实验1 递归算法实现

姓名：叶倩琳

班级：软工1706

完成时间：2019/03/06

**问题一**

1. **问题描述**

实现 fibonacci 的递归和非递归方法。要求计算F（100）的值，比较两种方法的性能。要求1）有合适的提示从键盘输入数据；例如“Please input the number to calculate:”2）有输出结果

1. **代码实现**

#include <iostream>

#include<string>

using namespace std;

//递归法求斐波那契

long long int function1(int n){

if(n==1 || n==0) return 1;

else

return function1(n-1)+function1(n-2);

}

//非递归求斐波那契（STL）

long long int function2(int n){

if(n==1 || n==0) return 1;

else{

long long int i,num1=1,num2=1,cmp=0;

for(i=2;i<=n;i++){

cmp=num1+num2;

num1=num2;

num2=cmp;

}

return cmp;

}

}

int main(int argc, char\*\* argv) {

cout<<" Please input the number to calculate: ";

int n;

while(cin>>n){

cout<<" case1: 使用非递归方法 fibonacci( "<<n<<" ) = "<<function2(n)<<endl;

cout<<" case2: 使用递归的方法 fibonacci( "<<n<<" ) = "<<function1(n)<<endl;

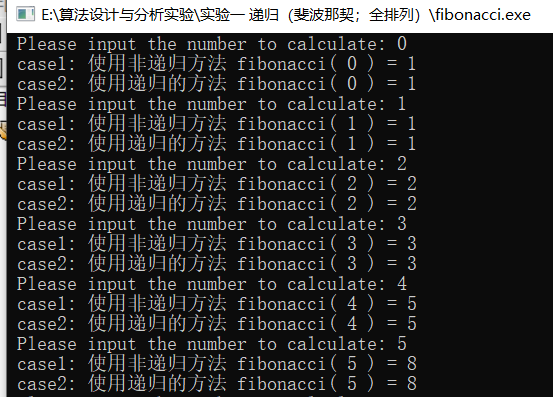
cout<<" Please input the number to calculate: ";

}

return 0;

}

1. **输出结果**





1. **性能分析**

用递归算法实现时，如果要求 fibonacci(n),必须先计算 fibonacci(n-1)和fibonacci(n-2),计算 fibonacci(n-1)和 fibonacci(n-2)，又必须先计算 fibonacci(n-3)和 fibonacci(n-4)。以此类推，直至必须先计算 fibonacci(1)和 fibonacci(0),然后逆推得到 fibonacci(n)，递归算法在时间上造成了很大的浪费，算法的时间复杂度随着N的增大呈现指数增长，时间的复杂度为O(2^n)，这个效率是非常低的，从而导致计算 fibonacci(100)时程序崩溃。

但用非递归实现时，通过for循环不断求 fibonacci(n-1)和 fibonacci(n-2)两数相加的值，它的时间复杂度即for循环的次数，为O(n)。因此可以求出 fibonacci(100)的值。

1. **实验体会**

总的来说，递归算法的效率很低，我们一般采用把递归算法转化为非递归求解，降低问题的复杂度。

**问题二**

1. **问题描述**

假设有输入数字串“123456”，实现对数串全排列问题的递归与循环方法。比较两种方法的程序运行性能并分析复杂度差别。

1. **代码实现**

#include<iostream>

#include<cstring>

#include<algorithm>

using namespace std;

//递归计算全排列

void Perm(char\* a, int k, int m) {

int i,j;

if (k == m) { //只剩下一个元素

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

cout<<a[i];

cout<<endl;

}

else { //还有多个元素待排序，递归产生排序

for (j = k; j <= m; j++) {

swap(a[j], a[k]); //从固定的数后第一个依次交换

Perm(a, k + 1, m);

swap(a[j],a[k]); //此时须将之前交换过的值换回来，保证后序的排列顺序

}

}

}

//非递归计算全排列（STL）

void permutation(char\* str,int length)

{

sort(str,str+length); //先对字符串进行排序

while(next\_permutation(str,str+length)){ //运用STL函数

for(int i=0;i<length;i++)

cout<<str[i];

cout<<endl;

}

}

int main() {

char \*s=new char[100];

cout<<" 请输入要进行排列的字符串：";

cin>>s;

int n=strlen(s); //字符串长度

cout<<" 递归算法的全排列结果为："<<endl;

Perm(s,0,n-1);

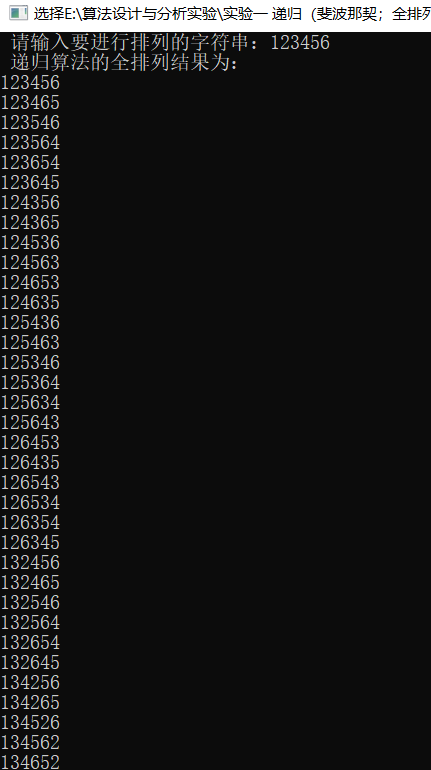
cout<<" 非递归算法的全排列结果为："<<endl;

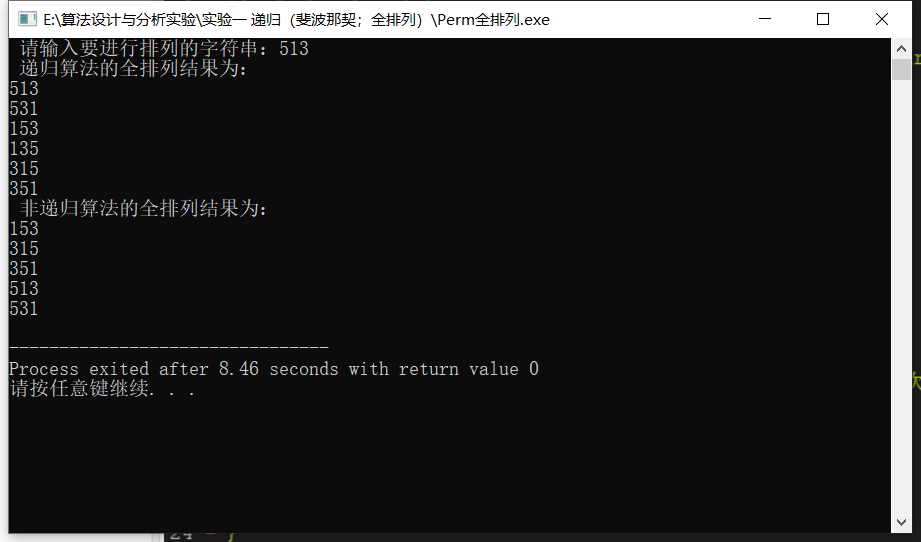
permutation(s,n);

return 0;

}

1. **输出结果**





1. **性能分析**

使用递归实现时，将a[k,m]中的每一个元素分别与a[k]交换，然后递归地计算a[k+1,m]的全排列，将计算结果作为a[0,k]的后缀。n个数的全排列一共有n！种，所以它的复杂度就是排列组合总数，即 O(n!)。如果要对全排列进行输出，则时间复杂度为 O(n\*n!)。

使用非递归实现时，采用了STL中的函数next\_permutation()直接求出全排序，性能高。

1. **实验体会**

递归的效率很低，因此一般会把递归算法转化为非递归形式的算法。本题中求全排列可以采用STL中的函数next\_permutation()，它的作用是假设对于一个序列，存在依照字典排序后这个排列的下一个排列，那么就返回true且产生这个排列，否则返回false。为了产生全排列，要先对序列sort，使其有序。

所以，用标准库算法代替递归也是一种不错的选择。