**运用栈模拟阶乘函数的调用过程实验报告**

姓名：陆明奇 学号：2050283 日期：2021年11月4日

1. **涉及数据结构和相关背景**

栈是限定仅在表尾进行插入或删除的线性表。栈的修改是按后进先出的原则进行的。因此，栈又称为后进先出的线性表（LIFO结构）。当调用一个函数时，编译器会将参数和返回地址入栈；当函数返回时，这些值出栈。递归通常有两个过程：（1）递归过程：不断递归入栈push，直到停止调用n=1。（2）回溯过程：不断回溯出栈pop, 计算n\*f(n-1)，直到栈空，结束计算。

**2. 实验内容**

**2.1 问题描述**

用栈模拟n的阶乘的递归调用过程。

**2.2 数据结构设计**

此题考察的是用栈模拟n的阶乘的递归调用过程。因此采用栈这一数据结构。

**2.4功能说明（函数、类）**

bool InitStack(sqstack& s, int num)

{

s.base = (int\*)malloc(num \* sizeof(int));

if (!s.base) {

cout << "动态内存申请失败" << endl;

exit(-1);

}

s.top = s.base;

s.stacksize = num;

return true;

}

功能：初始化一个最大容量为num的栈。

Para s: 顺序栈

Para num：栈的最大容量

返回值：初始化栈是否成功

bool push\_stack(sqstack& s, int e)

{

if (s.top - s.base >= s.stacksize){

s.base = (int\*)realloc(s.base, (s.stacksize + stackincrement) \* sizeof(int));

if (!s.base) {

cout << "动态内存申请失败" << endl;

exit(-1);

}

s.top = s.base + s.stacksize;

s.stacksize += stackincrement;

}

\*s.top++ = e;

return true;

}

功能：插入元素e为新的栈顶元素

Para s: 顺序栈

Para e: 插入的元素

返回值：插入元素操作是否成功

bool pop\_stack(sqstack& s, int& e)

{

if (s.top == s.base) {

cout << "栈为空" << endl;

return false;

}

e = \*--s.top;

return true;

}

功能：若栈不空，则删除s的栈顶元素，用e返回其值

Para s: 顺序栈

Para e: 被删除的元素

返回值：删除元素操作是否成功

bool cacl(sqstack& s, int e, long long& res)

{

int k = 1;

res = 1;

while (k <= e){

push\_stack(s, k);

k++;

}

while (s.top != s.base){

int tmp;

pop\_stack(s, tmp);

res \*= tmp;

}

return true;

}

功能：非递归方式计算阶乘

Para s: 顺序栈

Para e: 要计算的数

Para res：e的阶乘

返回值：计算阶乘是否成功

算法复杂度：O（n）

**2.5 调试分析（遇到的问题和解决方法）**

最初各个函数的参数没有传入栈s的引用，导致函数对栈的各个操作都没有实现。之后我把参数改为栈s的引用之后，问题得到解决。

**3.实验总结**

当n超过4000时，递归函数会出现堆栈溢出的错误。用栈消解递归后错误消失，不再出现堆栈溢出的错误。由此可见，当递归的层次过大时，采用递归函数可能会导致堆栈溢出。若我们用自己实现的栈模拟函数递归过程，当递归层次过大时，可以动态地增加我们的栈的容量，不会导致堆栈溢出。