**车厢调度**

**学号：U201813466**

**姓名：赵凯豪**

**问题描述**

1、用编号依次为1，2，3，……，n表示停在铁路调度站入口处的车厢序列。

2、用一个栈形象地表示为火车的调度站。

3、利用栈先进后出的性质，结合递归和回溯算法，实现编号1…n的车厢的所有可能的序列。

设计

为了使车厢能够调度，需要定义一个栈，利用栈先进后出的性质，改变车厢的顺序。

对栈的抽象数据类型进行定义：

ADT stack

{

数据对象：D={ai | ai∈CharSet，i=1，2，……，n，n≥0}

数据关系; R={<ai-1,ai> | ai-1,ai∈D,i=2，……，n}

}

MakeEmpty()：

操作结果：构造一个空栈S。

IsEmpty()：

判断栈是否为空栈，若s为空栈，则返回1，否则返回0。

Push(int item)

在栈s的栈顶插入新的栈顶元素item。

Pop()

删除s的栈顶元素并返回其值。

以上定义的详细设计如下代码所示：

struct StackNode

{

int data[maxSize];

int top;

}s;

int n;

void MakeEmpty() //置空栈

{

s.top=-1;

}

int IsEmpty() //判栈空否

{

if(s.top==-1)

return 1;

else

return 0;

}

void Push(int item) //进栈

{

s.top++;

s.data[s.top]=item;

}

int Pop() //出栈并取栈顶

{

int temp;

temp=s.data[s.top];

s.top--;

return temp;

}

生成输出序列的函数的伪码算法如下：

void OutPut(int pp,int cc,int Ways[])

if(pp<n)

{

一个数进栈后,有两种处理方式：要么立刻出栈,要么进行下一个数的进栈

}；

if(!IsEmpty())

{

一个数出栈后,有两种处理方式：要么继续出栈,要么继续下一个数的进栈

}

if(pp==n&&IsEmpty())

{

一种可能输出序列产生，输出

}

该函数详细代码如下：

void OutPut(int pp,int cc,int Ways[])

{

int m,i;

if(pp<n) //一个数进栈后,有两种处理方式：要么立刻出栈,要么进行下一个数的进栈

{

Push(pp+1);

OutPut(pp+1,cc,Ways);

Pop();

}

if(!IsEmpty()) //一个数出栈后,有两种处理方式：要么继续出栈,要么继续下一个数的进栈

{

m=Pop();

Ways[cc]=m;

cc++;

OutPut(pp,cc,Ways);

Push(m);

}

if(pp==n&&IsEmpty()) //一种可能输出序列产生，输出

{

for(i=0;i<cc;i++)

cout<<Ways[i]<<" ";

cout<<"\t";

}

}

主函数如下：

int main

{

输入数据并初始化该栈，

之后调用函数void OutPut(int pp,int cc,int Ways[])

输出结果；

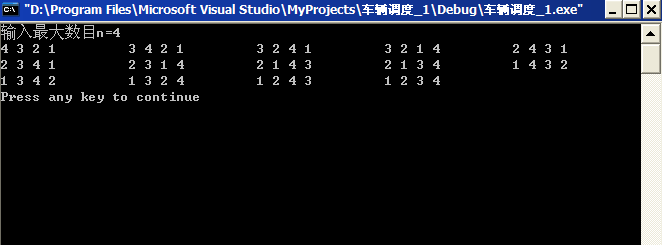
}

调试报告

本程序的栈其实也是一个简单的数组，它存储先后进入该栈的数，然后输出各个不同的序列。由于此程序比较简单明了，不是很复杂，按照概要设计来进行代码设计的时候基本没有什么大的问题。在调试过程中，主要是在OutPut这个函数设计过程中遇到不小的麻烦，而这主要的原因就在于平时对于递归的调用算法没太钻研所致，在此过程中借助于课本上以及网上查到的关于递归的资料才算勉强完成这个函数。

下面是程序运行结果：

输入n=4的情况下：



结果正确无误。

经验体会

通过这次课程设计，我对于将课本上的知识运用到实际生活中的整个过程深有体会了，不像之前对知识单纯的停留在课本及考试的要求上。对于我们学过的知识，我们不仅要尽量深入而透彻地去理解以及渗透，因为，在我看来，只有充分的了解渗透才能更好的将课本所学的知识运用到实验和生活中去。这次课程设计，我受益匪浅。