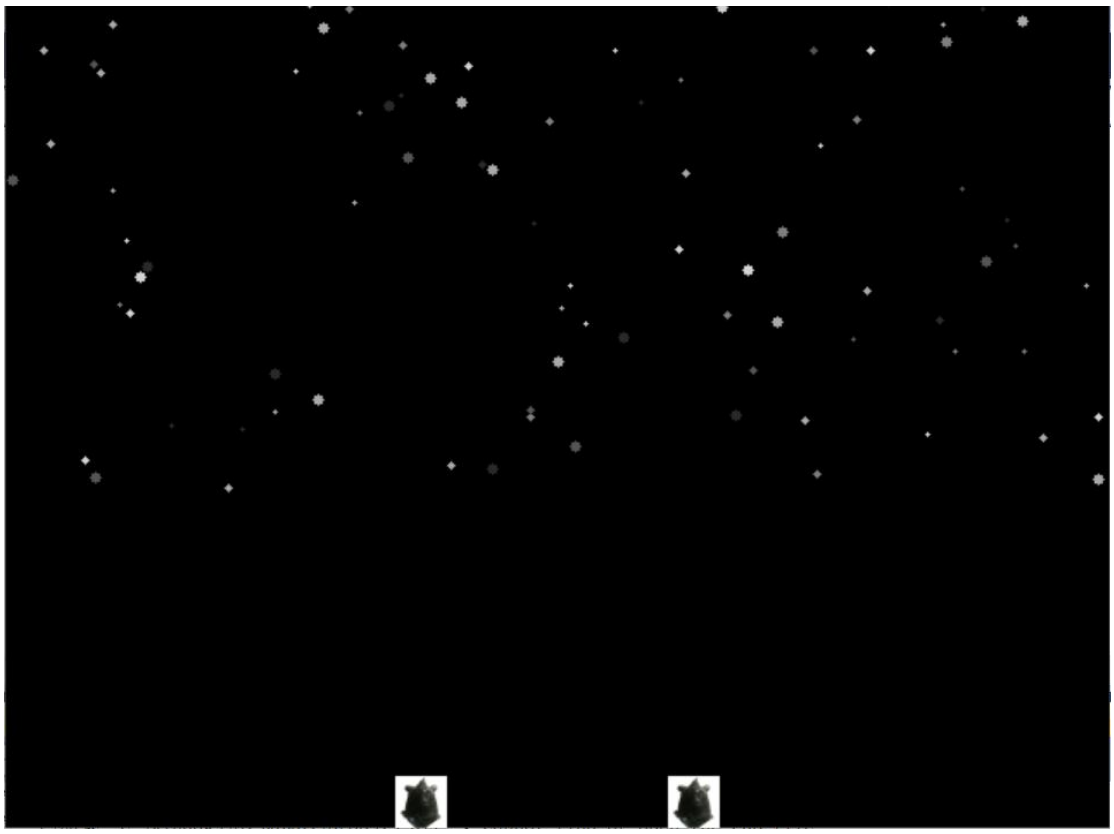
C/C++从入门到精通-高级程序员之路

【 数据结构 】

顺序表及其企业级应用

第 1 节 顺序表的故事导入

“我要看小星星！”，小芳轻轻的对程序员男友 Jack 说。
“亲爱的，现在是冬天，哪里有小星星耶？”
“我不，我就要嘛！”
看着可爱带着小小野蛮的女友，Jack 沉思了一会，说了一句，“好吧！”
默默的回到了电脑前... ... 一个小时后，做出来如下的效果:



当小芳看到满天星星时，开心得不得了！但是当她看到下面时，刚刚还是晴空万里的心情，马上就乌云密布了，结果是：

1. Jack 把接下来一个月的家务都包下来，包括洗衣做饭
2. 小芳要求看到星星慢慢的从屏幕上方消失，如果实现不了，搓衣板和榴莲，自己看着办！

……

难道这就是我的野蛮女友嘛？从认识到现在一直都是这么野蛮。。。！ 哎，不想这么多了，现在最重要的是考虑该怎么实现？

以下是 Jack 目前实现的代码：

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <string>

using namespace std;

#define MAX_STAR        100
#define SCREEN_WIDTH    640
#define SCREEN_HEIGHT   480
#define MAX_STEP        5
#define MAX_RADIUS      3
#define BOTTOM_MARGIN   100

//星星状态
enum STATUS{
    STOP=0,
    UP,
    DOWN,
    LEFT,
    RIGHT,
    RANDOM,
    ALL_STATUS
};


struct STAR{
    int x;              //星星的 x 坐标
    int y;              //星星的 y 坐标
    enum STATUS stat;   //状态
    unsigned radius;    //星星的半径
    int step;           //每次跳跃的间隔
    int color;          //星星的颜色
};

struct STAR star[MAX_STAR];


/**************************************
 * 功能：初始化星星
```

```
 *  输入参数:
 *       i - 星星在全局数组中的下标
 *  返回值: 无
 ************************************/
void initStar(int i){
    int rgb = 0;

    if(i<0 || i>MAX_STAR) {
        fprintf(stderr,"老司机,你传的值 i[%d]我受不了!",i);    //log 日
志
        return;
    }

    //rand()  得到随机数范围 0 - 32767  RAND_MAX
    star[i].x = rand()% SCREEN_WIDTH;                          // x 范围 0-639
    star[i].y = rand()% (SCREEN_HEIGHT - BOTTOM_MARGIN);// y 范围 0 - 379
    //star[i].stat = (enum STATUS)(rand()%ALL_STATUS);
    star[i].radius = 1+rand()%MAX_RADIUS;  //半径控制  1 - 3
    star[i].step = rand() % MAX_STEP +1;   //步长  1 - 5
    rgb = 255 * star[i].step / MAX_STEP;  // 0 - 255
    star[i].color = RGB(rgb, rgb, rgb);
}

int main(){
    initgraph(SCREEN_WIDTH, SCREEN_HEIGHT);

    for(int i=0; i<MAX_STAR; i++){
        initStar(i);
    }

    for(int i=0; i<MAX_STAR; i++){
        setfillcolor(star[i].color);
         solidcircle(star[i].x, star[i].y, star[i].radius);
    }

    IMAGE tortoise;//王八图片
    loadimage(&tortoise, "tortoise.jpg", 30, 30, false);
    putimage(SCREEN_WIDTH*4/10-30, SCREEN_HEIGHT-30, &tortoise);
    putimage(SCREEN_WIDTH*6/10, SCREEN_HEIGHT-30, &tortoise);

    system("pause");
    closegraph();
    return 0;
}
```

看着家里的各种“刑具”，Jack 绞尽脑汁，思索着怎么让星星慢慢从屏幕上方消失！
请大家帮帮这位 Jack ,庆幸我们之前就写过推箱子，算是老司机了，我们来帮 Jack 一起实现这个功能！

```
#include <graphics.h>
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>
#include <string>

using namespace std;

#define MAX_STAR 100
#define SCREEN_WIDTH   640
#define SCREEN_HEIGHT  480
#define MAX_STEP       5
#define MAX_RADIUS     3
#define BOTTOM_MARGIN  100

//星星状态
enum STATUS{
    STOP=0,
    UP,
    DOWN,
    LEFT,
    RIGHT,
    RANDOM,
    ALL_STATUS
};


struct STAR{
    int x;              //星星的 x 坐标
    int y;              //星星的 y 坐标
    enum STATUS stat;   //状态
    unsigned radius;    //星星的半径
    int step;           //每次跳跃的间隔
    int color;          //星星的颜色
};

struct STAR star[MAX_STAR];


bool isQuit(){
    for(int i=0; i<MAX_STAR; i++){
        if(star[i].x>0 && star[i].x<SCREEN_WIDTH  && star[i].y>0 &&
```

```
star[i].y<SCREEN_HEIGHT){
            return  false;
        }
    }
    return true;
}


void MoveStar(int i){

    if(star[i].stat == STOP) return ;

    //擦除原来的星星
    setfillcolor(BLACK);
    solidcircle(star[i].x, star[i].y, star[i].radius);

    if(star[i].stat == DOWN){
        star[i].y =star[i].y + star[i].step;
        //if(star[i].y>SCREEN_HEIGHT) star[i].y = 0;
    }else if(star[i].stat == UP){
        star[i].y -= star[i].step;
        //if(star[i].y<0) star[i].y = SCREEN_HEIGHT;
    }else if(star[i].stat == LEFT){
        star[i].x -= star[i].step;
        //if(star[i].x<0) star[i].x = SCREEN_WIDTH;
    }else if(star[i].stat == RIGHT){
        star[i].x += star[i].step;
        //if(star[i].x>SCREEN_WIDTH) star[i].x = 0;
    }

    setfillcolor(star[i].color);
    solidcircle(star[i].x, star[i].y, star[i].radius);

}

/*************************************
 * 功能：初始化星星
 *  输入参数：
 *      i - 星星在全局数组中的下标
 *  返回值：无
 *************************************/
void initStar(int i){
    int rgb = 0;

    if(i<0 || i>MAX_STAR) {
        fprintf(stderr,"老司机，你传的值 i[%d]我受不了！",i);    //log 日
志
```

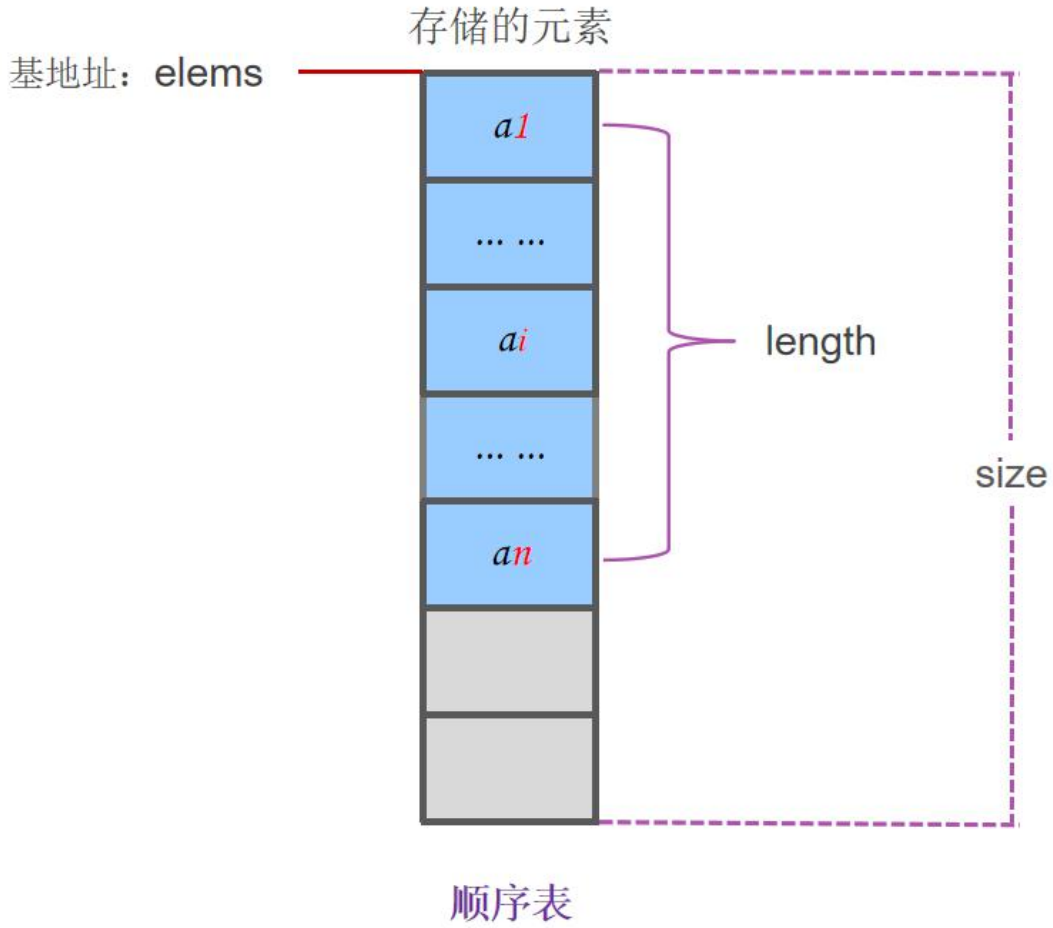
```
        return;
    }

    //rand()  得到随机数范围 0 - 32767  RAND_MAX
    star[i].x = rand()% SCREEN_WIDTH;                    // x 范围 0-639
    star[i].y = rand()% (SCREEN_HEIGHT - BOTTOM_MARGIN);// y 范围 0 - 379
    star[i].stat = UP;
    star[i].radius = 1+rand()%MAX_RADIUS;  //半径控制  1 - 3
    star[i].step = rand() % MAX_STEP +1;   //步长  1 - 5
    rgb = 255 * star[i].step / MAX_STEP;  // 0 - 255
    star[i].color = RGB(rgb, rgb, rgb);
}

int main(){
    bool quit = false;

    initgraph(SCREEN_WIDTH, SCREEN_HEIGHT);

    for(int i=0; i<MAX_STAR; i++){
        initStar(i);
    }

    for(int i=0; i<MAX_STAR; i++){
        setfillcolor(star[i].color);
        solidcircle(star[i].x, star[i].y, star[i].radius);
    }

    IMAGE tortoise;//王八图片
    loadimage(&tortoise, "tortoise.jpg", 30, 30, false);
    putimage(SCREEN_WIDTH*4/10-30, SCREEN_HEIGHT-30, &tortoise);
    putimage(SCREEN_WIDTH*6/10, SCREEN_HEIGHT-30, &tortoise);

    while(quit==false){
        for(int i=0; i<MAX_STAR; i++){
            MoveStar(i);
        }
        if(isQuit()){
            quit = true;
        }

        Sleep(50);
    }

    system("pause");
    closegraph();
    return 0;
}
```

第 2 节 顺序表的原理精讲

顺序表是简单的一种线性结构，逻辑上相邻的数据在计算机内的存储位置也是相邻的，可以快速定位第几个元素，中间不允许有空值，插入、删除时需要移动大量元素。

顺序表的三个要素：

- 用 elems 记录存储位置的基地址
- 分配一段连续的存储空间 size
- 用 length 记录实际的元素个数，即顺序表的长度



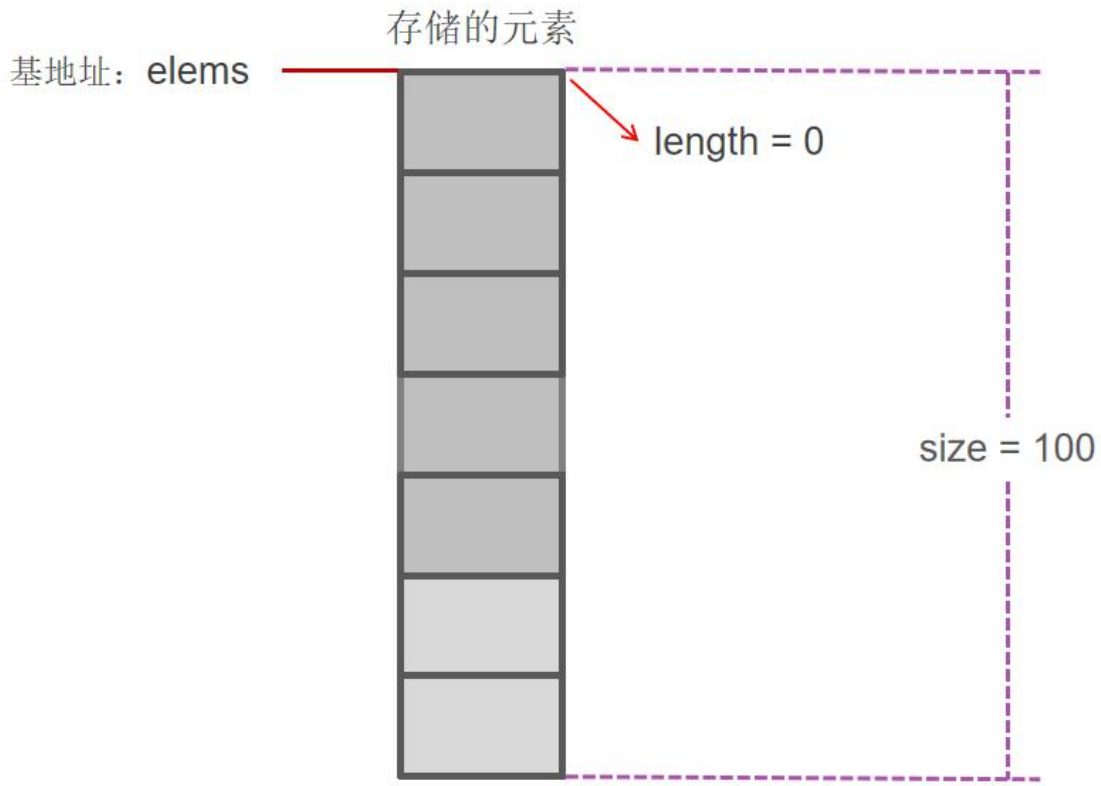
结构体定义

```
#define    MAX_SIZE        100
struct _SqList{
    ElemType    *elems;          // 顺序表的基地址
    int             length;          // 顺序表的长度
    int             size;            // 顺序表总的空间大小
}
```

第 3 节 顺序表的算法实现

顺序表的初始化

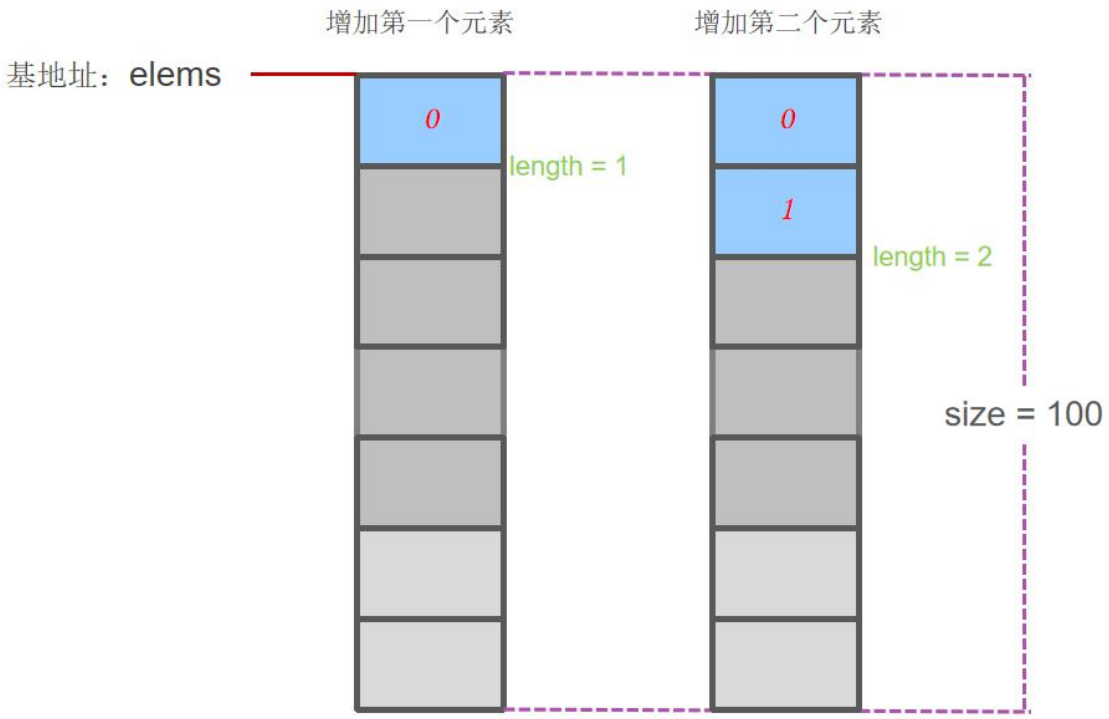
```
#define MAX_SIZE  100

typedef struct{
    int *elems; // 顺序表的基地址
    int length; // 顺序表的长度
    int size;   // 顺序表的空间
}SqList;


bool initList(SqList &L) //构造一个空的顺序表 L
{
    L.elems=new int[MAX_SIZE];    //为顺序表分配 Maxsize 个空间
    if(!L.elems) return false;  //存储分配失败
    L.length=0;                 //空表长度为 0
    L.size = MAX_SIZE;
    return true;
}
```



顺序表初始化

顺序表增加元素

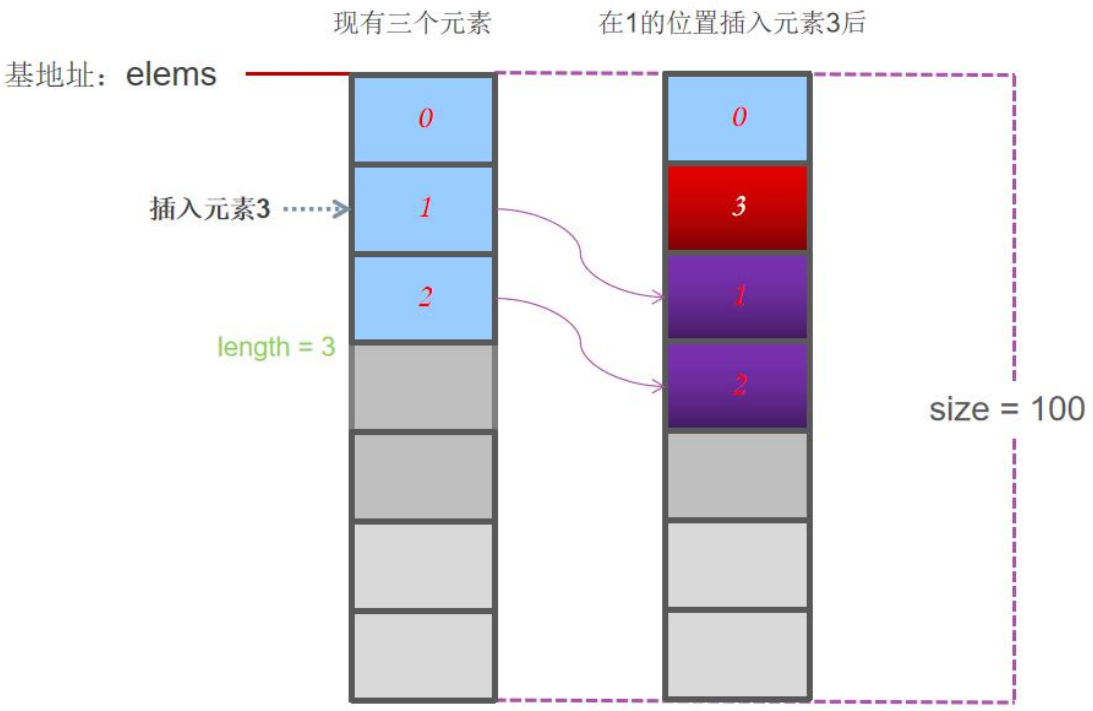
```
bool listAppend(SqList &L, int e)
{
    if(L.length==MAX_SIZE) return false; //存储空间已满
    L.elems[L.length] = e;
    L.length++;                 //表长增 1
    return true;
}
```



顺序表增加元素

顺序表插入元素

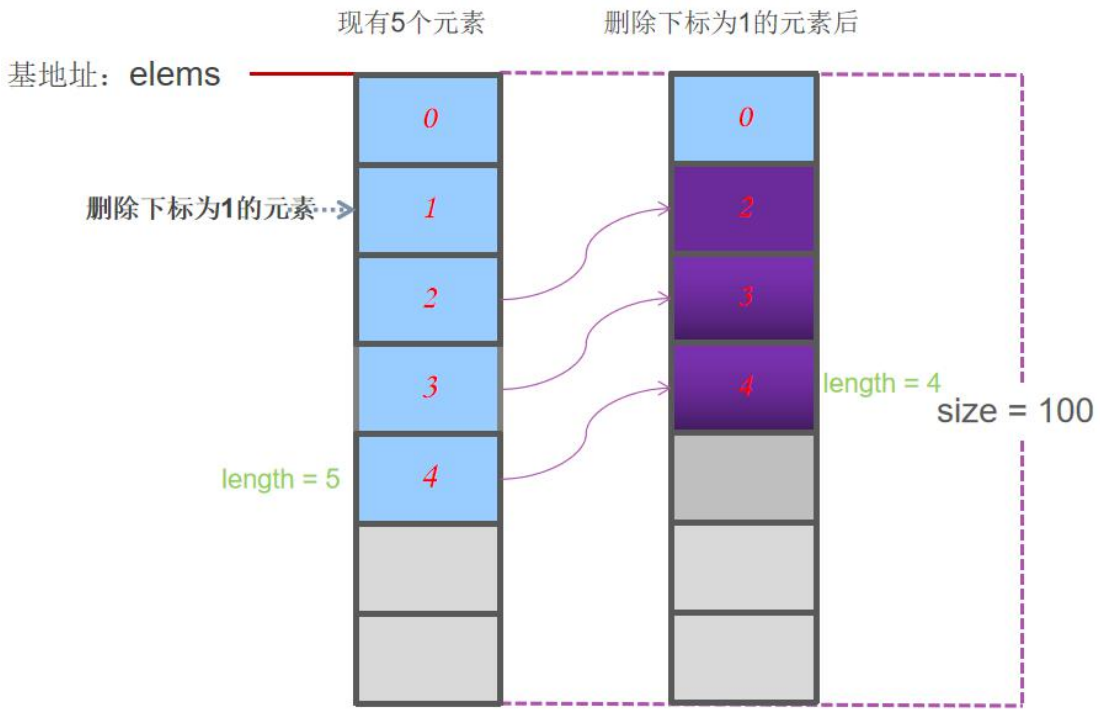
```
bool listInsert(SqList &L,int i, int e)
{
    if(i<0 || i>=L.length)return false; //i 值不合法
    if(L.length==MAX_SIZE) return false; //存储空间已满

    for(int j=L.length-1; j>=i; j--){
        L.elems[j+1]=L.elems[j]; //从最后一个元素开始后移，直到第 i 个元素后移
    }

    L.elems[i]=e;                //将新元素 e 放入第 i 个位置
    L.length++;                  //表长增 1
    return true;
}
```



顺序表插入元素

顺序表删除元素

```
bool listDelete(SqList &L,int i)
{
    if(i<0 || i>=L.length) return false;   //不合法

    if(i == L.length-1){//删除最后一个元素，直接删除
        L.length--;
        return true;
    }

    for (int j=i; j<L.length-1; j++){
        L.elems[j] =L.elems[j+1]; //被删除元素之后的元素前移
    }

    L.length--;
    return true;
}
```



顺序表删除元素

顺序表销毁

```
void destroyList(SqList &L)
{
    if (L.elems) delete []L.elems;//释放存储空间
    L.length = 0;
    L.size = 0;
}
```

完整代码实现:

```
#include <iostream>

using namespace std;

#define MAX_SIZE  100

typedef struct{
    int *elems; // 顺序表的基地址
    int length; // 顺序表的长度
    int size;   // 顺序表的空间
}SqList;

bool initList(SqList &L) //构造一个空的顺序表 L
{
    L.elems=new int[MAX_SIZE];    //为顺序表分配 Maxsize 个空间
    if(!L.elems) return false;  //存储分配失败
    L.length=0;                 //空表长度为 0
    L.size = MAX_SIZE;
    return true;
}

/*
bool getElem(SqList &L,int i,int &e)
{
    //防御性检查
    if (i<1||i>L.length) return false;
    e=L.elems[i-1];   //第 i-1 的单元存储着第 i 个数据
    return true;
}
*/

bool listAppend(SqList &L, int e)
{
```

```
    if(L.length==MAX_SIZE) return false; //存储空间已满
    L.elems[L.length] = e;
    L.length++;              //表长增 1
    return true;
}

bool listInsert(SqList &L,int i, int e)
{
    if(i<0 || i>=L.length)return false; //i 值不合法
    if(L.length==MAX_SIZE) return false; //存储空间已满

    for(int j=L.length-1; j>=i; j--){
        L.elems[j+1]=L.elems[j];//从最后一个元素开始后移，直到第 i 个元
素后移
    }

    L.elems[i]=e;             //将新元素 e 放入第 i 个位置
    L.length++;                  //表长增 1
    return true;
}



bool listDelete(SqList &L,int i)
{
    if(i<0 || i>=L.length) return false;  //不合法

    if(i == L.length-1){//删除最后一个元素，直接删除
        L.length--;
        return true;
    }

    for (int j=i; j<L.length-1; j++){
        L.elems[j] =L.elems[j+1]; //被删除元素之后的元素前移
    }

    L.length--;
    return true;
}

void listPrint(SqList &L)
{
    cout << "顺序表元素 size: "<<L.size<<", 已保存元素个数 length: "<<
L.length <<endl;
    for(int j=0; j<=L.length-1; j++){
        cout<<L.elems[j]<<" ";
    }
    cout<<endl;
```

```
}

void destroyList(SqList &L)
{
    if (L.elems) delete []L.elems;//释放存储空间
    L.length = 0;
    L.size = 0;
}


int main()
{
    SqList list;
    int i,e;

    cout << "顺序表初始化..." <<endl;

    //1. 初始化
    if(initList(list)){
        cout <<"顺序表初始化成功！" << endl;
    }

    //2. 添加元素
    int count = 0;
    cout<<"请输入要添加的元素个数：";
    cin>>count;

    for(int i=0; i<count; i++){
        cout << "\n 请输入要添加元素 e:";
        cin>>e;
        if(listAppend(list, e)){
            cout <<"添加成功！" << endl;
        }else{
            cout <<"添加失败！" << endl;
        }
    }
    listPrint(list);

    //3. 插入元素
    cout << "请输入要插入的位置和要插入的数据元素 e:";
    cin>>i>>e;
    if(listInsert(list, i, e)){
        cout <<"插入成功！" << endl;
    }else{
        cout <<"插入失败！" << endl;
    }
    listPrint(list);
```

```
    //4. 删除元素
    cout << "请输入要删除的位置 i:";
    cin>>i;
    if(listDelete(list, i)){
        cout <<" 删除成功！" << endl;
    }else{
        cout <<"删除失败！" << endl;
    }

    listPrint(list);

    //5. 销毁
    cout << "顺序表销毁..."<<endl;
    destroyList(list);

    system("pause");
    return 0;
}
```

浪漫星空代码优化

```
//star.cpp

#include <graphics.h>
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>
#include <string>
#include "star.h"

using namespace std;


void MoveStar(SqList &L, int i){

    if(L.elems[i].stat == STOP) return ;

    //擦除原来的星星
    setfillcolor(BLACK);
    solidcircle(L.elems[i].x, L.elems[i].y, L.elems[i].radius);

    if(L.elems[i].stat == DOWN){
        L.elems[i].y =L.elems[i].y + L.elems[i].step;
```

```
        if(L.elems[i].y>SCREEN_HEIGHT) listDelete(L, i);
    }else if(L.elems[i].stat == UP){
        L.elems[i].y -= L.elems[i].step;
        if(L.elems[i].y<0) listDelete(L, i);
    }else if(L.elems[i].stat == LEFT){
        L.elems[i].x -= L.elems[i].step;
        if(L.elems[i].x<0) listDelete(L, i);
    }else if(L.elems[i].stat == RIGHT){
        L.elems[i].x += L.elems[i].step;
        if(L.elems[i].x>SCREEN_WIDTH) listDelete(L, i);
    }

    setfillcolor(L.elems[i].color);
    solidcircle(L.elems[i].x, L.elems[i].y, L.elems[i].radius);

}

/************************************
 * 功能：初始化星星
 *  输入参数：
 *      i - 星星在全局数组中的下标
 *  返回值：无
 ************************************/
void initStar(struct STAR &_star){
    int rgb = 0;


    //rand()  得到随机数范围 0 - 32767  RAND_MAX
    _star.x = rand()% SCREEN_WIDTH;                    // x 范围 0 -639
    _star.y = rand()% (SCREEN_HEIGHT - BOTTOM_MARGIN);// y 范围 0 - 379
    _star.stat = UP;
    _star.radius = 1+rand()%MAX_RADIUS;  //半径控制  1 - 3
    _star.step = rand() % MAX_STEP +1;   //步长  1 - 5
    rgb = 255 * _star.step / MAX_STEP;  // 0 - 255
    _star.color = RGB(rgb, rgb, rgb);
}

int main(){
    bool quit = false;
    struct STAR  star;

    SqList  starList;

    //初始化保存星星状态的顺序表
    initList(starList);

    initgraph(SCREEN_WIDTH, SCREEN_HEIGHT);
```

```
    for(int i=0; i<MAX_STAR; i++){
        initStar(star);
        listAppend(starList, star);
    }

    for(int i=0; i<starList.length; i++){
        setfillcolor(starList.elems[i].color);
         solidcircle(starList.elems[i].x, starList.elems[i].y,
starList.elems[i].radius);
    }

    IMAGE tortoise;//王八图片
    loadimage(&tortoise, "tortoise.jpg", 30, 30, false);
    putimage(SCREEN_WIDTH*4/10-30, SCREEN_HEIGHT-30, &tortoise);
    putimage(SCREEN_WIDTH*6/10, SCREEN_HEIGHT-30, &tortoise);

    while(quit==false){
        for(int i=0; i<starList.length; i++){
            MoveStar(starList, i);
        }
        /*if(isQuit()){
            quit = true;
        }*/
        if(starList.length==0) {
            quit = true;
        }

        Sleep(50);
    }

    system("pause");
    closegraph();
    return 0;
}
```

```
//star.h
#ifndef _STAR_H__
#define _STAR_H__


#define MAX_STAR 100

#define SCREEN_WIDTH   640
#define SCREEN_HEIGHT  480
#define MAX_STEP       5
#define MAX_RADIUS     3
#define BOTTOM_MARGIN  100

//星星状态
enum STATUS{
    STOP=0,
    UP,
    DOWN,
    LEFT,
    RIGHT,
    RANDOM,
    ALL_STATUS
};


struct STAR{
    int x;              //星星的 x 坐标
    int y;              //星星的 y 坐标
    enum STATUS stat;   //状态
    unsigned radius;    //星星的半径
    int step;           //每次跳跃的间隔
    int color;          //星星的颜色
};


typedef struct{
    struct STAR *elems; // 顺序表的基地址
    int length;         // 顺序表的长度
    int size;           // 顺序表的空间
}SqList;


//顺序表的接口
bool initList(SqList &L);
bool listAppend(SqList &L, struct STAR e);
```

```
bool listDelete(SqList &L, int i);
void destroyList(SqList &L);

#endif
```

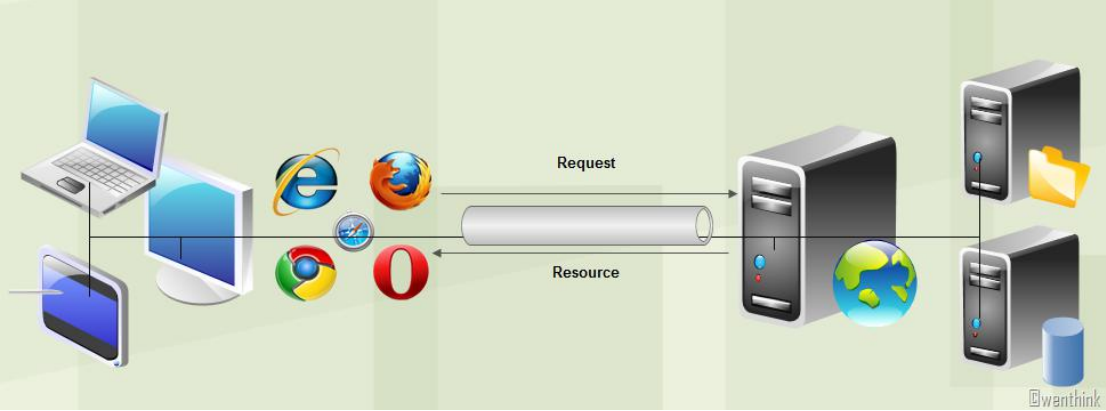
```
//starSqList.cpp
#include <iostream>
#include "star.h"

using namespace std;


bool initList(SqList &L){
    L.elems = new struct STAR[MAX_STAR];

    if(!L.elems) return false;

    L.length = 0;
    L.size = MAX_STAR;
    return true;
}


bool listAppend(SqList &L, struct STAR e){
    if(L.length == L.size) return false; //存储空间已满

    L.elems[L.length] = e;
    L.length++;   //表长加 1

    return true;
}


bool listDelete(SqList &L, int i){
    if(i<0 || i>=L.length) return false;

    if(i == L.length -1){//删除最后一个元素
        L.length--;
        return true;
    }

    for(int j=i; j<L.length -1; j++){
        L.elems[j] = L.elems[j+1];//删除位置的后续元素一次往前移
```

```
    }

    L.length--;
    return true;
}


void destroyList(SqList &L)
{
    if (L.elems) delete []L.elems;//释放存储空间
    L.length = 0;
    L.size = 0;
}

void listPrint(SqList &L){
    cout<<"顺序表容量 size: "<<L.size<<", 已保存元素的个数 length:
"<<L.length<<endl;
    for(int i=0; i<L.length; i++){
        cout<<"第"<<i+1<<" 颗星星: x="<<L.elems[i].x<<",
y="<<L.elems[i].y<<", radius="<<L.elems[i].radius<<endl;
    }
    cout<<endl;
}
```

第 4 节 顺序表的企业级应用案例

4.1 高并发 WEB 服务器中顺序表的应用

高性能的 web 服务器 Squid 每秒可处理上万并发的请求，从网络连接到服务器的客户端与服务器端在交互时会保持一种会话（和电话通话的场景类似）。服务器端为了管理好所有的客户端连接，给每个连接都编了一个唯一的整数编号，叫做文件句柄，简称 fd.



为了防止某些恶意连接消耗系统资源，当某个客户端连接超时（在设定的一定时间内没有发送数据）时，服务器就需要关闭这些客户端的连接

具体实现方案：

1. 当有新的请求连到服务器时，如果经过服务器频率限制模块判断，貌似恶意连接，则使用顺序表来保存此连接的超时数据，超时值使用时间戳来表示,时间戳是指格林威治时间 1970 年 01 月 01 日 00 时 00 分 00 秒(相当于北京时间 1970 年 01 月 01 日 08 时 00 分 00 秒)起至现在的总秒数，其结构体定义如下：

```
typedef struct {
    int fd ;
    time_t timeout; // 使用超时时刻的时间戳表示
```

}ConnTimeout;

2. 服务器程序每隔一秒钟扫描一次所有的连接，检查是否超时，如果存在超时的连接，就关闭连接，结束服务，同时将顺序表中的记录清除！