# 10.7 抽象数据类型

Stock类非常具体。然而，程序员常常通过定义类来表示更通用的概念。例如，就实现计算机专家们所说的抽象数据类型（abstract data type，ADT）而言，使用类是一种非常好的方式。顾名思义，ADT以通用的方式描述数据类型，而没有引入语言或实现细节。例如，通过使用栈，可以以这样的方式存储数据，即总是从堆顶添加或删除数据。例如，C++程序使用栈来管理自动变量。当新的自动变量被生成后，它们被添加到栈项；消亡时，从栈中删除它们。

下面简要地介绍一下栈的特征。首先，栈存储了多个数据项（该特征使得栈成为一个容器——一种更为通用的抽象）；其次，栈由可对它执行的操作来描述。

* 可创建空栈。
* 可将数据项添加到栈顶（压入）。
* 可从栈顶删除数据项（弹出）。
* 可查看栈否填满。
* 可查看栈是否为空。

可以将上述描述转换为一个类声明，其中公有成员函数提供了表示栈操作的接口，而私有数据成员负责存储栈数据。类概念非常适合于ADT方法。

私有部分必须表明数据存储的方式。例如，可以使用常规数组、动态分配数组或更高级的数据结构（如链表）。然而，公有接口应隐藏数据表示，而以通用的术语来表达，如创建栈、压入等。程序清单10.10演示了一种方法，它假设系统实现了bool类型。如果您使用的系统没有实现，可以使用int、0和1代替bool，false和true。

**程序清单10.10 stack.h**

// stack.h -- 用于表示抽象数据类型——栈的类定义

#ifndef STACK\_H\_

#define STACK\_H\_

typedef unsigned long Item;

class Stack

{

private:

 enum {MAX = 10}; // 常量

 Item items[MAX]; // 保存数据项的数组

 int top; // 栈顶数据项的索引

public:

 Stack();

 bool isempty() const;

 bool isfull() const;

 // 如果栈已满，则push()方法返回falses，否则返回true

 bool push(const Item & item); // 在栈中添加数据项

 // 如果栈为空，则pop()方法返回false，否则返回true

 bool pop(Item & item); // 从栈顶删除数据

};

#endif

在程序清单10.10所示的示例中，私有部分表明，栈是使用数组实现的；而公有部分隐藏了这一点。因此，可以使用动态数组来代替数组，而不会改变类的接口。这意味着修改栈的实现后，不需要重新编写使用栈的程序，而只需重新编译栈代码，并将其与已有的程序代码链接起来即可。

接口是冗余的，因为pop()和push()返回有关栈状态的信息（满或空），而不是void类型。在如何处理超出栈限制或者清空栈方面，这为程序员提供了两种选择。他可以在修改栈前使用isempty()和isfull()来查看，也可以使用push()和pop()的返回值来确定操作是否成功。

这个类不是根据特定的类型来定义栈，而是根据通用的Item类型来描述。在这个例子中，头文件使用typedef用Item代替unsigned long。如果需要double栈或结构类型的栈，则只需修改typedef语句，而类声明和方法定义保持不变。类模板（参见第14章）提供了功能更强大的方法，来将存储的数据类型与类设计隔离开来。

接下来需要实现类方法，程序清单10.11提供了一种可行的实现。

**程序清单10.11 stack.cpp**

// stack.cpp -- Stack成员函数

#include "stack.h"

Stack::Stack() // 创建一个空栈

{

 top = 0;

}

bool Stack::isempty() const

{

 return top == 0;

}

bool Stack::isfull() const

{

 return top == MAX;

}

bool Stack::push(const Item & item)

{

 if (top < MAX)

 {

 items[top++] = item;

 return true;

 }

 else

 return false;

}

bool Stack::pop(Item & item)

{

 if (top > 0)

 {

 item = items[--top];

 return true;

 }

 else

 return false;

}

默认构造函数确保所有栈被创建时都为空。pop()和push()的代码确保栈顶被正确处理。这种保证措施是OOP更为可靠的原因之一。假设要创建一个独立数组来表示栈，创建一个独立变量来表示栈顶索引。则每次创建新栈时，都必须确保代码是正确的。没有私有数据提供的保护，则很可能由于无意修改了数据二导致程序出现非常严重的故障。

下面来测试该栈。程序清单10.12模拟了售货员的行为——使用栈的后进先出方式，从购物筐的最上面开始处理购物订单。

**程序清单10.12 stacker.cpp**

// stacker.cpp -- 测试Stack类

#include <iostream>

#include <cctype> // or ctype.h

#include "stack.h"

int main()

{

 using namespace std;

 Stack st; // 创建一个空栈

 char ch;

 unsigned long po;

 cout << "Please enter A to add a purchase order,\n"

 << "P to process a PO, or Q to quit.\n";

 while (cin >> ch && toupper(ch) != 'Q')

 {

 while (cin.get() != '\n')

 continue;

 if (!isalpha(ch))

 {

 cout << '\a';

 continue;

 }

 switch(ch)

 {

 case 'A':

 case 'a': cout << "Enter a PO number to add: ";

 cin >> po;

 if (st.isfull())

 cout << "stack already full\n";

 else

 st.push(po);

 break;

 case 'P':

 case 'p': if (st.isempty())

 cout << "stack already empty\n";

 else {

 st.pop(po);

 cout << "PO #" << po << " popped\n";

 }

 break;

 }

 cout << "Please enter A to add a purchase order,\n"

 << "P to process a PO, or Q to quit.\n";

 }

 cout << "Bye\n";

 return 0;

}

程序清单10.12中的while循环删除输入行中剩余部分，就现在而言这并非是必不可少的，但它使程序的修改更方便（第14章将对这个程序进行修改）。下面是该程序的运行情况：

Please enter A to add a purchase order,

P to process a PO, or Q to quit.

A

Enter a PO number to add; 17885

Please enter A to add a purchase order,

P to process PO, or Q to quit.

P

PO #17885 popped

Please enter A to add a purchase order,

P to process a PO, or Q to quit.

Enter a PO number to add: 17965

Please enter A to add a purchase order,

P to process a PO, or Q to quit.

A

Enter a PO number to add: 18002

Please enter A to add a purchase order,

P to process a PO, or Q to quit.

P

PO #18002 popped

Please enter A to add a purchase order,

P to process a PO, or Q to quit.

P

PO #17965 popped

Please enter A to add a purchase order,

P to process a PO, or Q to quit.

P

stack already empty

Please enter A to add a purchase order,

P to process a PO, or Q to quit.

Q

Bye