作业 PA3 实验报告

姓名：钱子贤 学号：2054170 日期：2021 年 11 月 7 日

实验报告格式按照模板来即可，对字体大小、缩进、颜色等不做要求

实验报告要求在文字简洁的同时将内容表示清楚

1. 涉及数据结构和相关背景

```
typedef struct {
    int n; //函数的输入参数
    int returnAddress; //函数的返回地址（是否需要，留给大家思考）
    //构造器及getter、setter
    ...
}Data;
```

栈和队列是两种重要的线性结构，从数据结构的角度看，栈和队列也是线性表，其特殊性在于栈和队列的基本操作是线性表的子集。他们是操作受限的线性表，因此，可称为限定性的数据结构。但从数据类型角度看，他们是和线性表大不相同的两类重要的的抽象数据类型。

2. 实验内容

栈的一个重要应用是在程序设计语言中实现递归。所谓递归，是指如果一个对象部分地包括它自己，或用它自己给自己定义，则称这个对象是递归的，或定义为在一个过程中直接或间接地调用自己，则称这个过程是递归的。在调用一个函数(程序)的过程中又直接或间接地调用该函数(程序)本身，称为函数的递归调用。一个递归的求解问题必然包含终止递归的条件，当满足一定条件时就终止向下递归，从而使问题得到解决。描述递归调用过程的算法称为递归算法。在递归算法中，需要根据递归条件直接或间接地调用算法本身，当满足终止条件时结束递归调用。

我的代码中在n=4794时会出现错误，用栈消解递归后不会出错

非最终代码如下：

```
Stack InitStack(Stack* S)/*初始化栈*/

{
    S->base = (int*)malloc(100 * sizeof(int));

    if (!S->base)
        exit(overflow);
    S->top = S->base;
    return *S;
}
```

复杂度O(n)

```
//压栈，循环压栈，因为先进后出，便于取阶乘，倒着压
int push(Stack * s, int n) {
    int i;
    if (s->top >= MAXSIZE) {
        printf("栈有毛病");
        return 0;
    }
    printf("%d\n\n",n);
    for (i = n; i >= 2; i--) {
        printf("i===%d  ",i);
        s->top++;
        s->data[s->top] = i;
        printf("  s->top=====%d   ",s->data[s->top]);
    }
    print_stack(s);
    return 1;
}
```

复杂度O(n)

```
//出栈
int pop(Stack* s) {
    int i;
    long t = 1;
    int p = s->top;
    int pp = 0;
    if (s->top == -1) {
        printf("栈空了");
        return 0;
    }
    while (p != -1) {
        t = t * s->data[p];
        if (s->data[p] % 2 == 0) {
            pp += 2;
            printf("%d! =  %ld   \n", pp, t);
        }
        p--;
    }
    return 1;
}
```

复杂度O(n)

```
//遍历栈
void print_stack(Stack* s) {
```

```
    int t = s->top;
    while (t != -1) {
        printf("%d   ", s->data[t]);
        t--;
    }
    printf("\n");
}
```

复杂度 O(n)

```
请输入要递归的数量:
12
由正常递归得到的数据为：479001600
由运用栈模拟阶乘函数的调用得到的数据为：479001600
```

```
请输入要递归的数量:
31
由正常递归得到的数据为：738197504
由运用栈模拟阶乘函数的调用得到的数据为：738197504
```

```
请输入要递归的数量:
4794
由正常递归得到的数据为:
D:\计算机\数据结构\Debug\调试.exe (进程 15524)已退出，
按任意键关闭此窗口. . .
```

有许多实际问题采用递归方法来解决，使用递归的方法编写程序将使许多复杂的问题大大简化。例如计算求 n 的阶乘问题，可以利用阶乘的递推公式 n!=n*(n-1)!，对该问题进行分解，把计算 n 的阶乘问题化为等式右边涉及规模较小的同类问题(n-1)的阶乘的计算。

例 用递归函数求解正整数 n 的阶乘(n!)

设 f(n)=n!，则递归函数 f(n)

此处 n=0 为递归终止条件，使函数返回 1；当 n>0 时实现递归调用，由 n 的值乘以 f(n-1)的返回值求出 f(n)的值。

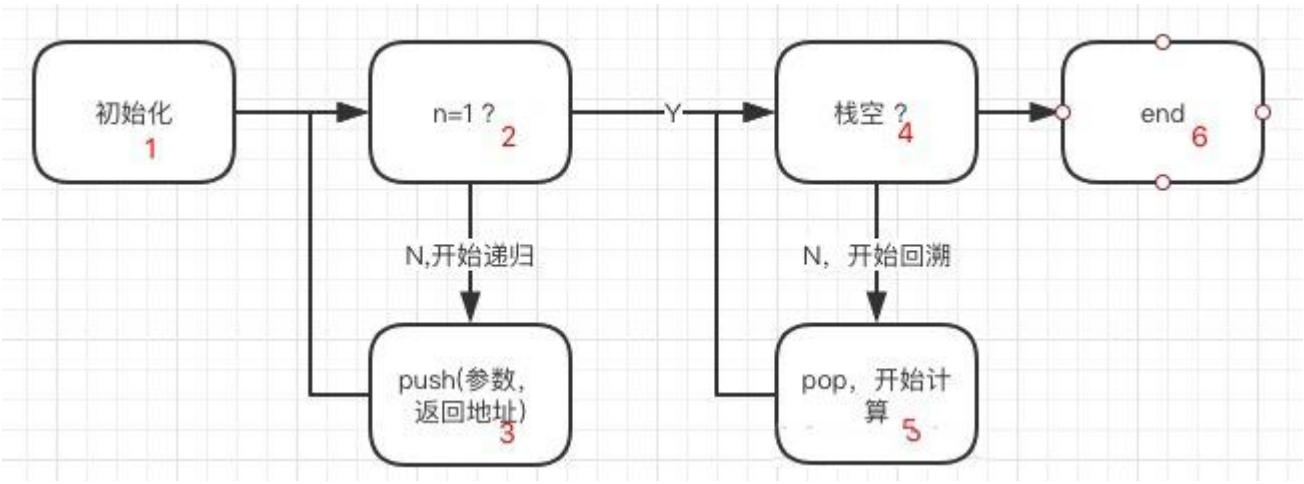
上述递归函数用 c 语言描述为：

```
int f(int n){ if(n==0) return 1; else return n*f(n-1);}
```

可见，递归算法设计的原则是用自身的简单情况来定义自身，使其一步比一步更简单，直至终止条件。设计递归算法的方法是：

(1)寻找递归的通式，将规模较大的原问题分解为规模较小、但具有类似于原问题特性的子问题。即较大的问题递归地用较小的子问题来描述，解原问题的方法同样可用来解这些子问题(例如 n!=n*(n-1)!)。

(2)设置递归出口，确定递归终止条件(例如求解 n!时，当 n=0 时，f(0)=1)。



为什么要学习递归与非递归的转换的实现方法?

1)并不是每一门语言都支持递归的.

2)有助于理解递归的本质.

3)有助于理解栈,树等数据结构.

```
//递归压栈
while (n != 0)
{
    Push(Stack, n);
    n--;
}

//递归出栈
while (Pop(Stack, &e))
{
    Result *= e;
}
```

3. 实验总结

递归与非递归的转换基于以下的原理:所有的递归程序都可以用树结构表示出来.需要说明的是，这个"原理"并没有经过严格的数学证明. 有三种方法可以遍历树:前序,中序,后序.理解这三种遍历方式的递归和非 递归的表达方式是能够正确实现转换的关键之处,所以我们先来谈谈这个.需要说明的是,这里以特殊的 二叉树来说明,不过大多数情况下二叉树已经够用,而且理解了二叉树的遍历,其它的树遍历方式就不难了.

递归函数调用时，是按照“后调用先返回”的原则处理调用过程，如上述求阶乘的递归函数调用，最后调用的是 f(0)，因而最先返回 f(0)的值。因此执行递归函数是通过具有后进先出性质的栈来实现的。系统将整个程序运行时所需的数据空间安排在一个栈中，每当调用一个函数时就为它在栈顶分配一个存储空间，而每当从一个函数退出时，就释放它的存储区。

递归实际上是在执行到递归函数时，将现有的函数中现场保存入栈中，执行完函数后，

恢复现场。于是在处理这类问题是，关键便是思考那些数据是必须要存入栈中的，这点非常的重要。(其实什么程序，预先思考都是非常重要的)

总代码如下:

```
#include<iostream>
#include <stack>
using namespace std;
int fact1(int n)
{
    if (n == 1) return 1;
    else return n * fact1(n - 1);
}
int fact2(int n)
{
    int ret;
    struct data
    {
        int a0;
        int v0;
        int ra;
    };
    stack<data> S;
    S.push({ n,0,0 });//递归栈初始化，ra=0表示先进入代码块0
    while (!S.empty())
    {
        data now = S.top(); S.pop();
        int a0 = now.a0;
        int ra = now.ra;
        int v0 = now.v0;
        switch (ra)
        {
            case 0:
            {
                if (a0 == 1)//递归结束条件
                {
                    S.push({ a0,1,1 });//此条语句也可省略
                    ret = 1;//将本层返回值传给ret
                }
                else
                {
                    S.push({ a0,v0,1 });//本层入栈，排队准备进入代码块1
                    S.push({ a0 - 1,v0,0 });//下一层入栈，马上执行代码块0
```

```
                }
                continue;
            }
            case 1:
            {
                v0 = a0 * ret;//将下层的返回值和自己的a0相乘并变为自己的返回值
                ret = v0;//将自己的返回值赋给ret，供自己的上一层使用
                continue;//不会有push语句，对应正常递归里面的本层执行完毕就注销内存
            }
        }
    }
    return ret;
}
int main(void)
{
    int n;
    cout << "请输入要递归的数量：" << endl;
    cin >> n;
    cout << "由正常递归得到的数据为：";
    int n1 = fact1(n);
    cout << n1 << endl;
    cout << "由运用栈模拟阶乘函数的调用得到的数据为：";

    int n2 = fact2(n);
    cout << n2 << endl;


}
```