《数据结构》课程实践报告

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院、系 | 计算机学院 | | 年级专业 | 2018 | 姓名 |  | 学号 |  |
| 实验布置日期 | | 2019.9.13 | | 提交  日期 | 2019.9.20 | | 成绩 |  |

课程实践实验1：生命之数（题目）

## 问题描述及要求

生命游戏在一个无边界的矩形网格上进行，这个矩形网格中的每个单元可被占据，或者不被占据。被占据的单元称为活的，不被占据的单元称为死的。哪一个单元是活的是根据其周围活的邻居单元的数目而一代一代地发生变化的。一代一代转换的具体规则如下：

给定单元的邻居单元指的是与它在垂直、水平或对角方向上相接的8个单元。

如果一个单元是活的，则如果它具有2个或3个活的邻居单元，则此单元在下一代还是活的。

如果一个单元是活的，则如果它具有0个或1个、4个或4个以上的活的邻居单元，则此单元在下一代会因为孤独或拥塞而死亡。

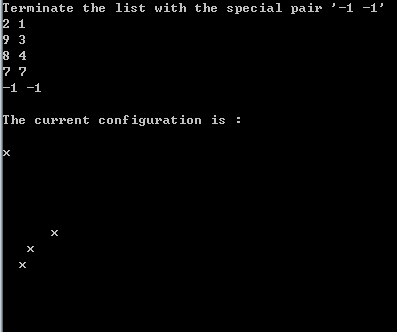
如果一个单元是死的，则如果它具有恰好有3个活的邻居，则此单元在下一代会复活，否则该单元在下一代仍然是死的。

（1）要求编写程序，模拟任意一个初始输入配置以及代代更替的不同状态并进行显示。

（2）修改以上程序,要求生成一个网格时，用“空格”和“x”分别表示网格中每一个死的单元和活的单元，并且可根据用户选择从键盘或者从文件读入初始配置。

## 概要设计

1. 输入表格 初始化表格 更新表格 询问客户是否继续
2. **程序运行的界面设计**



3. 建立一个生成表格的类。在类里分别initialize()构构造初始化表格、print()

将表格化成要求输出和update()更新表格的三个函数。

## 三、详细设计

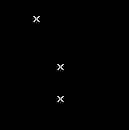
1. 主函数主要是说明引导输入
2. 通过双层循环初始化
3. 通过双层循环输出
4. 询问客户是否继续
5. 继续的话用循环更新
6. 难点是循环

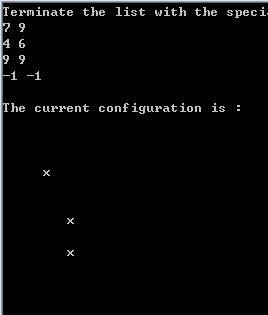
## 四、实验结果

**紧扣题目要求设计提供相应的测试方法和结果。可以给出具体的测试用例，每个测试用例一般可列出：**

**测试输入：7 ，9 4，6 9，9**

**测试目的：随机性**

**正确输出：**

**实际输出：**

**测试结论：通过**

## 六、小结

**学会了用表格循环**

## 附录：源代码

#include <iostream>

using namespace std;

const int maxrow=20, maxcol=60;

class Life

{

int grid[maxrow+2][maxcol+2];

int neighbor\_count(int row,int col)

{

int count=0;

for(int i=row-1;i<=row+1;i++)

for(int j=col-1;j<=col+1;j++)

count+=grid[i][j];

count-=grid[row][col];

return count;

}

public:

void initialize()

{

int r,c;

for(r=0;r<=maxrow+1;r++)

for(c=0;c<=maxcol+1;c++)

grid[r][c]=0;

cout<<"List the coordinates for living cells.\n";

cout<<"Terminate the list with the special pair '-1 -1'\n";

cin>>r>>c;

while(r!=-1||c!=-1)

{

if(r>=1&&r<=maxrow)

{

if(c>=1&&c<=maxcol)

{

grid[r][c]=1;

}

else

cout<<"Colomn "<<c<<" is out of range."<<endl;

}

else

cout<<"Row "<<r<<" is out of range."<<endl;

cin>>r>>c;

}

}

void print()

{

int r,c;

cout<<"\nThe current configuration is :\n";

for(r=1;r<=maxrow;r++)

{

for(c=1;c<=maxcol;c++)

{

if(grid[r][c]==1) cout<<"x";

else cout<<" ";

}

cout<<endl;

}

cout<<endl;

}

void update()

{

int r,c,new\_grid[maxrow+2][maxcol+2];

for(r=1;r<=maxrow;r++)

for(c=1;c<=maxcol;c++)

switch(neighbor\_count(r,c))

{

case 2: new\_grid[r][c]=grid[r][c]; break;

case 3: new\_grid[r][c]=1; break;

default: new\_grid[r][c]=0;

}

for(r=1; r<=maxrow;r++)

for (c=1;c<=maxcol;c++)

grid[r][c]=new\_grid[r][c];

}

};

bool user\_says\_yes()

{

int c;

bool ini\_response=true;

do

{

if(ini\_response) cout<<"y or n?"<<endl;

else cout<<"Type in y or n"<<endl;

do

{

c=cin.get();

} while(c=='\n'||c==' '||c=='\t');

ini\_response=false;

} while (c!='y'&&c!='Y'&&c!='n'&&c!='N');

return (c=='y'||c=='Y');

}

int main()

{

Life configuration;

cout<<"Welcome to Game of Life.\n";

cout<<"This game uses a grid of size "<<maxrow<<" by "<<maxcol<<" in which each\n";

cout<<"cell can either be occupied by an organism or not.\n";

cout<<"The occupied cells change from generation to generation according to how many neighboring cells are alive.\n";

configuration.initialize();

configuration.print();

cout<<"Continue?"<<endl;

while(user\_says\_yes())

{

configuration.update();

configuration.print();

cout<<"Continue?"<<endl;

}

}