数据结构实验报告2

学号：2022114467 姓名：林泽宇 专业：软件工程

知识范畴：栈与队列 完成日期：2023年 10 月 3 日

实验题目：用堆栈消除递归

|  |  |  |  |
| --- | --- | --- | --- |
| 课程目标1（60%） | 课程目标2（40%） | 得分(100分)  期末成绩占比5% | 批阅人签字 |
| 程序代码及测试 | 写作、排版、代码注释等 |  |
|  |  |  |

实验内容及要求：

已知某递归函数定义如下：

f(0)=0

f(1)=1

f(2n)=f(n)

f(2n+1)=f(n)+f(2n-1)

编写程序A，定义递归函数f(m)，计算并输出f(0), f(1), …, f(39)的值到字符文件a.txt中（要求每输出8个数换行）。

编写程序B，用堆栈消除递归实现非递归的函数f(m)，调用非递归的f(m)计算并输出f(0), f(1), …, f(39)的值到字符文件b.txt中（输出格式与a.txt相同）。

验证b.txt是否与a.txt完全相同。

实验目的：掌握堆栈操作以及用堆栈消除递归的方法。

数据结构设计简要描述：

采用顺序栈实现, 每个栈包括指向整型的栈顶指针和栈底指针和一个栈容量

typedef struct

{

    int \*top;

    int \*base;

    int stacksize;

} sqStack;

算法设计简要描述：

程序A采用递归算法实现；

程序B使用堆栈结构消除递归；先定义一个数组存储已知量，后往栈内推入待求量；如果待求量无法求得，继续推入未知量；如果未知量可以计算则弹出。直到栈空。

输入/输出设计简要描述：

没有输入；

输出共8行，每行有5个位宽为6的数据，表示f(i);

编程语言说明：

使用Visual C++编程。 主要代码采用C语言实现 ；动态存储分配采用C++的new和delete操作符实现；输入与输出采用C++的cin和cout流；程序注释采用C/C++规范。

主要函数说明：

Status initialStack(sqStack &s) //初始化链表

Status push(sqStack &s, int e) //向栈s压入一个元素e

Status pop(sqStack &s, int &e) //从栈s中弹出一个元素并赋值给e

Status getTop(sqStack &s, int &e) //取栈顶但不弹出, 赋值给e

程序测试简要报告：

图形用户界面, 应用程序, Teams

描述已自动生成

经过对比，a.txt与b.txt完全相同。

源程序代码：

程序A：

#include <iostream>

#include <fstream>

#include <iomanip>

using namespace std;

ofstream outputFile;

int f(int x) //递归没什么好说的

{

    if (!x)

        return 0;

    else if (x == 1)

        return 1;

    else if (x % 2)

        return f(x / 2) + f(x - 2);

    else

        return f(x / 2);

}

int main()

{

    outputFile.open("a.txt");

    for (int i = 0; i <= 39; ++i)

    {

        outputFile << setw(6) << f(i);

        if (i % 5 == 4)

            outputFile << '\n';

    }

    outputFile.close();

    return 0;

}

程序B：

#include <iostream>

#include <fstream>

#include <iomanip>

#include <string.h>

using namespace std;

#define Stack\_init\_size 100

#define Stackincrement 10

#define Status int

#define OK 0

#define OVERFLOW -2

#define ERROR -1

ofstream outputFile;

int num[45];

typedef struct

{

    int \*top;

    int \*base;

    int stacksize;

} sqStack;

Status initialStack(sqStack &s) //初始化链表

{

    s.base = new int[Stack\_init\_size];

    if (!s.base) //避免内存分配失败

        exit(OVERFLOW);

    s.top = s.base;

    s.stacksize = Stack\_init\_size;

    return OK;

}

Status push(sqStack &s, int e) //向栈s压入一个元素e

{

    if (s.top - s.base >= s.stacksize) //避免上溢

    {

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

        s.top = s.base + s.stacksize;

        s.stacksize += Stackincrement;

    }

    \*s.top++ = e;

    return OK;

}

Status pop(sqStack &s, int &e) //从栈s中弹出一个元素并赋值给e

{

    if (s.top == s.base) //避免下溢

        return ERROR;

    e = \*--s.top;

    return OK;

}

Status getTop(sqStack &s, int &e) //取栈顶但不弹出, 赋值给e

{

    if (s.top == s.base) //避免下溢

        return ERROR;

    e = \*(s.top - 1);

    return OK;

}

int main()

{

    outputFile.open("b.txt");

    memset(num, -1, sizeof num);

    num[0] = 0, num[1] = 1; //已知量只有f(0)和f(1), 其余暂定为-1

    sqStack s;

    initialStack(s);

    push(s, 39); //推入未知量

    while (s.top != s.base) //当栈不为空

    {

        int tmp;

        getTop(s, tmp);

        if (tmp % 2)

        {

            int f1 = tmp / 2, f2 = tmp - 2;

            if (num[f1] == -1 || num[f2] == -1) //若不能计算则继续推入未知量

                push(s, f1), push(s, f2);

            else //可以计算则弹出并记录

            {

                pop(s, tmp);

                num[tmp] = num[f1] + num[f2];

            }

        } // tmp为奇数

        else

        {

            int f = tmp / 2;

            if (num[f] == -1) //不能计算推入

                push(s, f);

            else //可以计算弹出

            {

                pop(s, tmp);

                num[tmp] = num[f];

            }

        } // tmp为偶数

        if (num[tmp - 1] == -1)

            push(s, tmp - 1);

    } // while

    for (int i = 0; i <= 39; ++i)

    {

        outputFile << setw(6) << num[i];

        if (i % 5 == 4)

            outputFile << '\n';

    } // for

    outputFile.close();

    return 0;

}