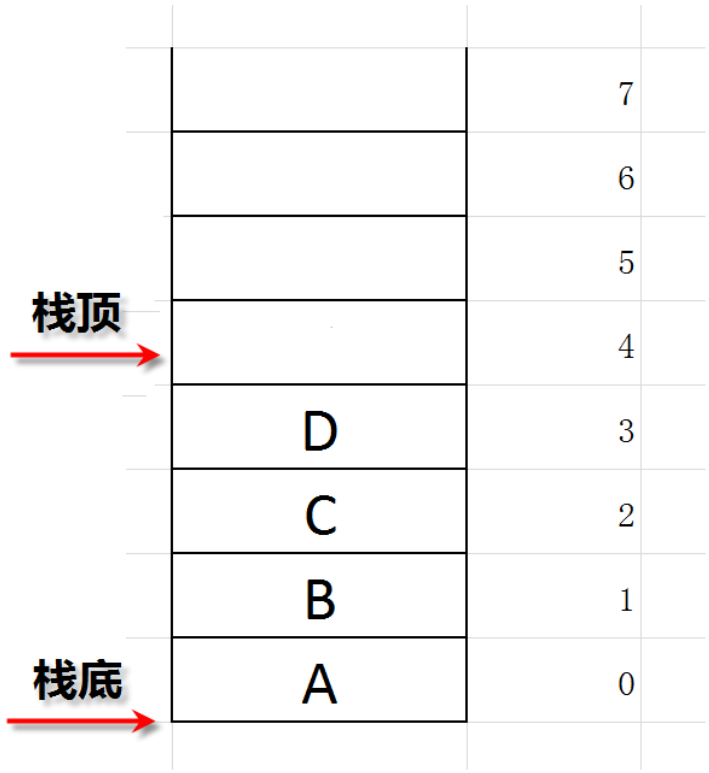
- 整型类：2
- 双精度类：2.7
- 字符类：b
- 字符串类：template

【应用案例】栈

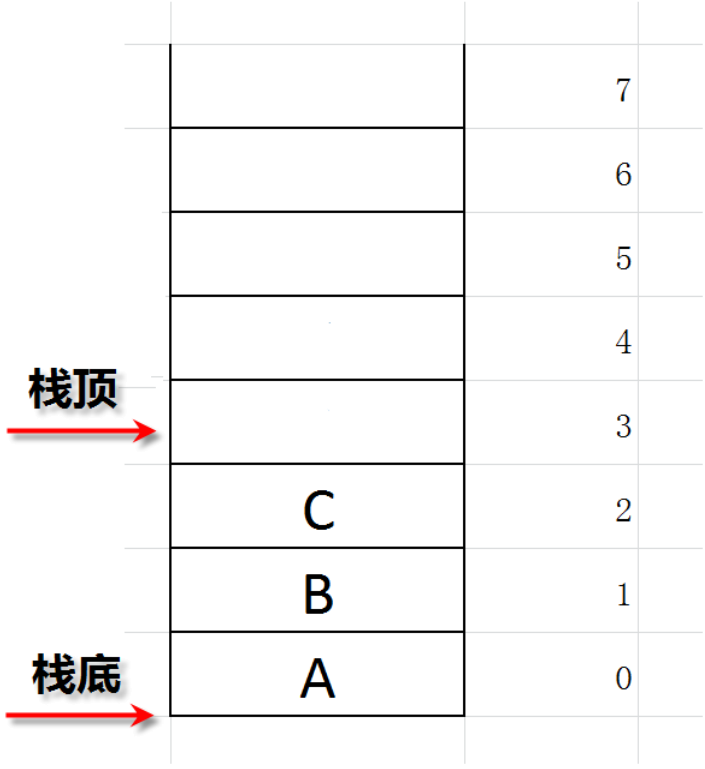
- 栈是一种操作受限的线性表。
- 栈只能从表的一端对数据进行操作（插入、删除），该端称为栈顶(Top)，另一端称为栈底(Bottom)。
- 栈底固定，栈顶浮动
- 栈又称为先进后出的表。



进出栈示意图

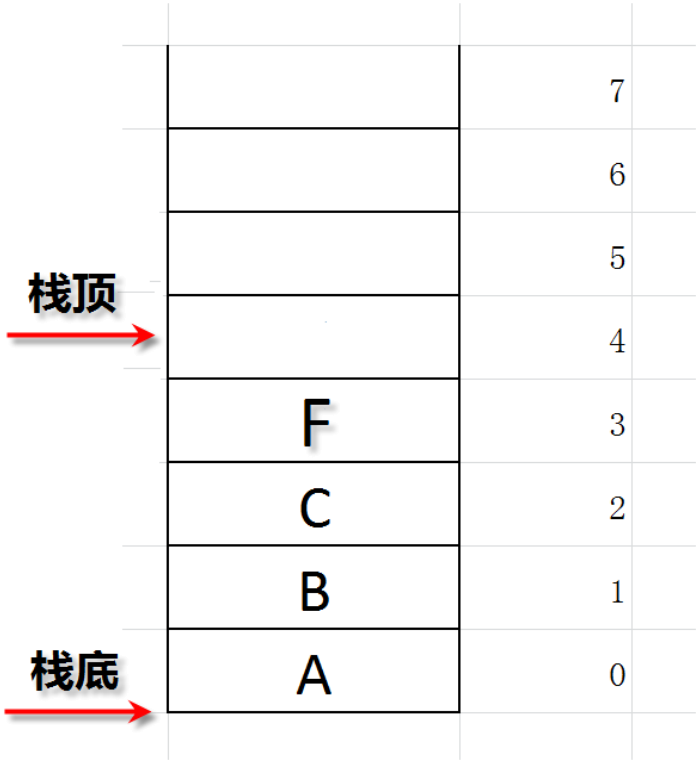


（1）A B C D E入栈



（2）D出栈

进出栈示意图



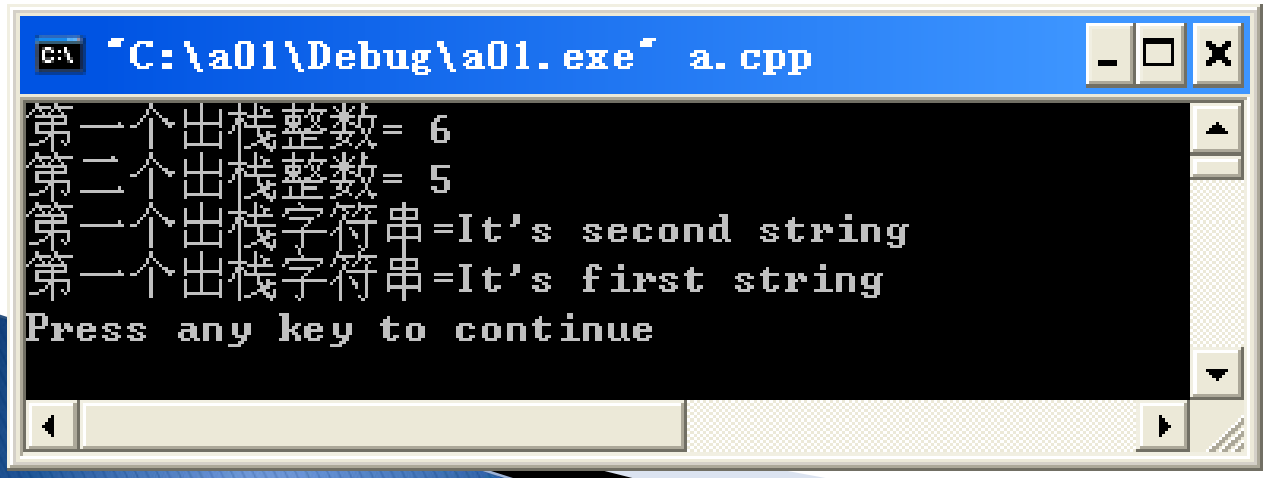
（3）F入栈

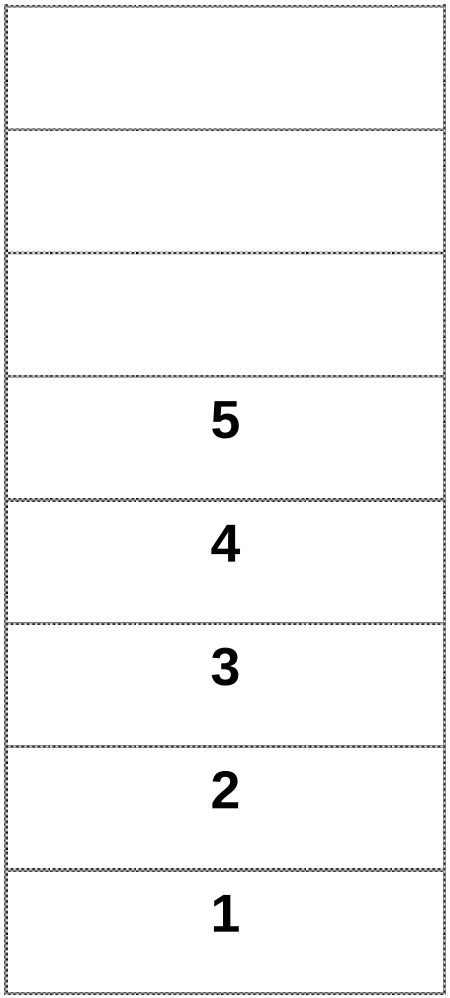
例8-3 定义一个通用的栈类

栈中最大数目
m_nMaxElement

栈中数据数目
m_nTop

```
bool Push(T);
bool Pop(T&);
```





整型、字符型

例8-3 定义一个通用的栈类

```
1.  using System;
2.  class AnyStack<T>
3.  {
4.      T[] m_tStack;
5.      int m_nMaxElement;
6.      int m_nTop;
7.      public AnyStack(int n)
8.      {
9.          m_tStack = new T[n];
10.         m_nMaxElement = n;
11.         m_nTop = 0;
12.     }
```

例8-3 定义一个通用的栈类

```
1.  public int GetLength() { return m_nTop; }
2.      public bool Push(T elem)
3.      {
4.          if (m_nTop <= m_nMaxElement)
5.          {
6.              m_tStack[m_nTop] = elem;
7.              m_nTop++;
8.              return true;
9.          }
10.         else
11.             return false;
12.     }
```

例8-3 定义一个通用的栈类

```
1.  public bool Pop(ref T elem)
2.      {
3.          if (m_nTop > 0)
4.          {
5.              m_nTop--;
6.              elem = m_tStack[m_nTop];
7.              return true;
8.          }
9.          else
10.             return false;
11.     }
12. }
```

例8-3 定义一个通用的栈类

```
class My{
    static int Main()   {
        int n = 0;
        string s1 = "";
        AnyStack<int> iStack = new AnyStack<int>(10);
        iStack.Push(5);
        iStack.Push(6);
        iStack.Pop(ref n);
        Console.WriteLine("第一个出栈整数= " + n);
        iStack.Pop(ref n);
        Console.WriteLine("第二个出栈整数= " + n);
```

例8-3 定义一个通用的栈类

```
1.          AnyStack<string> strStack = new AnyStack<string>(10);
2.          strStack.Push("It's first string");
3.          strStack.Push("It's second string");
4.          strStack.Pop(ref s1);
5.          Console.WriteLine("第一个出栈字符串=" + s1);
6.          strStack.Pop(ref s1);
7.          Console.WriteLine("第二个出栈字符串=" + s1);
8.          return 0;
9.      }
10. }
```

【泛型类实例】栈Stack

（1）栈的长度-Count，返回栈中元素个数
（2）复制到现有一维数组-CopyTo
（3）置（清）空栈-Clear
（4）入栈-Push，将x添加到栈顶，栈变化
（5）出栈-Pop，取出栈顶元素，栈变化
（6）取栈顶元素-Peek，返回栈顶元素值，栈不变化

【例8-8】利用泛型类Stack，实现字符串的入栈、出栈、清空等操作

【例8-4】实现字符串入栈、出栈、清空等操作

```
1.  class My{
2.      static int Main()   {
3.          Stack<string> stack1=new Stack<string> ();
4.          stack1.Clear();
5.          stack1.Push("程序设计");
6.          stack1.Push("英语");
7.          stack1.Push("数学");
8.          stack1.Push("物理");
9.          Console.WriteLine("各个元素出栈的顺序如下： ");
10.         while( stack1.Count > 0)
11.             Console.WriteLine(stack1.Pop());
12.         return 0;
13.     }
14. }
```

【例8-4】

- 输入和输出
- 各个元素出栈的顺序如下：
- 物理
- 数学
- 英语
- 程序设计

【泛型类实例】 队列Queue

- 队列也是一种操作受限的线性表。
- 队列只允许在表的前端（队头）进行删除操作，在表的后端（队尾）进行插入操作。
- 队列又称为先进先出（FIFO）的线性表。

队列的示意图

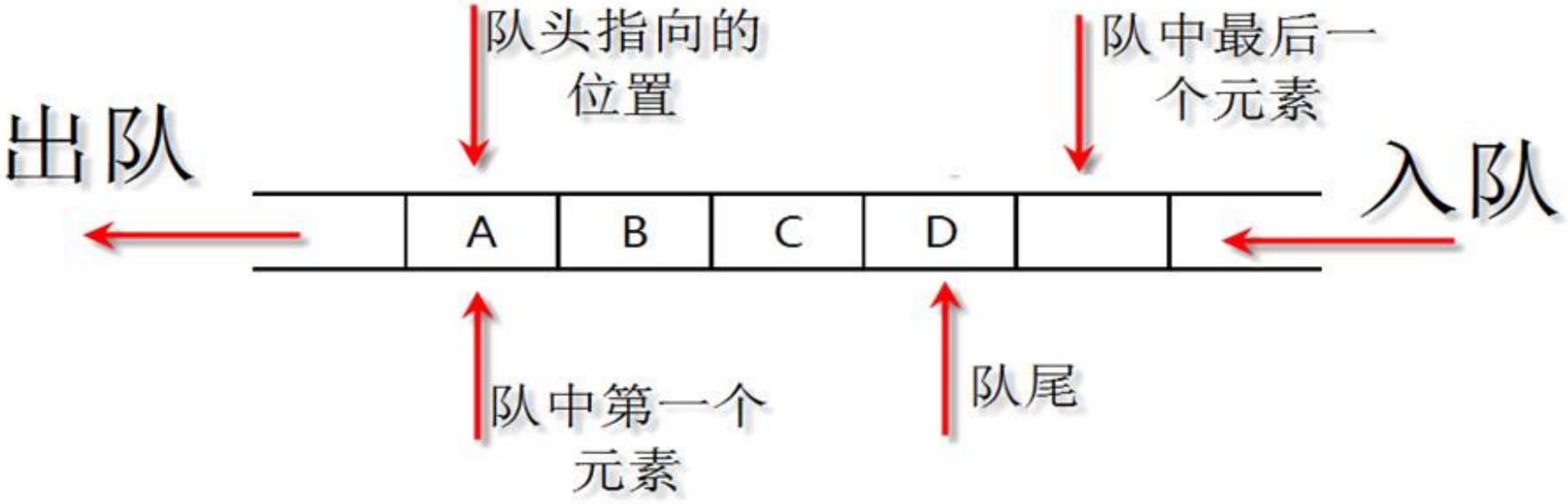
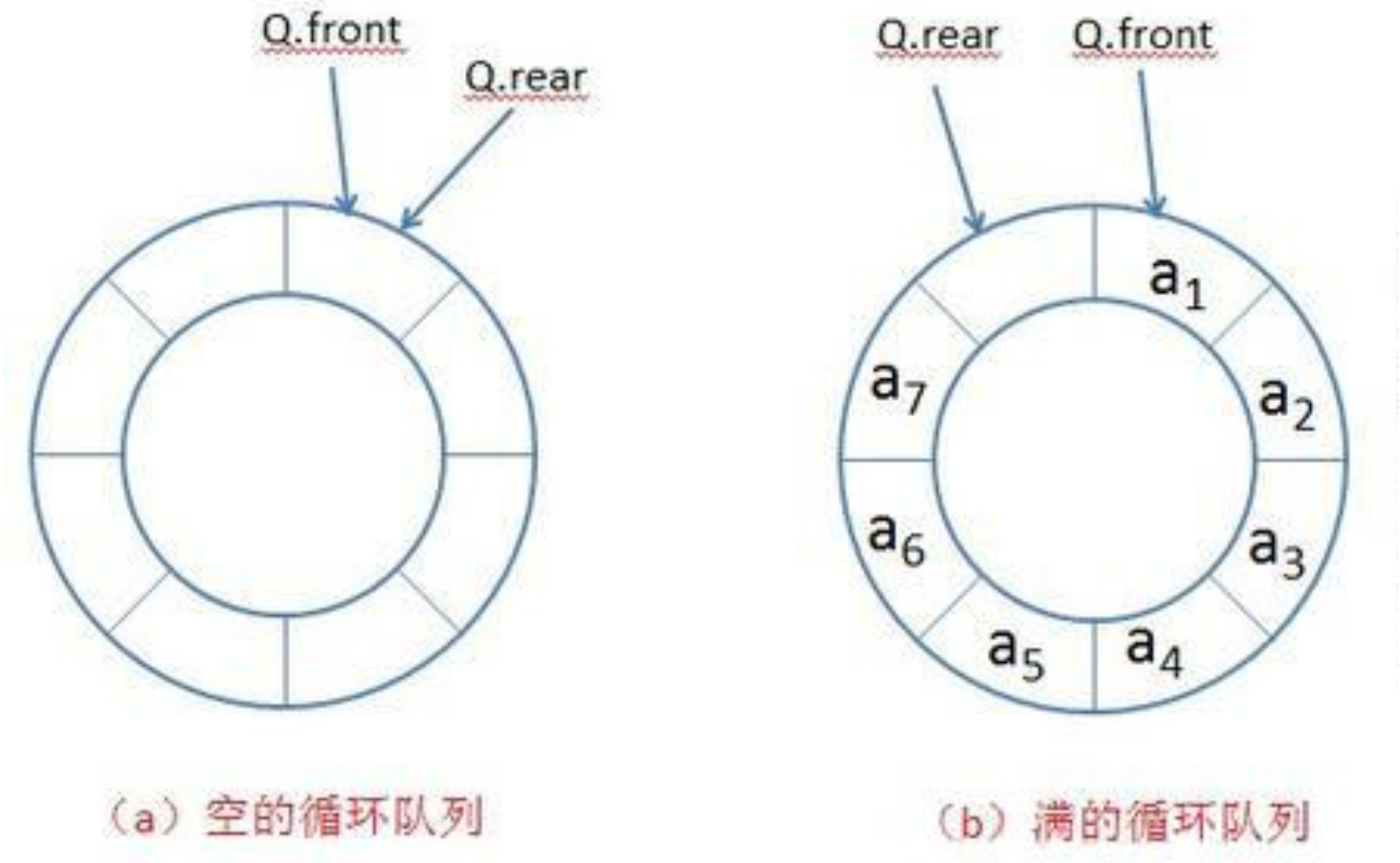


图6-15 一个循环队列的示意图



（a）空的循环队列

（b）满的循环队列

队列允许的操作

- （1）置空队列-Clear(Q)
- （2）求队列长度-Count
- （3）复制到现有一维数组-CopyTo(T[ ] , int)
- （4）入队-EnQueue(x)
- （5）出队-DelQueue()
- （6）取队头元素-Peek()

【例8-5】利用泛型类Queue

- 输入和输出
- 现在入队3个元素
- 目前队列中有以下3个元素：
- (0) 程序设计
- (1) 数据结构
- (2) 高等数学
- 现在出队1个元素
- 目前队列中有以下2个元素：
- (0) 数据结构
- (1) 高等数学

【例8-5】实现字符串的入队、出队、清空和输出队列元素等操作

- 现在入队1个元素
- 目前队列中有以下3个元素：
- (0) 数据结构
- (1) 高等数学
- (2) 操作系统

【例8-5】实现字符串的入队、出队、清空和输出队列元素等操作

```
1. class My{
2.     static int Main()   {
3.         Queue<string> q=new Queue<string> ();
4.         q.Clear();
5.         Console.WriteLine("现在入队3个元素");
6.         q.Enqueue("程序设计");
7.         q.Enqueue("数据结构");
8.         q.Enqueue("高等数学");
9.         Print(q);
```

【例8-5】实现字符串的入队、出队、清空和输出队列元素等操作

```
1.          Console.WriteLine("现在出队1个元素");
2.          q.Dequeue();
3.          Print(q);
4.          Console.WriteLine("现在入队1个元素");
5.          q.Enqueue("操作系统");
6.          Print(q);
7.          return 0;
8.      }
```

【例8-5】实现字符串的入队、出队、清空和输出队列元素等操作

```
1.  static void Print(Queue<string> q)
2.      {
3.          string []quearray=new string [q.Count];
4.          int i;
5.          q.CopyTo(quearray, 0);
6.          Console.WriteLine("目前队列中有以下{0}个元素： ", q.Count)
7.          for(  i = 0; i< q.Count; i++)
8.              Console.WriteLine("({0}) {1}", i, quearray[i]);
9.      }
10. }
```

【例8-6】线性表综合题目

- 设计自定义数据类型，该类型中由4个不同类型的数据构成，分别表示学生的学号、姓名、年龄和身高，用来处理一个学生的记录。利用泛型类List来实现线性表的插入、删除、显示和按学生定位等操作。

- 输入和输出
- 表长：3
- 学号:13010101 姓名:张三 年龄:18 身高170.5
- 学号:13010102 姓名:李四 年龄:19 身高167.2
- 学号:13010103 姓名:王五 年龄:18 身高176.7

【例8-6】

- 请输入要查找学生的学号
- 13010101
- 学号:13010101 姓名:张三 年龄:18 身高170.5
- 请输入要删除的记录的序号（从0开始）
- 0
- 删除一条记录后的线性表如下：
- 表长：2
- 学号:13010102 姓名:李四 年龄:19 身高167.2
- 学号:13010103 姓名:王五 年龄:18 身高176.7
- 清空线性表后内容：
- 表长：0

```
1.  using System;
2.  using System.Collections.Generic;
3.  class student {
4.      public string id;
5.      public string name;
6.      public int age;
7.      public double height;
8.      public student(string a, string b, int c, double d)  {
9.          id = a;          name = b;       age = c;       height = d;
10.     }
11.     public void ShowMe()   {
12.         Console.WriteLine("学号:{0}  姓名:{1}  年龄:{2}  身高{3}",
13. id, name, age, height);
14.     }
15. }
```

```
1.  class My {
2.      static int Main()  {
3.          List<student> list = new List<student>();
4.          student s1 = new student("13010101", "张三", 18, 170.5);
5.          student s2 = new student("13010102", "李四", 19, 167.2);
6.          student s3 = new student("13010103", "王五", 18, 176.7);
7.          list.Add(s1);
8.          list.Add(s2);
9.          list.Add(s3);
10.         Print(list);
11.         string sid;
12.         int n;
13.         Console.WriteLine("请输入要查找学生的学号");
14.         sid = Console.ReadLine();
```

```
1.          n = list.FindIndex(param => param.id.Equals(sid));
2.          if (n > -1)       {
3.              s1 = list[n];
4.              s1.ShowMe();
5.          }
6.          else
7.              Console.WriteLine("该学号不存在");
8.          Console.WriteLine("请输入要删除的记录的序号（从0开始
9.          n = Convert.ToInt32(Console.ReadLine());
10.         if (n >= 0 && n < list.Count)       {
11.             list.RemoveAt(n);
12.             Console.WriteLine("删除一条记录后的线性表如下： ");
13.             Print(list);
14.         }
15.         else
16.             Console.WriteLine("输入的记录号不对");
```

```
1.          list.Clear();
2.          Console.WriteLine("清空线性表后内容： ");
3.          Print(list);
4.          return 0;
5.      }
6.      static void Print(List<student> L)   {
7.          int i;
8.          Console.WriteLine("表长： {0}", L.Count);
9.          for (i = 0; i < L.Count; i++)
10.             L[i].ShowMe();
11.     }
12. }
```

8-2 异常处理机制

8-2 异常处理机制

- 异常是程序中的运行时错误，它违反了一个系统约束或应用程序约束，或出现了在正常操作时未预料的情况，如程序试图进行除0操作等。

- 在 C#中，程序中的运行时错误使用一种称为“异常”的机制在程序中传播。

- 程序可以选择对这个异常进行处理，即进行异常处理。如果程序没有提供处理异常的代码，系统会挂起这个程序。

8-2 异常处理机制

- C# 异常处理时建立在四个关键词之上的：try、catch、finally 和 throw。

- try：一个 try 块标识了一个将被激活的特定的异常的代码块。后跟一个或多个 catch 块。
- catch：程序通过异常处理程序捕获异常。catch 关键字表示异常的捕获。
- finally：finally 块用于执行给定的语句，不管异常是否被抛出都会执行。例如，如果您打开一个文件，不管是否出现异常文件都要被关闭。
- throw：当问题出现时，程序抛出一个异常。使用 throw 关键字来完成。

8-2 异常处理机制

- try块是必须的，catch块和finally块至少存在一个。
- 如果catch块和finally块都存在，则finally块必须放在最后。
- .net基础类库中定义很多类，每个类代表一个指定的异常。
- 当异常发生时，系统首先创建该类型的异常变量，然后寻找合适的catch子句处理它。

- 异常处理的实现
- throw语句可以使代码显式的抛出一个异常，其语法如下：
- throw 异常表达式;//引发一个异常，此异常的值就是通
- 过计算该表达式而产生的值
- throw;//只能用在catch块中，重新引发当前正由该
- catch块处理的那个异常。

例8-7 异常处理机制的使用

```
1.  using System;
2.  class My{
3.      static void testfun(int StudentAge)   {
4.          try      {
5.              if (StudentAge < 0 || StudentAge > 20)
6.              throw new Exception("学生年龄必须在0~20之间!");
7.              Console.WriteLine("学生年龄：" + StudentAge);
8.          }
9.          catch (Exception e)
10.         {
11.             Console.WriteLine(e.Message);
12.         }
13.     }
```

例8-7 异常处理机制的使用

```
1.  static int Main()
2.      {
3.          Console.WriteLine("请输入小学生年龄： ");
4.          int x = Convert.ToInt32(Console.ReadLine());
5.          testfun(x);
6.          return 0;
7.      }
8.  }
```

异常规格说明

```
try{
   // 引起异常的语句
}
catch( ExceptionName e1 ){
   // 错误处理代码
}
catch( ExceptionName e2 ){
   // 错误处理代码
}
catch( ExceptionName eN ){
   // 错误处理代码
}
finally
{
   // 要执行的语句
}
```

Sytem.SystemException 类的预定义的异常类

异常类	描述
System.IO.IOException	处理 I/O 错误。
System.IndexOutOfRangeException	处理当方法指向超出范围的数组索引时生成的错误。
System.ArrayTypeMismatchException	处理当数组类型不匹配时生成的错误。
System.NullReferenceException	处理当依从一个空对象时生成的错误。
System.DivideByZeroException	处理当除以零时生成的错误。
System.InvalidCastException	处理在类型转换期间生成的错误。
System.OutOfMemoryException	处理空闲内存不足生成的错误。
System.StackOverflowException	处理栈溢出生成的错误。

【例8-8】除0异常。

```
1.  using System;
2.  class My
3.  {
4.      static double Division(double x, double y)
5.      {
6.          if (y == 0)
7.              throw new Exception("divided by zero.");
8.          return x / y;
9.      }
10.     static void Main()
11.     {
12.         double a = 1, b = 0;
```

【例8-8】除0异常。

```
1.          double result = 0;
2.          Console.WriteLine("Input two numbers:");
3.          a = Convert.ToDouble(Console.ReadLine());
4.          b = Convert.ToDouble(Console.ReadLine());
5.          try       {
6.              result = Division(a, b);
7.              Console.WriteLine("{0} / {1} = {2}", a, b, result);
8.          }
9.          catch (Exception e)   {
10.             Console.WriteLine("Exception occurred: " + e.Message);
11.         }
12.     }
13. }
```

[例8-9]求一元二次方程的根

```
1.  using System;
2.  class My  {
3.      static void Root(double a, double b, double c)    {
4.          double x1, x2, delta;
5.          delta = b * b - 4 * a * c;
6.          if (a == 0) throw new Exception("divide by zero");
7.          if (delta < 0) throw new Exception("delta<0");
8.          x1 = (-b + Math.Sqrt(delta)) / (2 * a);
9.          x2 = (-b - Math.Sqrt(delta)) / (2 * a);
10.         Console.WriteLine("x1=" + x1 + "\nx2=" + x2);
11.     }
```

```
1.      static int Main()   {
2.          double a, b, c;
3.          Console.WriteLine("Please input a, b, c = ? ");
4.          a = Convert.ToDouble(Console.ReadLine());
5.          b = Convert.ToDouble(Console.ReadLine());
6.          c = Convert.ToDouble(Console.ReadLine());
7.          try       {
8.              Root(a, b, c);
9.          }
10.         catch (Exception e)       {
11.             Console.WriteLine("Exception occurred. " + e.Message);
12.         }
13.         return 0;
14.     }
15. }
```

（1）编写一个求绝对值的函数模板，并测试。

```
1.  using System;
2.  class abstype<T>
3.  {
4.      T x;
5.      public abstype(T a)
6.      {
7.          x = a;
8.      }
9.      public double Abs()
10.     {
11.         string s = x.ToString();
12.         double y = Convert.ToDouble(s);
```

（1）编写一个求绝对值的函数模板，并测试

```
1.          y = y > 0 ? y : -y;
2.          return y;
3.      }
4.      public void Print()
5.      {
6.          Console.WriteLine("输入的数的绝对值为：{0}", Abs());
7.      }
8.  }
```

（1）编写一个求绝对值的函数模板，并测试

```
1.  static class test
2.  {
3.      static void Main()
4.      {
5.          int x,y;
6.          Console.Write("请输入任意一个数： ");
7.          x = Convert.ToInt32(Console.ReadLine());
8.          abstype<int>s = new abstype<int>(x);
9.          s.Print();
10.     }
11. }
```

其他语句

- 1．using语句
- using语句获取一个或多个资源，执行一个语句，然后释放该资源。

- 2．lock语句
- lock语句用于获取某个给定对象的互斥锁，执行一个语句，然后释放该锁。

- 3．checked语句和unchecked语句
- checked 语句和 unchecked 语句用于控制整型算术运算和转换的溢出检查上下文。

习 题

- 1．编写一个求绝对值的函数模板，并测试。
- 2．请将例4-5的冒泡排序函数改写成为模板函数并编写一个程序进行测试。
- 3．在8-1.2类模板一节中，我们定义了一个任意类类型AnyType，请编写一个程序来使用该AnyType类模板。
- 4．例8-2中所定义的通用栈类实际上是不完善的，如无法根据用户需求改变栈的大小，没有提供栈满溢出无法压入和空栈无法弹出提示等，请改进该程序。
- 5．C++中的数组类型比较简单，它的下标只能从0开始，没有负数下标，而且没有数组越界检查。请用类模板设计一个newArray类，该类的对象可以是整型、浮点型、字符型等任何元素类型的数组，而且当访问数组成员时，如果下标越界，程序可以报错并终止。如下是一些例子：

习 题

- newArray <int> A1(3)　　//同传统类型的整型数组
- //包含5个元素的浮点型数组，其成员为A2[-2], A2[-1], A2[0], A2[1], A2[2]
- newArray <float> A2(-2, 3)
- 请编写一个测试程序。
- 6．例6-1给出的求阶乘n!的函数，当用户的输入太大时（如51），会出现错误，请编写一个程序，使用异常处理机制来解决这一问题。
- 7．编程并观察当库函数sqrt()的参数为负数，log()的参数为0时，系统会出现什么情况，请解决之。

结　束　语

- 学好程序设计语言的唯一途径是

上机练习

- 你的编程能力与你在计算机上投入的时间成

正比