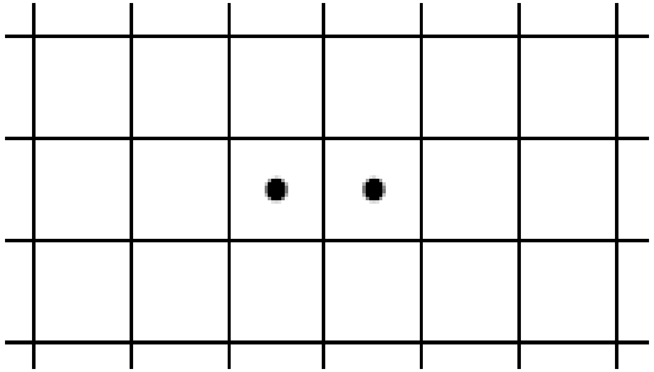
Game of life



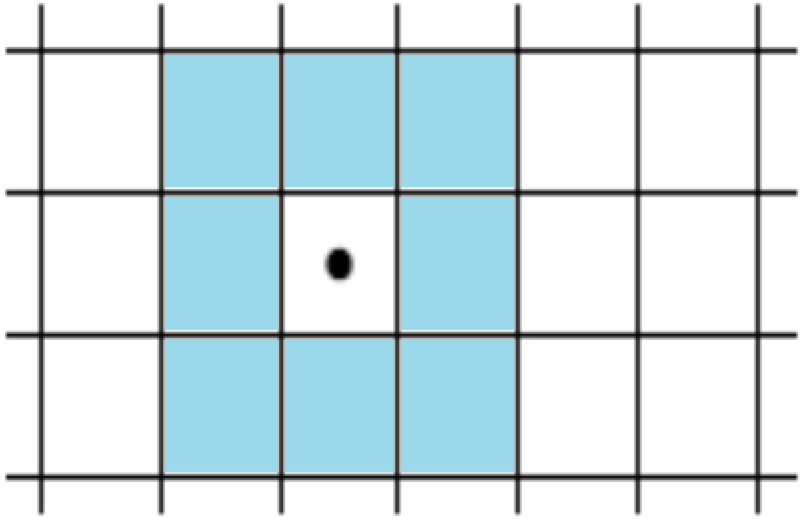
the game of life 游戏规则

- 该游戏在没有边界的矩形网格发生，矩形网格中的每个单元格可能被一个生物占据或没有占据。
- 被生物占据的单元称为活单元，未占据的单元称为死单元。
- 某个单元在下一代是活还是死，由该单元的活邻居数来决定。具体情况如下：

Rules of the Game of life

（1）给定单元格的邻居是与它垂直、 水平或对角相邻的八个单元。每一个细胞或者是活的或者是死的。（即只有两种状态）

（2）对于一个活单元：如果它有 2个或3个活的邻居，则它在下一代依然是活的；如果它有 0个、 1个、 4个 或多余4个活的邻居，则它在下一代就变成死单元。



Rules of the Game of life

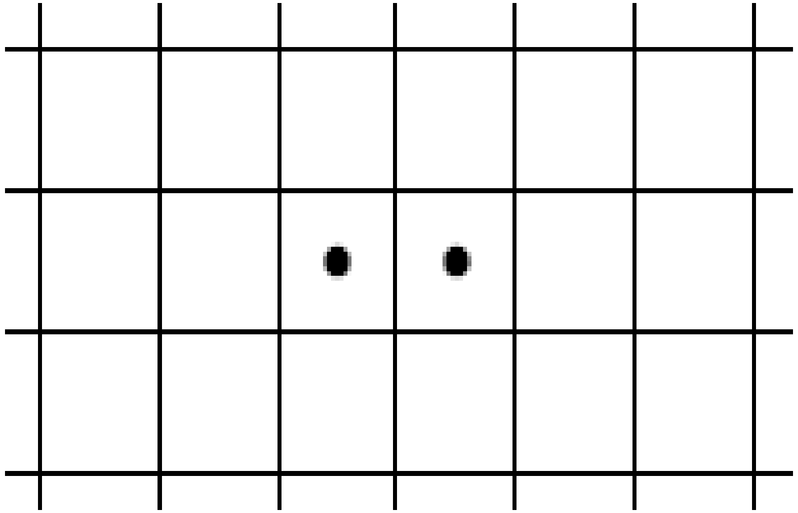
（3）对于一个死单元，如果它正好有3个相邻的活单元，没有更多或更少，那在下一代它会变成活单元，否则，这个单元仍然是死单元。

（4）所有的复活和死亡在同时发生。

configuration

在栅格中的一个特定的由活单元和死单元构成的布局称为一个配置（**Configuration**）。

应用上述规则，一个初始配置将会一代代更替。



4.2Examples

0	0	0	0	0	0
0	1	2	2	1	0
0	1	● 1	● 1	1	0
0	1	2	2	1	0
0	0	0	0	0	0

- 消亡的例子

由规则 2 和 3，两个活单元在下一代，都将死亡，而所有死单元都不会复活，所以，这个配置将会消亡。

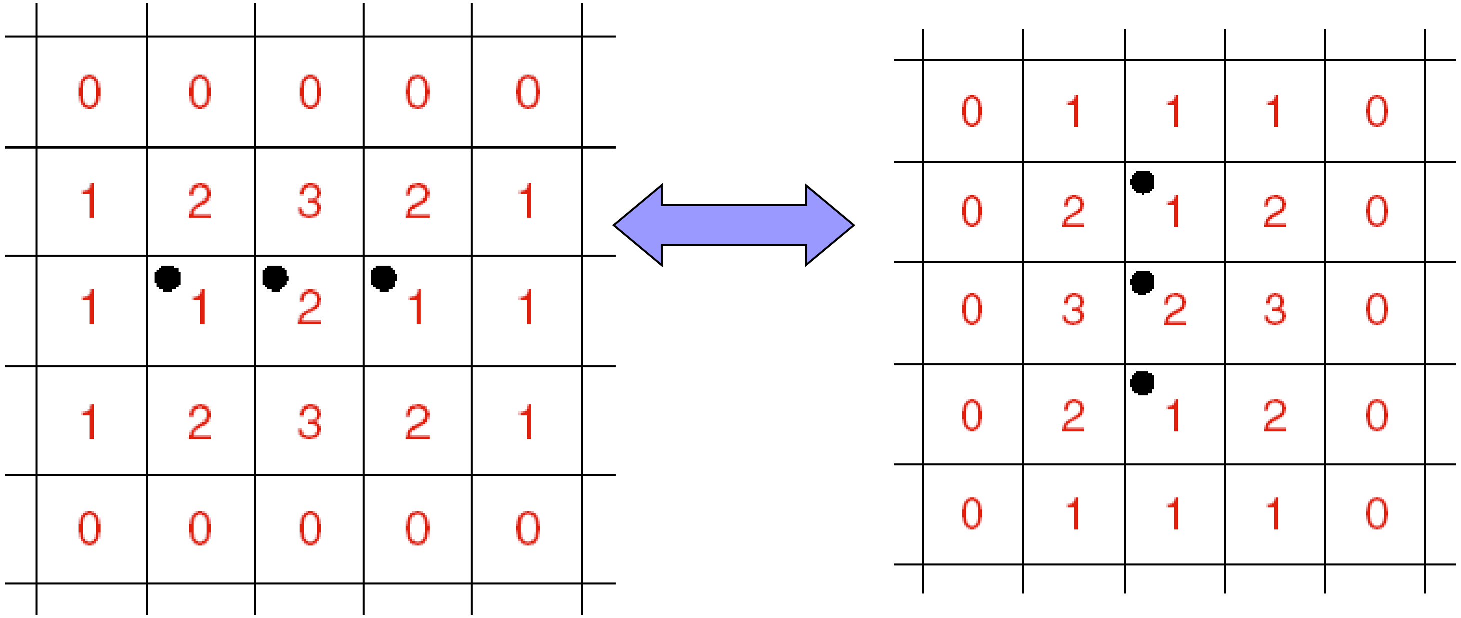
Examples --2

0	0	0	0	0	0
0	1	2	2	1	0
0	2	● 3	● 3	2	0
0	2	● 3	● 3	2	0
0	1	2	2	1	0
0	0	0	0	0	0

- 静止不变的例子
  - 每个活的单元都有**3**个活邻居，所以仍然活着；
  - 每个死的单元都有**2**个或少于**2**个的活邻居，所以仍然是死的；
  - 因此这个配置是静止不变的。

Examples --3

- □这是交替往复的两个例子



0	0	0	0	0
1	2	3	2	1
1	● 1	● 2	● 1	1
1	2	3	2	1
0	0	0	0	0

0	1	1	1	0
0	2	● 1	2	0
0	3	● 2	3	0
0	2	● 1	2	0
0	1	1	1	0

Variety(多样化)

一个简单的初始配置，通过代代更替，可以：

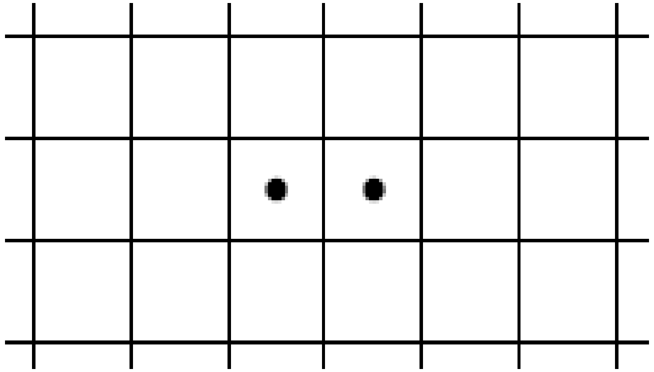
- 变成很大的配置
- 消亡
- 静止不变
- 每几代之间交替重复变化

Our goal（我们的目标）

写出一个程序，用于展示一个初始的配置是如何代代更替的。

4.3 the solution: classes ,objects and methods

- 创建一个life游戏初始配置
- 打印当前的配置
- 当用户希望看下一代配置时
  - 运用life game游戏规则，更新配置
  - 打印当前的配置



Review of C++ elements

- A *class* collects data and the methods used to access or change the data.
- 类是将数据、以及访问数据的方法集合在一起的一个整体。
- Such a collection of data and methods is called an *object* belonging to the given class.
- Every C++ class consists of *members* that represent either variables (called *data members*) or functions (called *methods* or *member functions*). The member functions of a class are normally used to access or alter the data members.

Review of C++ elements

- By relying on its *specifications*, a client can use a method without needing to know how the data are actually stored or how the methods are actually programmed. This important programming strategy known as *information hiding*.
- Data members and methods available to a client are called *public*; further *private* variables and functions may be used in the implementation of the class, but are not available to a client.

Convention（约定）

Methods of a class are public.

Functions in a class are private.

C++ solution for the Life game,

- Class: Life class
- Object: a configuration
- Method: initialize(), update() ,print()

4.4 Life: The Main Program

```
#include “utility.h”   //page 678-679
#include “life.h “//life类的定义部分
int main( ) // Program to play Conway’s game of Life.
/*Pre: The user supplies an initial configuration of
living cells.
  Post: The program prints a sequence of pictures
showing the changes in the configuration of living
cells according to the rules for the game of Life.
  Uses: The class Life and its methods
initialize（）,print（）， and update（）.The functions
instructions（）,user_ says_ yes（）. */
```

```
{
Life configuration;
instructions( );
configuration.initialize( );
configuration.print( );
cout << ″Continue viewing new generations? ″ << endl;
while (user_says_yes( )) {
     configuration.update( );
     configuration.print( );
     cout << ″Continue viewing new generations? ″ <<
endl;
}
}
```

Utility Package(p678-679)

- We shall assemble a *utility package* that contains various declarations and functions that prove useful in a variety of programs, even though these are not related to each other as we might expect if we put them in a class.
- For example, we shall put declarations to help with error processing in the utility package.
- Our first function for the utility package obtains a yes-no response from the user:

bool user_says_ yes( )
Pre: None.
Post: Returns **true** if the user enters ‘y’ or ‘Y’; returns **false** if the user enters ‘n’ or ‘N’; otherwise requests new response

Other C++ elements

- Clients, that is, user programs with access to a particular class, can declare and manipulate objects of that class.

For example:declare a Life object by: Life configuration;

- member selection operator We can now apply methods to work with configuration, using the C++ operator. For example, we can print out the data in configuration by writing:

```
configuration.print( );
```

- The *specifications* of a method, function, or program are statements of precisely what is done. *Preconditions* state what is required before; *postconditions* state what has happened when the method, function, or program has finished.

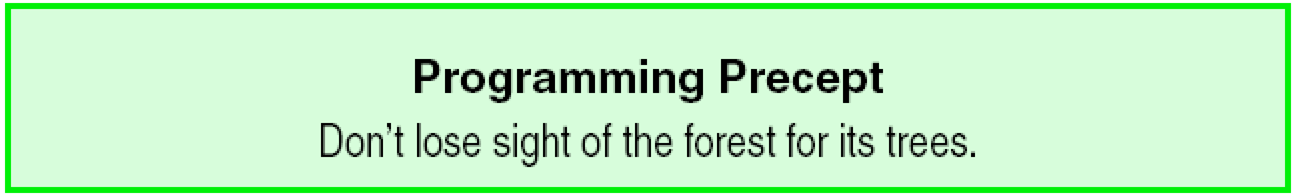
Programming Precept

Include precise specifications
with every program, function, and method that you write.

Programming Style(自学page 10-20)

- ❖Names（命名）: Guidelines of names: see p11-12
- ❖Documentation and Format （文档化和书写格式）
  - Some commonly accepted guidelines: see p13

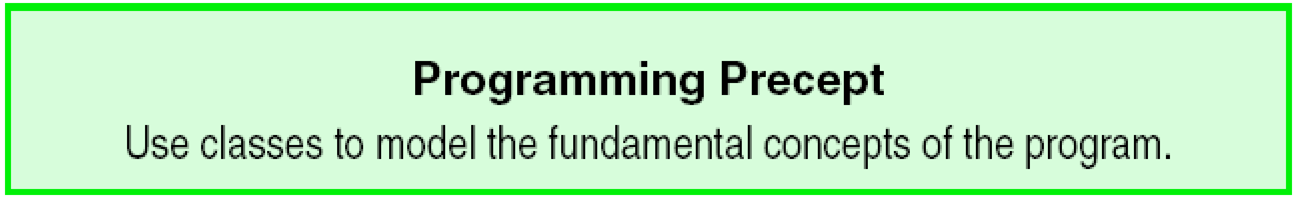
- ❖Refinement and Modularity （求精和模块化）:
  - Top-down refinement



Programming Precept

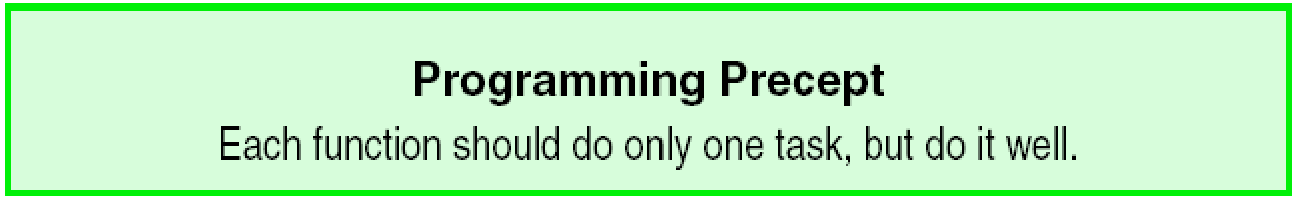
Don't lose sight of the forest for its trees.

- As we write the main program ,we decide exactly how the work will be divided among them.,how to divide the work?



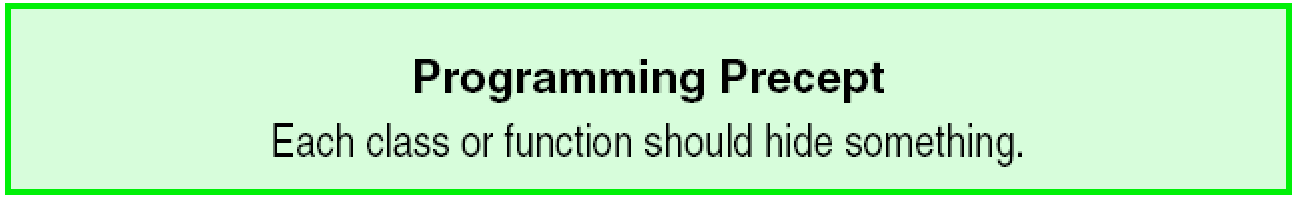
Programming Precept

Use classes to model the fundamental concepts of the program.



Programming Precept

Each function should do only one task, but do it well.

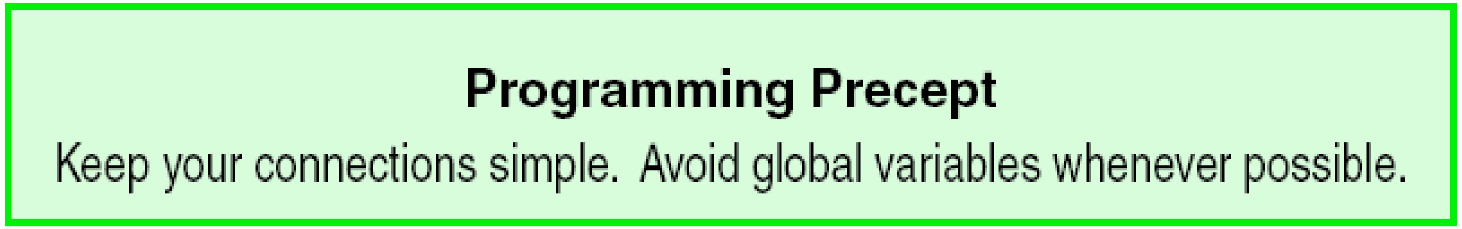


Programming Precept

Each class or function should hide something.

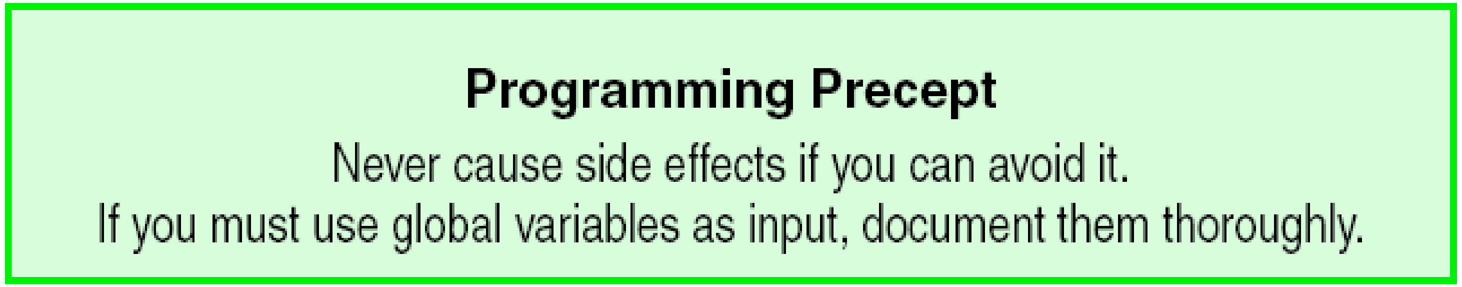
自学

- Data Categories
  - *Input* parameters--passed by value or by reference(const)
  - *Output* parameters—passed by reference (suggest)
  - *Inout* parameters---used for both input and output
  - *Local* variables—defined in the function and exist only while the function is being executed.
  - *Global* variables----dangerous, cause side effect



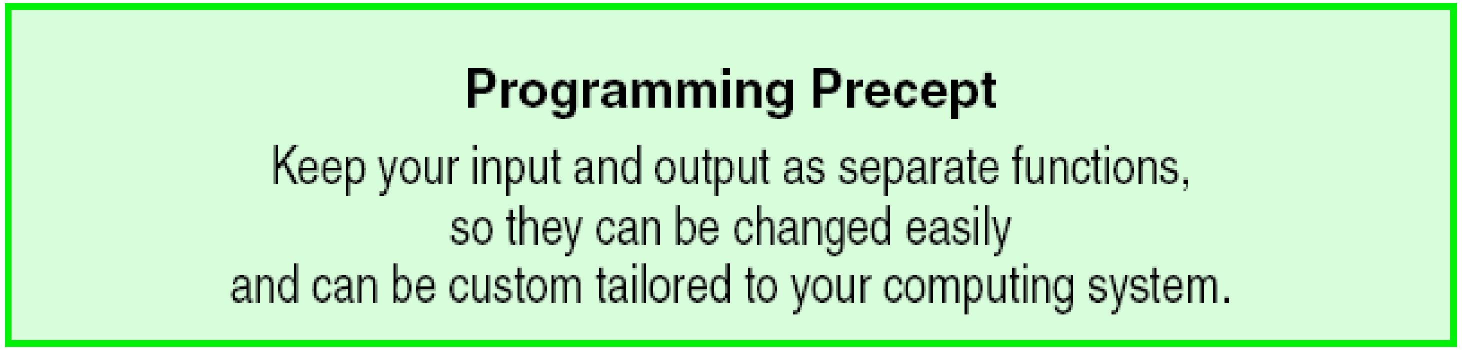
Programming Precept

Keep your connections simple. Avoid global variables whenever possible.



Programming Precept

Never cause side effects if you can avoid it.
If you must use global variables as input, document them thoroughly.



Programming Precept

Keep your input and output as separate functions,
so they can be changed easily
and can be custom tailored to your computing system.

1.4 Coding, Testing and Further Refinement

- Coding（编码）:writing an algorithm in the correct syntax of a computer language like C++
- Testing（测试）:running the program on sample data chosen to find errors
- Further Refinement（进一步细化）:turn to the functions not yet written and repeat these steps

Example demonstration
——game of life

- Coding

  1.Stubs（占位程序）

  - To compile each function and class separately
  - To compile the main program correctly, there must be something in the place of each function that is used, so put in short, dummy functions.——which named stubs
  - the simplest stubs are those that do little or nothing at all:

```
void instruction() {}
bool user_says_yes()        { return false;}
```

Coding of game of life

2.Definition of the class Life

```
const int maxrow=20,maxcol=60;  //living cells共有20行，60列
class Life{          //存放在*.h文件中的类体声明
public:
    void initialize();  //初始化living cells的状态
    void print();   //打印输出当前living cells的状态
    void update();  //进行dead⇔living间的转换
private:
    int grid[maxrow+2][maxcol+2];  //借助两维数组存放living cells
    //注意：为处理一致，添加了虚拟的2行2列，放围墙
    int neighbor_count(int row,int col);  //统计邻居cell的状态
};
```

Stub example2:

To check if the class definition is correct, We should supply the following stubs for its methods in life.cpp:

```
void Life :: initialize( ) {}
void Life :: print( ) {}
void Life :: update( ) {}
```

Instructions

```
void instructions( )
/* Pre: None.
Post: Instructions for using the Life program have been printed. */
{
cout << "Welcome to Conway’s game of Life." << endl;
cout << "This game uses a grid of size "
      << maxrow << " by " << maxcol << " in which each" << endl;
cout << "cell can either be occupied by an organism or not." << endl;
cout << "The occupied cells change from generation to generation"<< endl;
cout << "according to how many neighboring cells are alive."<< endl;
}
```

user_says_yes( )

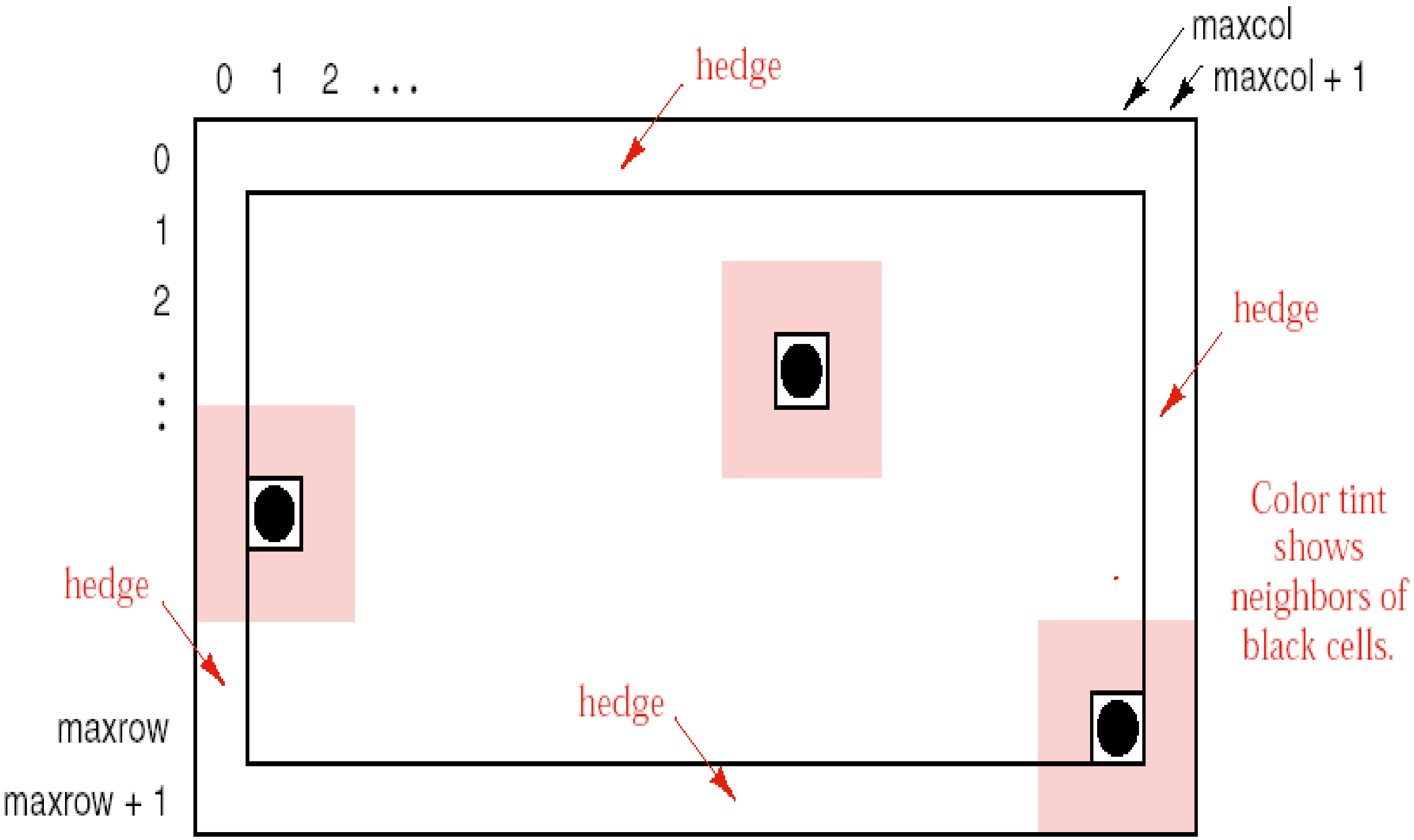
```
bool user_says_yes( )
{
    int c;
    bool initial_response = true;
    do { // Loop until an appropriate input is received.
    if (initial_response)
          cout << " (y,n)? " << flush;
    else
          cout << "Respond with either y or n: " << flush;
    do { // Ignore white space.
          c = cin.get( );
          } while (c == ‘\n’ || c == ‘ ’ || c == ‘\t’);
    initial_response = false;
} while (c != ‘y‘ && c != ‘Y‘ && c != ‘n‘ && c != ‘N’);
return (c == ‘y‘ || c == ‘Y‘);
}
```

Functions and Methods of class Life

3.Counting neighbors:

```
int Life::neighbor_count(int row,int col){
        int i,j,count=0;
            for (i=row-1;i<=row+1;i++)
                        for (j=col-1;j<=col+1;j++)
                            count+=grid[i][j];//如果存活，则累加；否则为0
            count-=grid[row][col];  //去除自己
            return count;
  }
```

sentinel（岗哨）**(hedge)**: A sentinel is an extra entry put into a data structures so that boundary conditions need not be treated as a special case.



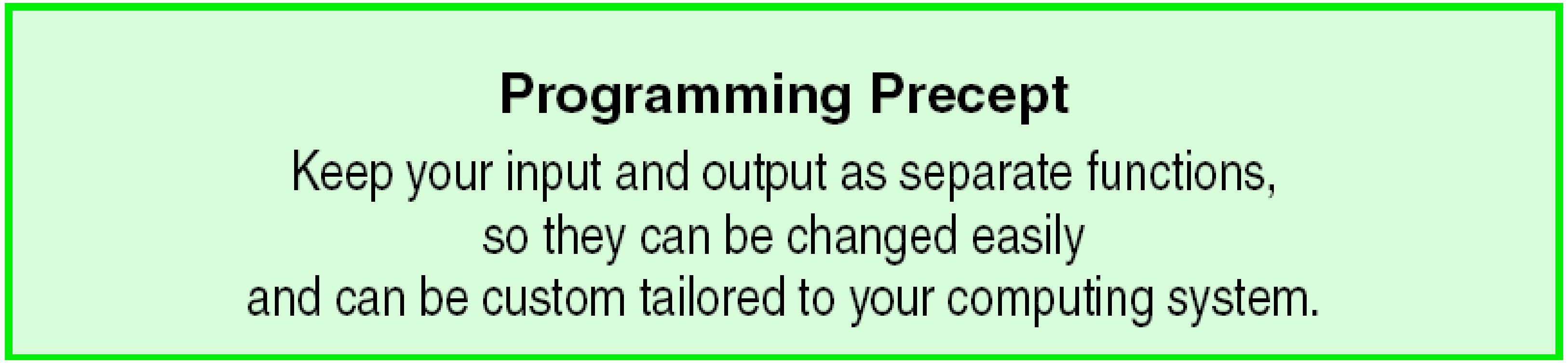
Functions and Methods of Class Life

4.Updating the grid

```
void Life::update(){
    int row,col,new_grid[maxrow+2][maxcol+2];
    for (row=1;row<=maxrow;row++)
      for (col=1;col<=maxcol;col++)
        switch(neighbor_count(row,col)){//调用统计函数,按结果分情况
            case 2: new_grid[row][col]=grid[row][col]; break;//不变
            case 3: new_grid[row][col]=1; break; //激活
            default: new_grid[row][col]=0; //dead
        }
    for (row=1;row<=maxrow;row++)
      for (col=1;col<=maxcol;col++)
        grid[row][col]=new_grid[row][col];//将临时数组中的数据拷贝回原
    grid数组
}
```

Functions and Methods of Class Life

5.Input and Output:



Programming Precept

Keep your input and output as separate functions,
so they can be changed easily
and can be custom tailored to your computing system.

Functions and Methods of Class Life

- Input——initialize

```
void Life::initialize(){
   int row,col;
   for (row=0;row<=maxrow+1;row++)
     for (col=0;col<=maxcol+1;col++)
         grid[row][col]=0;
   cout<<“List the coordinates for living cells.”<<endl;
   cout<<“Terminate the list with the special pair (-1,-1)”<<endl;
   cin>>row>>col;
   while (row!=-1||col!=-1){
     if (row>=1&&row<=maxrow)
         if (col>=1&&col<=maxcol) grid[row][col]=1;
         else cout<<“Column “<<col<<“ is out of range.”<<endl;
     else  cout<<“Row “<<row<<“ is out of range.”<<endl;
     cin>>row>>col;
   }
  }
```

Functions and Methods of class Life

- Output——print:

```
void Life::print(){
  int row,col;
  cout<<“\nThe current Life configuration is: “<<endl;
  for (row=1;row<=maxrow;row++){
        for (col=1;col<=maxcol;col++)
          if (grid[row][col]==1) cout<<“ * ”;
               else cout<<“  ”;
          cout<<endl;
        }
        cout<<endl;
  }
}
```

Functions and Methods of Class Life

6.Drivers:

Driver is a short auxiliary program whose purpose is to provide the necessary input for the function, and evaluate the result——so that each function can be isolated and studied by itself.

Example 1:driver for neighbor_ count():

```
int main ( ) // driver for neighbor_count( )
/* Pre: None.
Post: Verifies that the method neighbor_count ( ) if it
returns the correct values.
Uses: The class Life and its method initialize( ) . */
{
    Life configuration;
    configuration.initialize( );
    for (row = 1; row <= maxrow; row++) {
             for (col = 1; col <= maxrow; col++)
           cout << configuration.neighbor_count(row,
    col) << ” ”;
             cout << endl;
       }
}
```

Example 2 :driver for initialize() and print():

Sometimes two functions can be used to check each other.

```
        configuration.initialize( );
        configuration.print( );
```

Both methods can be tested by running this driver and making sure that the configuration printed is the same as that given as input.

7. **Methods for debugging: (self-study)**

1). Structured walkthrough

2). Trace tools and snapshots

3). Scaffolding

4). Static analyzer

Example demonstration ——game of life

8.Test(self-study)

- Principles of program Testing:
  - Black-Box Method（黑盒法）

(a) Easy values

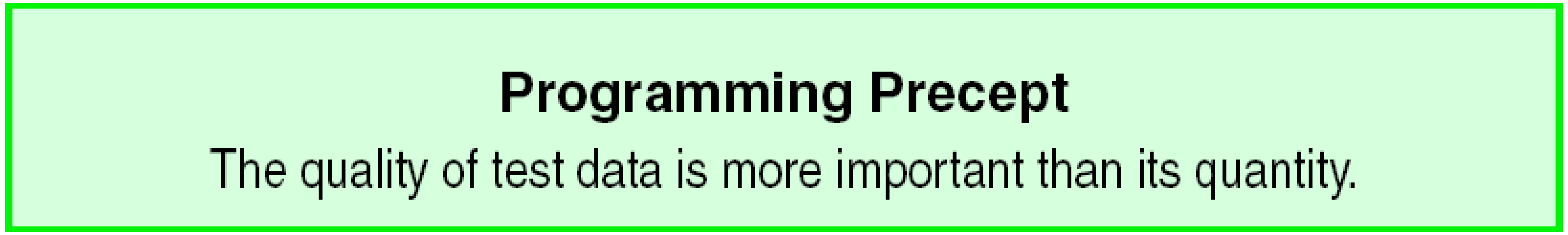
(b) Typical, realistic values

(c) Extreme values

(d) Illegal values

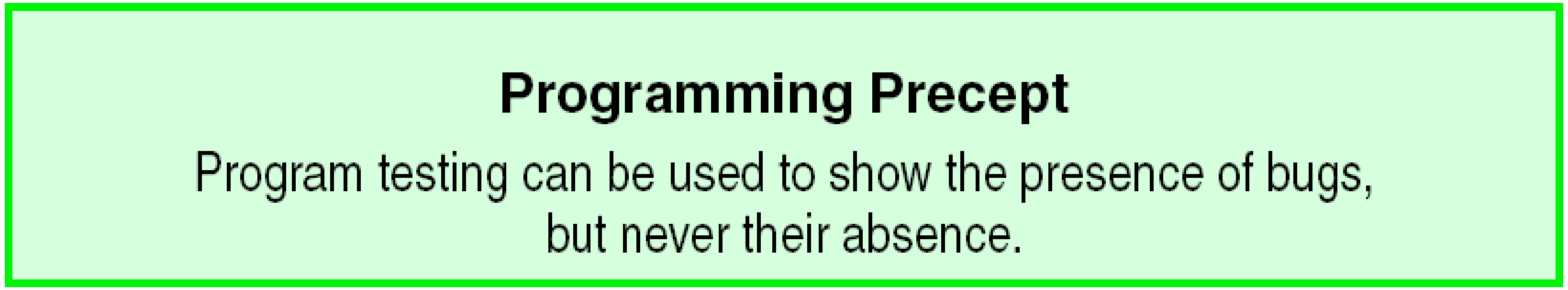
- Glass-Box Method（白盒法）：Trace all the paths through the Program,for a single small method, it is an excellent debugging and test method
- Ticking-Box Method——Don't do anything Just hope for the best!

- Action:
  - In practice, black-box testing is usually more effective in uncovering errors. One reason is that most subtle programming errors often occur not within a function but in the interface between functions, in misunderstanding of the exact conditions and standards of information interchange between function.



Programming Precept

The quality of test data is more important than its quantity.



Programming Precept

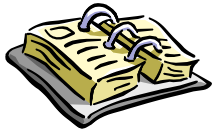
Program testing can be used to show the presence of bugs,
but never their absence.

Example demonstration ——game of life

- Program Maintenance (self-study)
  - For a large and import program, more than half the work comes in the maintenance phase, after it has been completely debugged, tested, and put into use.
  - First step of program maintenance is to begin the continuing process of review, analysis, and evaluation. (for some useful question, see p34)

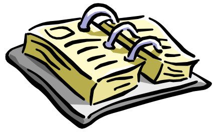
Example demonstration ——game of life

- Program Maintenance
  - Review of the Life program
    - problem specification（问题规范）
    - program correctness（程序纠错）
    - user interface（用户接口）
    - modularity and structure（模块化与结构化）
    - documentation（文档）
    - efficiency（效率）
  - Program revision and redevelopment（修订与发布）



Pointers and Pitfalls

- 1. To improve your program, review the logic. Don't optimize code based on a poor algorithm.
- 2. Never optimize a program until it is correct and working.
- 3. Don't optimize code unless it is absolutely necessary.
- 4. Keep your functions short; rarely should any function be more than a page long.
- 5.Be sure your algorithm is correct before starting to code.
- 6. Verify the intricate parts of your algorithm.



Algorithm and program

- Description of the particular steps of the process of a problem(处理某一问题的步骤的描述)
  - characteristics
    - Finity----有穷性，有限时间内执行完毕，不能有死循环
    - Certainty----确定性，对于一个确定的输入，只有一个固定的输出。不能使用随机函数等。
    - Feasibility---可行性，每个步骤都是明确的，理解起来没有二义性。
    - Input interface---0个或多个输入
    - Output interface---1个或多个输出
    - universality---通用性
  - Format is flexible，it can be described by Natural Language, Advanced Language ,and the syntax is not the most important thing,
  - Nothing to do with the environment

- ❐ program:The whole process of the problem（用计算机处理问题的全部代码的整体）
  - □ May be Infinite
  - □ Syntax must be accurate
  - □ Closely linked with the environment