**北京邮电大学软件学院**

**2016-2017学年第1学期实验报告**

**课程名称：\_算法与数据结构\_**

**实验名称：\_栈、队列与递归算法设计\_**

**实验完成人： \_刘浩博\_**

**姓名：\_刘浩博\_ 学号：\_2015212086\_ 成绩：\_\_\_\_\_\_\_\_\_\_\_**

**指导老师： \_贾红娓\_**

**日期： 2016 年 11 月 9 日**

⼀.**实验⽬的**

仅仅认识到栈和队列是两种特殊的线性表是远远不够的，本次实验的⽬的在于使学⽣

深⼊了解栈和队列的特征，以便在实际问题背景下灵活运⽤它们；同时还将巩固这两

种结构的构造⽅法，接触较复杂问题的递归算法设计。

**⼆. 实验内容**

**1.** 数制转换问题

**[问题描述]**

将⼗进制数N和其它d进制数的转换是计算机实现计算的基本问题，其解决⽅案很多，

其中最简单⽅法基于下列原理：即除d取余法。例如：（1348）10=（2504）8

**N N div 8 N mod 8**

1348 168 4

168 21 0

21 2 5

2 0 2

从表中我们可以看出，最先产⽣的余数4是转换结果的最低位，这正好符合栈的特性即后进

先出的特性。所以可以⽤顺序栈来模拟这个过程。

**[基本要求]**

对于键盘输⼊的任意⼀个⾮负的⼗进制整数，打印输出与其等值的⼋进制数。由于上

述

的计算过程是从低位到⾼位顺序产⽣的⼋进制数的各个数位，⽽打印输出，⼀般来说应从

⾼位到低位进⾏，恰好和计算过程相反。因此可以先将计算过程中得到的⼋进制数的各位

进栈，待相对应的⼋进制数的各位均产⽣以后，再使其按顺序出栈，并打印输出。即得到

了与输⼊的⼗进制数相对应的⼋进制数。

**[测试数据]**

　　由学⽣依据软件⼯程的测试技术⾃⼰确定。注意测试边界数据。

**2.** 括号匹配的检验

**[问题描述]**

假设表达式中允许有两种括号：圆括号和⽅括号，其嵌套的顺序随意，即（（）[ ]）

或

[（[ ] [ ]）]等为正确格式，[（ ]）或（（（]均为不正确的格式。检验括号是否匹配的⽅法

可⽤“**期待的紧迫程度**”这个概念来描述。例如：考虑下列的括号序列：

　　[　(　[　]　[　]　)　]

　　1　2　3　4　5　6　7　8

当计算机接受了第1个括号以后，它期待着与其匹配的第8个括号的出现，然⽽等来的却是

第2个括号，此时第1个括号“[”只能暂时靠边，⽽迫切等待与第2个括号相匹配的第7个括号

“）”的出现，类似的，因只等来了第3个括号“[”，此时，其期待的紧迫程度较第2个括号更

紧迫，则第2个括号只能靠边，让位于第3个括号，显然第3个括号的期待紧迫程度⾼于第2

个括号，⽽第2个括号的期待紧迫程度⾼于第1个括号；在接受了第4个括号之后，第3个括

号的期待得到了满⾜，消解之后，第2个括号的期待匹配就成了最急迫的任务了，…… ，

依次类推。可见这个处理过程正好和栈的特点相吻合。

**[基本要求]**

　　读⼊圆括号和⽅括号的任意序列，输出“匹配”或“此串括号匹配不合法”。

**[测试数据]**

　　输⼊（[ ]（）），结果“匹配”

　　输⼊ [（ ）]，结果“此串括号匹配不合法”

**[实现提⽰]**

设置⼀个栈，每读⼊⼀个括号，若是左括号，则作为⼀个新的更急迫的期待压⼊栈中；

若是右括号，并且与当前栈顶的左括号相匹配，则将当前栈顶的左括号退出，继续读下⼀

个括号，如果读⼊的右括号与当前栈顶的左括号不匹配，则属于不合法的情况。在初始和

结束时，栈应该是空的。

**三. 实验环境**

VC6.0

**四. 实验结果**

两个程序均能正确实现实验目标，并且无错误情况出现。

**五. 附录**

**Convert.cpp:**

#include<stdio.h>

#include<stdlib.h>

#define INITSIZE 100

#define INCREAMENT 10

typedef struct

{

int \*base;

int \*top;

int stacksize;

}SqStack;

int InitStack(SqStack &S)

{

S.base=(int \*)malloc(INITSIZE \* sizeof(int));

S.top=S.base;

S.stacksize=INITSIZE;

return 0;

}

int push(SqStack &S,int e)

{

if(S.top-S.base>=S.stacksize)

{

S.base=(int \*)realloc(S.base,(S.stacksize+INCREAMENT)\*sizeof(int));

S.top=S.base+S.stacksize;

S.stacksize+=INCREAMENT;

}

\*S.top++=e;

return 0;

}

int pop(SqStack &S,int &e)

{

if(S.base!=S.top)

e=\*--S.top;

return 0;

}

int isEmpty(SqStack S)

{

if(S.base==S.top)

return 1;

else

return 0;

}

int convert(SqStack &S,int n)

{

int a=n,b,i=0,t;

do{

b=a%8;

a=a/8;

push(S,b);

}while(a>1);

while(i==0){

pop(S,t);

printf("%d",t);

i=isEmpty(S);

};

return 0;

}

int main()

{

SqStack S;

int n;

InitStack(S);

printf("Please input the number:\n");

scanf("%d",&n);

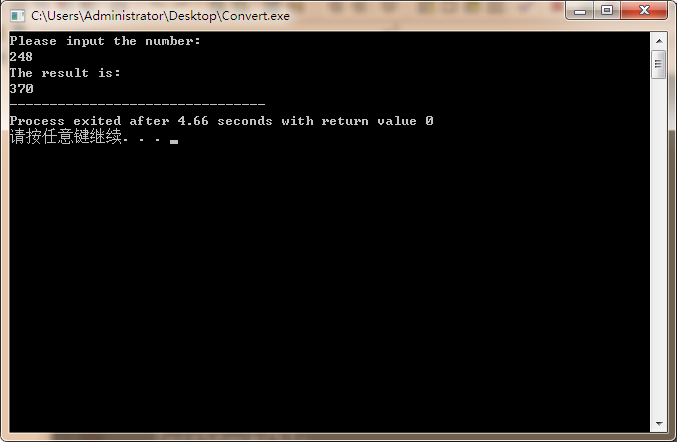
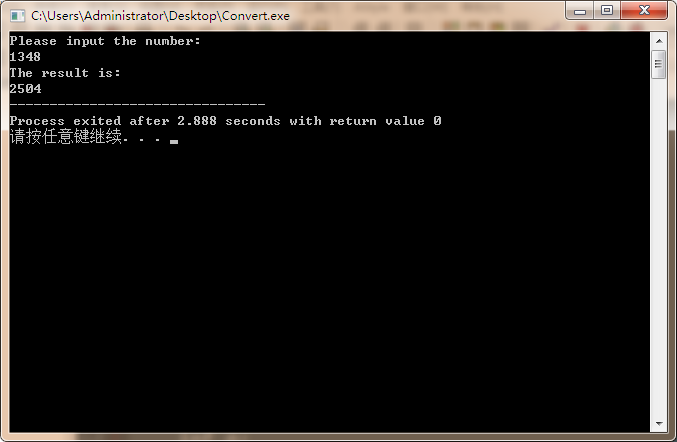
printf("The result is:\n");

convert(S,n);

return 0;

}

截图：



**BracketMatch.cpp:**

#include<stdio.h>

#include<stdlib.h>

#include<iostream>

#define INITSIZE 100

#define INCREAMENT 10

using namespace std;

typedef struct

{

char \*base;

char \*top;

int stacksize;

}SqStack;

int InitStack(SqStack &S)

{

S.base=(char \*)malloc(INITSIZE \* sizeof(char));

S.top=S.base;

S.stacksize=INITSIZE;

return 0;

}

int push(SqStack &S,char e)

{

if(S.top-S.base>=S.stacksize)

{

S.base=(char \*)realloc(S.base,(S.stacksize+INCREAMENT)\*sizeof(char));

S.top=S.base+S.stacksize;

S.stacksize+=INCREAMENT;

}

\*S.top++=e;

return 0;

}

int pop(SqStack &S,char &e)

{

if(S.base!=S.top)

e=\*--S.top;

return 0;

}

int judge(SqStack &S)

{

char c,\_c;

while((c=getchar())!=EOF)

{

if(c==']' || c==')'||c=='[' || c=='(')

{

if(c=='[' || c=='(')

push(S,c);

else if(c==']' || c==')')

{

pop(S,\_c);

if(c==']' && \_c=='(')

{

printf("此串括号匹配不合法\n");

return 0;

}

else if(c==')'&& \_c=='[')

{

printf("此串括号匹配不合法\n");

return 0;

}

}

}

else continue;

}

printf("匹配\n");

return 0;

}

int main()

{

SqStack S;

InitStack(S);

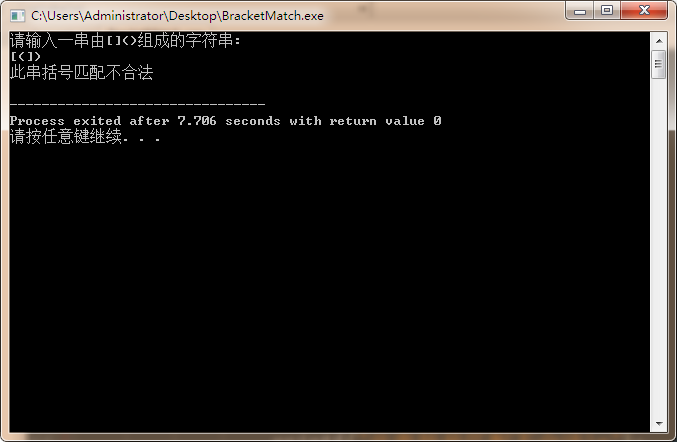
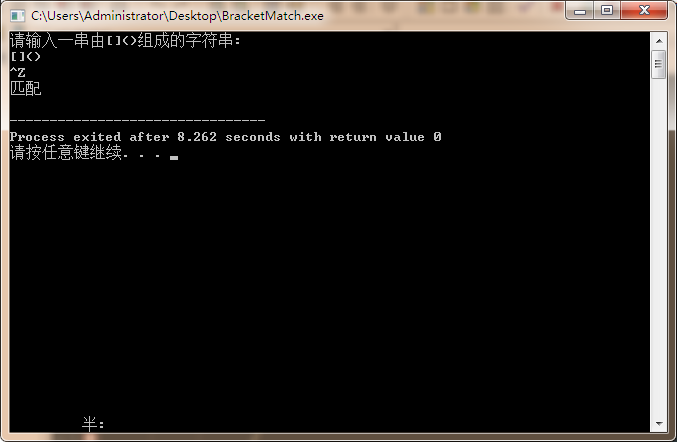
printf("请输入一串由[]()组成的字符串:\n");

judge(S);

return 0;

}

截圖：



注：匹配的情况需要输入Ctrl+Z 结束输入。