《软件设计实践 1》课程实验报告

课程名称：软件设计实践 1

实验名称：数据结构与算法基础实验

任务名称：任务 1（基础编程）

班级：计算机 16　姓名：詹威霖　学号：166001705

评分标准：

（1）作业完成度；

（2）叙述清晰、图文并茂；

（3）代码注释到位、可读性和程序风格良好；

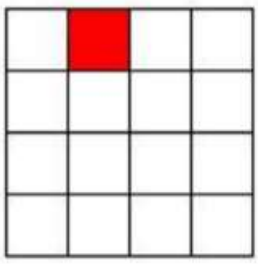
（4）分析说明（问题、错误、改进、难点解决方案等）等叙述清晰；

（5）内容有扩展，能发现和解决问题，探究能力强。

[题目一]

一、实验任务和要求

在一个 $2^k \times 2^k$ 个方格组成的棋盘中，恰有一个方格与其它方格不同，称该方格为一特殊方格，且称该棋盘为一特殊棋盘。在棋盘覆盖问题中，要用图示的 4 种不同形态的 L 型骨牌覆盖给定的特殊棋盘上除特殊方格以外的所有方格，且任何 2 个 L 型骨牌不得重叠覆盖。



提示：分解成 2*2，保证每个 2*2 中有一个在上一步被填充

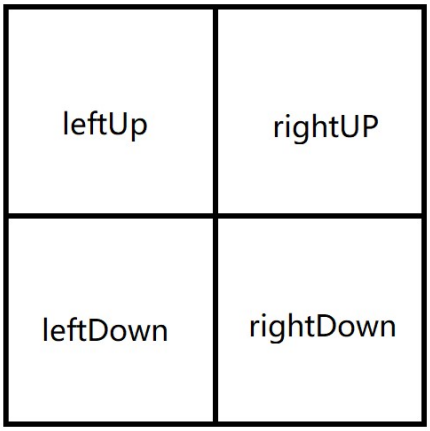
要求用 1,2,3 等数字分别表示第 1/2/3 步组合的 L 型骨牌。

二、实验内容和步骤（或程序源码/含注释说明）

Ⅰ、解题思路及说明

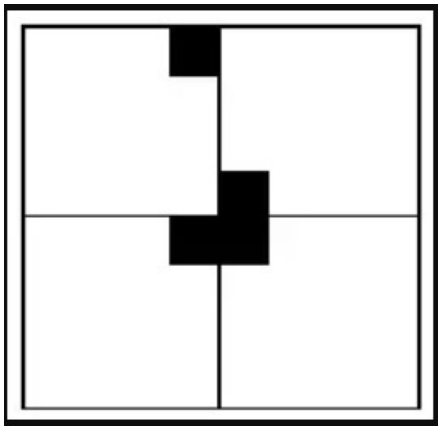
用分治策略，可以设计解棋盘问题的一个简捷的算法。

当 k>0 时，将 2^k * 2^k 棋盘分割为 4 个子棋盘，如下图所示：



特殊方格必位于 4 个较小子棋盘之一中，其余 3 个子棋盘中无特殊方格。

为了将这 3 个无特殊方格的子棋盘转化为特殊棋盘，我们可以用一个 L 型骨牌覆盖这 3 个较小的棋盘的汇合处，如下图所示，这 3 个子棋盘上被 L 型骨牌覆盖的方格就成为该棋盘上的特殊方格，从而将原问题化为 4 个较小规模的棋盘覆盖问题。



递归的使用这种分割，直至棋盘简化为 1x1 棋盘。

三、源码及注释

```
#include <bits/stdc++.h>

using namespace std;

enum position {//位置信息枚举类，分别代表左上角，右上角，左下角，右下角
    leftUp, rightUp, leftDown, rightDown
};
```

```
class Grid {//网格类，用于生成棋盘的布局
private:
    int k;//网格的大小为 2^k
    vector<vector<int> > grid;//使用二维向量存储
public:
    Grid(int _k) {//构造函数
        k = _k;
        grid.resize((1 << k) + 1);
        for (int i = 0; i < (1 << k) + 1; ++i) {
            grid[i].resize((1 << k) + 1);
        }
    }
    //因为是私有变量，所以需要以下函数设置信息
    int getK() {
        return k;
    }

    void set(int x, int y, int value) {
        grid[x][y] = value;
    }
    //输出棋盘信息
    void prt() {
        for (int i = 1; i <= (1 << k); i++) {
            for (int j = 1; j <= (1 << k); ++j) {
                printf("%10d", grid[i][j]);
            }
            putchar('\n');
        }
    }
};
//将一个棋盘分为四部分，根据其左上角坐标以及相应的 k 值来定位当前点在哪个部分
position locate(int x, int y, int k, int sx, int sy) {
    int width = 1 << k;
    int halfX = width / 2 + sx - 1;
    int halfY = width / 2 + sy - 1;
    int X = sx + width - 1;
    int Y = sy + width - 1;
    if (sx <= x && x <= halfX && sy <= y && y <= halfY) {
        return leftUp;
    } else if (sx <= x && x <= halfX && halfY < y && y <= Y) {
        return rightUp;
    } else if (halfX < x && x <= X && sy <= y && y <= halfY) {
        return leftDown;
```

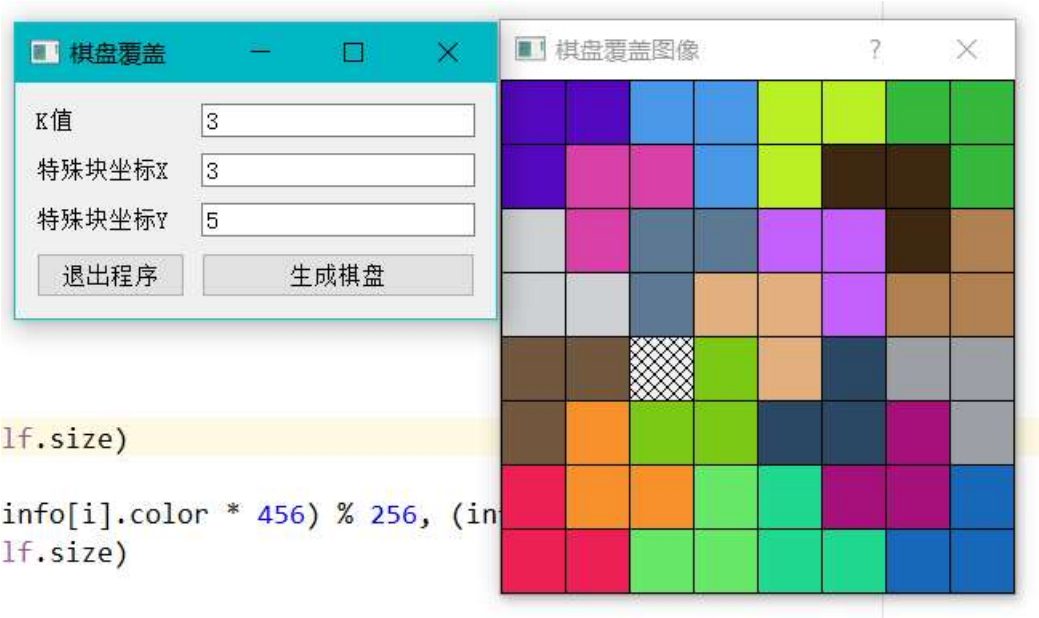
```
    } else if (halfX < x && x <= X && halfY < y && y <= Y) {
        return rightDown;
    }
}

int color = 1;
//设置覆盖块的颜色
void setColor(Grid &g, position pos, int mx, int my) {
    if (pos != leftUp)
        g.set(mx, my, color);
    if (pos != rightUp)
        g.set(mx, my + 1, color);
    if (pos != leftDown)
        g.set(mx + 1, my, color);
    if (pos != rightDown)
        g.set(mx + 1, my + 1, color);
    color++;
}
//递归进行画图
void Draw(Grid &g, int x, int y, int k, int sx, int sy) {
    position pos = locate(x, y, k, sx, sy);//判断当前坐标象限
    if (k == 0)return;
    int half = (1 << k) / 2;
    int mx = sx + half - 1;
    int my = sy + half - 1;
    switch (pos) {
        case leftUp://如果当前在左上角
            Draw(g, x, y, k - 1, sx, sy);//对自身这个部分递归
            setColor(g, pos, mx, my);//根据算法将其他三块染色
            Draw(g, mx, my + 1, k - 1, sx, my + 1);//染右上角区域
            Draw(g, mx + 1, my, k - 1, mx + 1, sy);//染左下角区域
            Draw(g, mx + 1, my + 1, k - 1, mx + 1, my + 1);//染右下角区域
            break;
        case rightUp://如果当前在右上角
            Draw(g, x, y, k - 1, sx, my + 1);//对自身这个部分递归
            setColor(g, pos, mx, my);//根据算法将其他三块染色
            Draw(g, mx, my, k - 1, sx, sy);//染左上角区域
            Draw(g, mx + 1, my, k - 1, mx + 1, sy);//染左下角区域
            Draw(g, mx + 1, my + 1, k - 1, mx + 1, my + 1);//染右下角区域
            break;
        case leftDown:
            Draw(g, x, y, k - 1, mx + 1, sy);//对自身这个部分递归
            setColor(g, pos, mx, my);//根据算法将其他三块染色
            Draw(g, mx, my, k - 1, sx, sy);//染左上角区域
```

```
            Draw(g, mx, my + 1, k - 1, sx, my + 1);//染右上角区域
            Draw(g, mx + 1, my + 1, k - 1, mx + 1, my + 1);//染右下角区域
            break;
        case rightDown:
            Draw(g, x, y, k - 1, mx + 1, my + 1);//对自身这个部分递归
            setColor(g, pos, mx, my);//根据算法将其他三块染色
            Draw(g, mx, my, k - 1, sx, sy);//染左上角区域
            Draw(g, mx, my + 1, k - 1, sx, my + 1);//染右上角区域
            Draw(g, mx + 1, my, k - 1, mx + 1, sy);//染左下角区域
            break;
    }
}


int main() {
    Grid g = Grid(3);
    int sx = 1;
    int sy = 2;
    Draw(g, sx, sy, g.getK(), 1, 1);
    g.prt();
    return 0;
}
```

四、实验结果



五、实验过程（调试、难点及解决方法）

中间定位某个点在哪个区域的逻辑有点繁琐，需要细致的调试，题目难点主要在于如何用程序实现分治的过程，我是用递归解决这个问题的。我把此题图形界面化。

六、实验拓展（在基本任务基础上，做的功能扩展和改进）

画出如下所示的分形图案

```
X  X   X  X
 X     X
X  X   X  X
   X  X
     X
   X  X
X  X    X  X
  X      X
X  X    X  X
```

实验结果：

```
#include <stdio.h>
#include <string.h>
int p[8] = {1,3,9,27,81,243,729};
char map[730][730];
//n 当前的图形大小，x，y 图形所在的坐标
void print(int n,int x,int y){
    if(n == 0){
        map[x][y] = 'X'; //
        return;
    }
    print(n-1, x, y); //左上
    print(n-1, x+2*p[n-1], y); //右上
    print(n-1, x+p[n-1], y+p[n-1]); //中间
    print(n-1, x, y+2*p[n-1]);
    print(n-1, x+2*p[n-1], y+2*p[n-1]);
}
int n;
int main(){
freopen("out","w",stdout);
    for(int i=0; i<p[6]; i++) memset(map[i], 32, p[6]);
    print(6, 0, 0); //打表
    while(scanf("%d", &n) && n-- >= 0){
        for(int i=0; i<p[n]; i++){
                map[i][p[n]]=0;
                puts(map[i]);
```

```
                map[i][p[n]]=' ';
        }
        puts("-");
    }
    return 0;
}
```

[题目二]

一、实验任务和要求

给定一个矩阵 k，产生如下的数据填充（下图为 k=5 时）

1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9

二、实验内容和步骤（或程序源码/含注释说明）

Ⅰ、解题思路及说明

使用一重循环完成这题，只需要通过三个变量即可完成，当前节点的坐标，以及当前节点的状态，先试图一直往一个方向走，当无法再往一个方向走时，转换方向，即可完成，详情可以看源码注释。

三、源码及注释

```
#include <iostream>

using namespace std;

const int MAXN = 100;//矩阵最大大小
int mx, my;//当前矩阵大小
int matrix[MAXN][MAXN];//初始值均为0

bool judge(int i, int j) {//判断当前节点是否是有效的节点
    if (i < 1 || j < 1 || i > mx || j > my)return false;//越界显然节点无效
    return matrix[i][j] == 0;//如果该节点值为0，那么节点有效，否则节点无效
}
/*
 * status 表示当前节点状态，0表示向右走，1表示向下走，2表示向左走，3表示向上走，
 *  i，j表示当前节点坐标
```

```
 * */
void change(int &status, int &i, int &j) {
    if (status == 0) {//当前为往右走
        if (judge(i, j + 1)) {//试图继续往右走，如果成功，则继续向右
            j++;
        } else {//无法继续向右走，那么就往下走，把当下节点状态改变成向下走，取余表示循环
            i++;
            status = (status + 1) % 4;
            return;
        }
    }
    if (status == 1) {//当前为往下走
        if (judge(i + 1, j)) {//试图继续往下走，如果成功，则继续向下
            i++;
        } else {//无法继续向下走，那么就往左走，把当下节点状态改变成向左走，取余表示循环
            j--;
            status = (status + 1) % 4;
            return;
        }
    }
    if (status == 2) {//当前为往左走
        if (judge(i, j - 1)) {//试图继续往左走，如果成功，则继续向左
            j--;
        } else {//无法继续向左走，那么就往上走，把当下节点状态改变成向上走，取余表示循环
            i--;
            status = (status + 1) % 4;
            return;
        }
    }
    if (status == 3) {//当前为往上走
        if (judge(i - 1, j)) {//试图继续往上走，如果成功，则继续向上
            i--;
        } else {//无法继续向上走，那么就往右走，把当下节点状态改变成向右走，取余表示循环
            j++;
            status = (status + 1) % 4;
            return;
        }
    }
}
```

```
int main() {
    mx = 10;
    my = 10;
    int num = 1;
    int status = 0, i = 1, j = 1;
    for (int step = 1; step <= mx * my; ++step) {//对于 mx*my 大小的矩阵，只需走 mx*my 步即可
        matrix[i][j] = num++;
        change(status,i,j);
    }
    for (int k = 1; k <= mx; ++k) {//输出该矩阵
        for (int l = 1; l <= my; ++l) {
            printf("%-3d", matrix[k][l]);
        }
        cout << endl;
    }
    return 0;
}
```

四、实验结果

```
[1, 2, 3, 4, 5]
[16, 17, 18, 19, 6]
[15, 24, 25, 20, 7]
[14, 23, 22, 21, 8]
[13, 12, 11, 10, 9]
```

五、实验过程（调试、难点及解决方法）

本题的编写较为简单，用一个 for 循环的话需要记录当前状态和坐标，然后就是一个很简单的判断逻辑了。

六、实验拓展（在基本任务基础上，做的功能扩展和改进）

完成可视化

[题目三]

一、实验任务和要求

给定一个整数的数组，找出这样的两个数，这两个整数的和加起来等于一个特定 的整数 target。要求尽可能降低复杂度。提示：哈希方法

二、实验内容和步骤（或程序源码/含注释说明）

我们限制整数的范围为 0~10000000，这样我们就可以用数组下标作为最简

单的哈希化方法，即可在 O（n）复杂度内完成。当数据大于 1e6，可以考虑使用 map 这个 STL 的实现方法（其实也是一种哈希化），或者考虑先将数据进行排序，然后二分搜索答案，源码中实现了这两种方法。

三、源码及注释

```
#include <iostream>
#include <algorithm>

using namespace std;
const int MAXN = 1e6 + 5;
//讨论数据大小，若数据大小在 1e6 以内，哈希化，即可 O（n）复杂度实现
//若数据大于 1e6，则考虑使用 map 做法,或者排序加二分方法

int f[MAXN];
int origin[MAXN];

pair<int, int> solve1(int n, int target) {
    for (int i = 0; i < n; ++i) {
        f[origin[i]]++;//最简单的哈希化，即其本身出现次数++
    }
    for (int i = 0; i < n; ++i) {
        if (f[target - origin[i]]) {//对于每一个数组中的元素，如果（目标和减去该数）存在（即另一个加数）存在
            if (2 * origin[i] == target) {
                if (f[origin[i]] >= 2) {//如果是这个数字本身，那么这个数字要至少出现 2 次
                    return make_pair(origin[i], origin[i]);
                }
            } else {//如果不是这数字本身，这则代表已经找到这样的数对，那么直接返回即可
                return make_pair(origin[i], target - origin[i]);
            }
        }
    }
}

pair<int, int> solve2(int n, int target) {
    sort(origin, origin + n);//对数组排序
    for (int i = 0; i < n; ++i) {
        if (*lower_bound(origin, origin + n,origin[i]) == target - origin[i])//在数组中二分查找另一个加数，找到则直接返回
            return make_pair(origin[i], target - origin[i]);
    }
```

```
}


int main() {
    int n;
    cin >> n;
    for (int i = 0; i < n; ++i) {
        cin >> origin[i];
    }
    return 0;
}
```

四、实验结果

```
请输入整数个数：40

随机生成的整数序列为：
25295 8413 26835 21439 12033 23515 8537 5355 13701 19321 9538 31786 25878 21015 443 1286 8684 32408 5996 9625 389 26902
21592 12520 29605 18387 17668 13170 31900 20595 16603 32264 25799 14697 13777 21214 345 28592 22343 26851

请输入整数target：788

和为788的整数对为：
345 443
```

五、实验过程（调试、难点及解决方法）

经典的两数之和问题，解决办法可以通过排序，哈希等方法进行搜索。

[题目四]

一、实验任务和要求

两组数据，用结构体表示，每个节点只放一个数字：

```
//Definition for singly-linked list.
struct ListNode
{
    int val;
    struct ListNode *next;
};
```

Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)

Output: 7 -> 0 -> 8

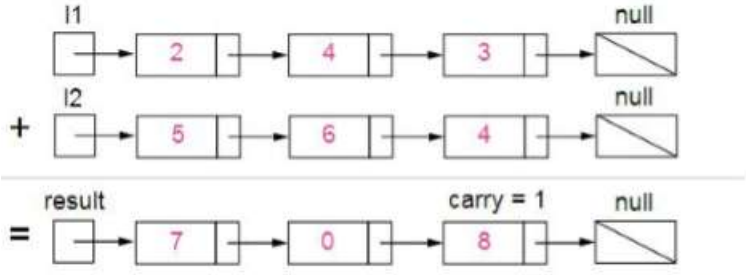


Figure 1. Visualization of the addition of two numbers: 342 + 465 = 807.

二、实验内容和步骤（或程序源码/含注释说明）

先用 gets()函数将式子读入，创建两个链表头结点 head1,head2。遍历整个式子在遇到‘+’号前的数字全部按顺序插入第一条链表中，遇到‘+’号后的数字全部按顺序插入第二条链表中。定义 add 为进位标志，初始为 0。同事遍历两条链表，将两数位与进位全部相加，判断结果是否大于 9，如果大于 9，结果要减 10，add=1，如果有一条链表遍历完了，则退出循环。然后分别判断两条链表是否都遍历完了，如果没有，则继续遍历。最后判断进位是否为 1，进位为 1 说明还要输出进位。其实这题就是用链表来实现两个数的加法。

三、源码及注释

```
class Solution {
    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {

        int carry = 0;
        ListNode ans = new ListNode(0);
        ListNode ans1 = ans;
        while (l1 != null && l2 != null) {
            ans.val = (l1.val + carry + l2.val) % 10;
            carry = (carry + l1.val + l2.val) / 10;
            l1 = l1.next;
            l2 = l2.next;
            if (l1 == null && l2 == null) break;
            ans.next = new ListNode(0);
            ans = ans.next;

        }
        while (l1 != null) {
            ans.val = (l1.val + carry) % 10;
            carry = (carry + l1.val) / 10;
            l1 = l1.next;
            if (l1 == null) break;
            ans.next = new ListNode(0);
            ans = ans.next;
        }
        while (l2 != null) {
            ans.val = (l2.val + carry) % 10;
            carry = (carry + l2.val) / 10;
            l2 = l2.next;
            if (l2 == null) break;
```

```
            ans.next = new ListNode(0);
            ans = ans.next;
        }
        if(carry==1){
            ans.next = new ListNode(carry);
        }
        return ans1;
    }
}
```

四、实验结果

我的输入

```
[1,2,3]
[9,9,9]
```

我的答案

```
[0,2,3,1]
```

五、实验过程（调试、难点及解决方法）

代码较为简单，就是最简单的单链表的使用，较为复杂的一点在于考虑到数据的进位信息，以及在边界情况下的处理，例如 1 加 999 这种数据。

六、实验拓展（在基本任务基础上，做的功能扩展和改进）

使用 C 语言实现多个大整数的相加，注意，大整数的范围可能超出 C 语言基本类型的范围。

实验结果：

```
#include <iostream>
#include <stack>
using namespace std;

int main()
{
    int T;
    stack<int> a,b,mid,end;
    cin>>T;
    while(T>0)
    {
```

```
string str;
cin>>str;
int t,len,j,k;
for (j = 0; j <str.length() ; ++j)
    {
        a.push(str[j]-'0');
    }
while(str!="0")
{
    cin>>str;
    for (j = 0; j <str.length() ; ++j)
    {
        b.push(str[j]-'0');
    }
    len=a.size()<=b.size()?a.size():b.size();
    t = 0;
    for (k = len-1; k>=0; --k)
    {
        mid.push((a.top()+b.top()+t)%10);
        if(a.top()+b.top()+t>=10) t = 1;
        else t=0;
        a.pop();b.pop();
    }
    while(!a.empty())
    {
        mid.push((a.top()+t)%10);
        if(a.top()+t>=10)   t=1;
        else t =0;
        a.pop();
    }
    while(!b.empty())
    {
        mid.push((b.top()+t)%10);
        if(b.top()+t>=10)   t=1;
        else t =0;
        b.pop();
    }
    if(t==1)mid.push(1);
    while(!mid.empty())
    {
        a.push(mid.top());
        mid.pop();
    }
}
```

```
        while(!a.empty())
        {
            end.push(a.top());
            a.pop();
        }
        while(!end.empty())
        {
            cout<<end.top();
            end.pop();
        }
        cout<<endl;
        T--;
    }
    return 0;
}
```

运行结果