苏 州 大 学 实 验 报 告

院、系	计算机学院	年级专业	19 计算机类	姓名	张昊	学号	1927405160
课程名称	数据结构课程实践					成绩	
指导教师	孔芳	同组实验者	无		实验日期	2020 年 10 月 3 日	

实验名称 生命游戏

一、实验目的

通过本次实验要达到如下目的：

1. 熟悉使用 C++解决问题；
2. 复习 C++面向对象知识。

二、实验内容

生命游戏

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。

三、实验步骤和结果

1 Life 类的设计与定义

我们将生命游戏中的每种配置抽象为一个 Life 对象，每个 Life 对象需要包含一个用矩形的二维数组表示的网格 `gird`，用两个成员变量 `row`、`col` 来表示配置的行数和列数，并且有三个公有方法 `initialize`、`update`、`show`，分别为初始化配置、迭代更新配置和输出当前配置这三个基本操作。

在二维数组 `grid` 中，我们用整数 1 表示活的单元，用 0 表示死的单元。所以统计某个特定单元的活邻居的数量，只需累加其所有的邻居单元，并且这个操作在更新 Life 配置时时需要不断重复的。因此，考虑为 Life 类设计一个私有的成员函数 `count` 来实现这一功能。

最后，需要必须确定一个配置最大能容纳的行数和列数，考虑将这些作为类的静态常成

员变量，需要变化时作简单的修改就可以重新设置程序中网格的大小。

综上所述，Life 类的定义如下：（放在文件 life.h 中）

```
1.  class Life {
2.  public:
3.      Life() : row(-1), col(-1) {}
4.      void initialize(std::istream &);
5.      void update();
6.      void show(std::ostream &);
7.  private:
8.      static const int max_row = 128;
9.      static const int max_col = 128;
10.     int grid[max_row + 2][max_col + 2]{};
11.     int row;
12.     int col;
13.     int count(int x, int y);
14. };
```

2　迭代更新的基本操作：统计邻居个数 count

对坐标为 row，col 的单元，统计其邻居时需要累加它的 8 个邻接的邻居单元。算法需要使用两层循环来完成，一层循环从 row-1 遍历到 row+1，在这一层循环中嵌套一层从 col-1 到 col+1 的循环。对于在网格内部的单元不会存在问题，但是需要注意的是，当 row，col 在网格的边界上时，只需要访问网格中合法的单元，即访问不能越界。一种办法是用一些判断语句来确认是否累加时跑出了网格，但是这种办法会增加代码的复杂程度，为 bug 的出现埋下隐患。另外的做法是扩大表示网格这个二位数组的大小，在数组的四周增加额外的两行两列，就像是在网格周围引人一圈栅栏，即 Life 类定义中的 `int grid[max_row + 2][max_col + 2]{};` 。这样一来，对于所有合法的单元，统计其四周 8 个邻接的邻居单元的算法将变得一致。

```
1.  int Life::count(int x, int y) {
2.      int num = 0;
3.      for (int i = x - 1; i <= x + 1; ++i) {
4.          for (int j = y - 1; j <= y + 1; ++j) {
5.              if (!(i == x && j == y)) {
6.                  num += grid[i][j];
7.              }
8.          }
9.      }
10.     return num;
11. }
```

由此在更新配置的方法 update 中，使用新的二位数组 new_grid 记录了更新后的配置。使用两层循环依次统计每个单元的邻居个数。依照题目条件，如果个数为 2，则状态保持不

变；如果个数为 3，则一定会变为活的；其他情况状态均会变为死的。统计完后再将 new_grid 拷贝到 grid 上即可。

```
1.  void Life::update() {
2.      int new_grid[max_row][max_row]{};
3.      for (int i = 1; i <= row; ++i) {
4.          for (int j = 1; j <= col; ++j) {
5.              switch (count(i, j)) {
6.                  case 2:   // keep still
7.                      new_grid[i][j] = grid[i][j];
8.                      break;
9.                  case 3:    // alive
10.                     new_grid[i][j] = 1;
11.                     break;
12.                 default:     // dead
13.                     new_grid[i][j] = 0;
14.             }
15.         }
16.     }
17.     for (int i = 1; i <= row; ++i) {
18.         for (int j = 1; j <= col; ++j) {
19.             grid[i][j] = new_grid[i][j];
20.         }
21.     }
22. }
```

3 配置的初始化

为了能使用户从不同的位置（如标准输入、文件等）输入配置文件，initialize 方法带有一个 std::istream 类型的参数，可以根据用户的不同选择来传入不同的输入流对象，从而实现从不同的位置读取输入配置。initialize 算法读取整个配置字符串，用空格表示死的单元，用字符“x”表示活的单元。算法对输入的是否为矩形、是否包含不合法字符进行检查，如果输入有误，则抛出包含错误信息的异常。

4 菜单驱动的交互程序设计

程序提供给用户两种运行模式。一种是直接运行，程序会提示用户输入由空格和字符“x”组成的配置信息。如图所示：

```
/mnt/d/WorkSpace/Codes/School/DS-Class/LifeGame/Src/cmake-build-debug/LifeGame
Life Game (Oct. 10 2020) By Holger Zhang

Welcome to the game of Life.
This game uses a grid of size 128*128.
Each cell can either be occupied by an organism or not.
The occupied cells change from generation to generation according to how many neighboring cells are alive.

Please input a configuration, multi-line rectangular text consisting of character ' ' and 'x', and end up with '#'
 in the new line:
```

另一种是传入一个命令行参数，为保存了配置信息的文本文件的路径，程序会读取其中包含的配置信息。如图所示：

```
/mnt/d/WorkSpace/Codes/School/DS-Class/LifeGame/Src/cmake-build-debug/LifeGame ./in
Life Game (Oct. 10 2020) By Holger Zhang

Welcome to the game of Life.
This game uses a grid of size 128*128.
Each cell can either be occupied by an organism or not.
The occupied cells change from generation to generation according to how many neighboring cells are alive.

Reading configuration from file './in'
Got configuration:

   x
   x
   x
```

为实现菜单驱动的功能，设计了以下辅助函数，并放在命名空间 `console_runtime` 中。（放在文件 console_runtime.h 中）

```
1.  namespace console_runtime {
2.      // 等待用户输入选项
3.      char wait_user(const string &info="wait for input");
4.      // 捕获到错误信息后，输出错误信息
5.      void show_error(exception &error,
6.                      const string &state,
7.                      const string &file="<stdin>",
8.                      bool terminal=false,
9.                      int exit_code=0);
10.     // 根据用户的选项调用 Life 类相应方法
11.     void execute(Life &life, char option);
12. }
```

并设计菜单如下：

```
-----------------------------
| -*-*-*- Life Game -*-*-*- |
|========= M E N U =========|
|[s] Show the configuration |
|[n] Next status            |
|[q] Quit the game          |
|[?] Show this menu again   |
-----------------------------
```

其中等待用户输入选项的函数 `wait_user` 输出提示信息并等待从标准输入读取，返回读取到的第一个字符；输出错误信息的函数 `show_error` 在 try-catch 结构中捕捉到异常时调用，根据异常信息和传入的上下文提示用户出现错误的位置，便于用户调整输入。这些函数相对简单。

较为关键的 `execute` 函数会根据根据用户的选项调用 Life 类相应方法，具体实现为利用 switch-case 语句，根据用户提供的选项调用转跳到相应的调用点：

```
1.  void console_runtime::execute(Life &life, char option) {
```

```
    switch (option) {
        case 's': case 'S': life.show(); break;
        case 'n': case 'N': life.update(); break;
        case 'q': case 'Q':
            cout << "Game process finished." << endl;
            exit(0);
        case '?': cout << MENU << endl; break;
        default: throw std::runtime_error{"Invalid command"};
    }
}
```

对于两种不同的运行模式，在提示信息的输出上也有所不同。main 函数则解析命令行参数，通过输入重定向的方法来调整输入配置的位置。如下代码所示：

```
string file_name = "<stdin>";
ifstream fin;
streambuf *backup;
if (argc >= 2) {
    file_name = argv[argc - 1];
    fin.open(file_name);
    if (!fin.is_open()) {
        cerr << "Invalid file path: " << file_name << endl;
        exit(-1);
    }
    // 从文件读取配置的提示信息
    backup = cin.rdbuf();
    cin.rdbuf(fin.rdbuf()); // 输入重定向到文件
} else {
    // 从标准输入读取配置的提示信息
}
try { life.initialize(cin); }
catch (std::exception &e) {
    console_runtime::show_error(e, "Setup ", file_name, true, -2);
}
if (argc >= 2) {
    // 从文件读取配置完成的提示信息
    cin.rdbuf(backup);  // 恢复输入重定向到标准输入
    fin.close();    // 记得关闭文件
}
```

完成配置的初始化后，main 函数在无限循环中调用 wait_user 和 execute 两个函数不断获取用户菜单选项并执行相应的任务。如下代码所示：

```
for (;;) {
    try { console_runtime::execute(life, console_runtime::wait_user()); }
```

```
3.      catch (std::exception &e) {
4.          console_runtime::show_error(e, "Executing command");
5.      }
6.  }
```

综上所述，程序运行效果如下：

```
D:\WorkSpace\Codes\School\DS-Class\LifeGame\Src\cmake-build-debug\LifeGame.exe ./in
Life Game (Oct. 10 2020) By Holger Zhang

Welcome to the game of Life.
This game uses a grid of size 128*128.
Each cell can either be occupied by an organism or not.
The occupied cells change from generation to generation according to how many neighboring
 cells are alive.

Reading configuration from file_name './in'
Got configuration:
      x
   x  x
   x
   x

Now let's begin!

    ----------------------------
    | -*-*-*- Life Game -*-*-*- |
    |========= M E N U =========|
    |[s] Show the configuration |
    |[n] Next status            |
    |[q] Quit the game          |
    |[?] Show this menu again   |
    ----------------------------
Game-[Input Command]$ n
Configuration is updated to next status.
Game-[Input Command]$ m
Error [class std::runtime_error] In '<stdin>'
  Executing command: Invalid command
Game-[Input Command]$ s
Now the configuration is:


  xxx


Game-[Input Command]$ n
Configuration is updated to next status.
Game-[Input Command]$ s
Now the configuration is:

   x
   x
   x

Game-[Input Command]$ q
Game process finished.
```