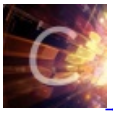
编程代码：用C语言来实现下雪效果，这个冬天，雪花很美

原创

MAX在码字 于 2020-11-02 16:24:48 发布 2695 收藏 12
分类专栏： C++ C 文章标签： C语言 知识分享 代码分享 学习教程 雪花效果

本文链接：https://blog.csdn.net/weixin_45713725/article/details/109451075



前言

1.本文主要围绕 如何 在 控制台上 下起 一场 只有自己能看见的雪

2.是个简易跨平台的,主要是C语言

3.动画 采用 1s 40帧, 雪花具有 x轴速度和y轴速度

4.比较简单,可以给学生作为C语言结课作业吧.

正文

1.1 先简单处理跨平台

本文写作动机,还是感谢一下大学的启蒙老师,让我知道了有条路叫做程序员,可以作为工作生存下去.那就上代码了.

首先代码定位 是 面向 简单跨平台,至少让 gcc 和 vs 能够跑起来.

其实跨平台都是噱頭, 说白了就是一些丑陋的宏. 真希望所有系统合二为一,采用统一的标准api 设计,但这是不可能的,就相当于很早之前的电视制式一样.

那么我们先看 围绕跨平台的宏

```
#include #include #include #include /** 时间 : 2015年12月26日11:43:22

* 描述 : 应该算过节吧,今天,写了个雪花特效 代码,

* *//** 清除屏幕的shell 命令/控制台命令,还有一些依赖平台的实现

* 如果定义了 __GNUC__ 就假定是 使用gcc 编译器,为Linux平台

*    否则 认为是 Window 平台*/#ifdefined(__GNUC__)//下面是依赖 Linux 实现#include #definesleep_ms(m) \        u

{

    inti = -1;

    while(++i

        printf("\033[1A");//先回到上一行  }#else// 创建等待函数 1s 60 帧 相当于 16.7ms => 1帧, 我们取16ms// 咱么

{

    COORD cr = {0,0};

    // GetStdHandle(STD_OUTPUT_HANDLE) 获取屏幕对象,  设置光标    SetConsoleCursorPosition(GetStdHandle(STD_OU

}#endif/*__GNUC__ 跨平台的代码都很丑陋 */
```

首先是 sleep_ms 这个宏, 传入一个毫秒数,让操作系统等待.

对于__curup 实现的不好. 功能是 让 控制台当前光标移动到 上面的 height 位置,对于 window直接移动到第一行(0,0)位置.

上面一共用了 5个头文件 还是容易的代码. string.h 主要用的是 memset 函数, 让一段内存初始化,用0填充.

对于time.h 主要是为了 初始化时间种子,方便每次运行都不一样.

// 初始化随机数种子,改变雪花轨迹srand((unsigned)time(NULL));

1.2 再说主业务代码

这里程序员运行的主业务,先说一说这里用的数据结构 如下

```
// 定义初始屏幕的宽高像素宏#define_INT_WIDTH        (100)#define_INT_HEIGHT        (50)// 屏幕刷新帧的速率#defin

* __FILE__        : 文件全路径

* __func__        : 函数名

* __LINE__        : 行数行

* __VA_ARGS__    : 可变参数宏,

* ##表示直接连接, 例如 a##b <=> ab*/#definecerr(msg,...) \    fprintf(stderr, "[%s:%s:%d]"msg"\n",__FILE__,__

*  frate  : 绘制一帧的周期, 单位是 毫秒

*  width  : 屏幕的宽,基于窗口的左上角(0,0)

*  height : 屏幕的高

*  pix    : 用一维模拟二维 主要结构如下

*           0 0 0 1 0 0 1 0 1 0

*           0 1 0 1 0 1 0 1 2 0

*           . . .

*           => 0表示没像素, 1表示1个像素,2表示2个像素....*/struct screen {

    intfrate;// 也可以用 unsigned 结构int width;

    int height;

    char*pix;

};
```

创建了一个绘图对象 struct screen 这里 构建这个结构体的时候用了下面一个技巧

//后面是 为 scr->pix 分配的内存 width*heightscr =malloc(sizeof(structscreen) +sizeof(char)*width*height);

一次分配两个内存空间.下面是主要实现的api 对象

```
/** 创建一个 屏幕结构指针 返回

*

* int frate    : 绘制一帧的周期

* int width    : 屏幕宽度

* int height    : 屏幕高度

* return        : 指向屏幕结构的指针

* */structscreen* screen_create(intfrate,intwidth,int height);/** 销毁一个 屏幕结构指针, 并为其置空

* struct screen** : 指向 屏幕结构指针的指针, 二级销毁一级的

* */voidscreen_destory(structscreen** pscr);/**

* 屏幕绘制函数,主要生成一个雪花效果

*

* struct screen* : 屏幕数据

* return          : 0表示可以绘制了,1表示图案不变*/intscreen_draw_snow(structscreen* scr);/**

* 屏幕绘制动画效果, 绘制雪花动画

*

* struct screen* : 屏幕结构指针*/voidscreen_flash_snow(structscreen* scr);
```

创建销毁, 绘制一个雪花界面, 绘制雪花动画效果的api. 其实都很相似,用opengl 库, 主要让我们省略了需要单独和操作系统显示层打交道工作.

这里介绍一下,个人 一个 简单避免 野指针的 的方法, 具体看下面实现

```
/** 销毁一个 屏幕结构指针, 并为其置空

* struct screen** : 指向 屏幕结构指针的指针, 二级销毁一级的

* */voidscreen_destory(structscreen** pscr)

{

    if(NULL == pscr || NULL == *pscr)

        return;

    free(*pscr);

    // 避免野指针*pscr = NULL;

}
```

在执行之后置空,因为C程序员对NULL一定要敏感,形成条件反射. 和大家开个玩笑 ,

请问：

C 语言中, NULL , 0,'\0',"0",false有什么异同？

欢迎同行,在招聘的时候问问,应聘初级开发工作者. 为什么C需要扣的那么细. 因为其它语言.你不明白是什么,

你可以用的很好. 但是C你写的代码,如果不知道会有怎样的结果,那么 线上就一大片服务器直接崩掉.而且还很难找出

问题所在. 因为C很简单,越简单就是越复杂.就越需要专业的维护人员.导致它成了'玩具'.

最后看一下 主业务

```
// 主函数,主业务在此运行intmain(intargc,char*argv[])

{

    structscreen* scr = NULL;

    //创建一个屏幕对象scr = screen_create(_INT_FRATE, _INT_WIDTH, _INT_HEIGHT);

    if(NULL == scr)

        exit(EXIT_FAILURE);

    //绘制雪花动画    screen_flash_snow(scr);

    //销毁这个屏幕对象screen_destory(&scr);

    return0;

}
```

还是非常容易看懂的, 创建一个屏幕对象,绘制雪花效果.销毁屏幕对象.

1.3 说一写 接口的实现细节

先看几个简单的api 实现,创建和销魂代码如下,很直白.

```
/** 创建一个 屏幕结构指针 返回

*

* int frate    : 绘制一帧的周期

* int width    : 屏幕宽度

* int height    : 屏幕高度

* return        : 指向屏幕结构的指针

* */structscreen*    screen_create(intfrate,intwidth,int height)
```

```
{
    structscreen *scr = NULL;
    if(frate<0|| width <=0|| height <=0) {
        cerr("[WARNING]check is frate<0 || width<=0 || height<=0 err!");
        return NULL;
    }
    //后面是 为 scr->pix 分配的内存 width*heightscr =malloc(sizeof(structscreen) +sizeof(char)*width*height);
    if(NULL == scr) {
        cerr("[FATALG]Out of memory!");
        return NULL;
    }
    scr->frate = frate;
    scr->width = width;
    scr->height = height;
    //减少malloc次数,malloc消耗很大,内存泄露呀,内存碎片呀scr->pix = ((char*)scr) +sizeof(struct screen);
    return scr;
}/** 销毁一个 屏幕结构指针, 并为其置空
* struct screen** : 指向 屏幕结构指针的指针, 二级销毁一级的
* */voidscreen_destory(structscreen** pscr)
{
    if(NULL == pscr || NULL == *pscr)
        return;
    free(*pscr);
    // 避免野指针*pscr = NULL;
}
```

后面说一下 如何 绘制 屏幕中雪花

主要算法 是

a.有个屏幕 w x h

b.屏幕从上面第一行 出雪花 , 出雪花 位置是随机的[0,w], 但是有个距离,这个距离内只有一个雪花

c.下一行 雪花 依赖上一行雪花的生成, 每个雪花在可以飘动的时候, 只能 在[-1,1] 范围内

d.实现动画 效果 就是 每画一帧就等待 一段时间

下面看具体一点的 a

//创建一个屏幕对象scr = screen_create(_INT_FRATE, _INT_WIDTH, _INT_HEIGHT);

scr对象就是我们的创建屏幕. _INT_WIDTH 和 _INT_HEIGHT 就是屏幕大小. 对于_INT_FRATE 表示绘制一帧时间.

b实现 代码如下:

```
//构建开头 的雪花,下面宏表示每 _INT_SHEAD 个步长,一个雪花,需要是2的幂//static 可以理解为 private, 宏,位操作代码多了确

{

    intr =0;

    //数据需要清空memset(snow,0, len);

    for (;;) {

        //取余一个技巧 2^3 - 1 = 7 => 111 , 并就是取余数intt = rand() & (_INT_SHEAD -1);

        if(r + t >= len)

            break;

        snow[r + t] =1;

        r += _INT_SHEAD;

    }

}#undef_INT_SHEAD
```

技巧如上,可以看说明. 这里 科普一下, 对于 for(;;) {} 和 while(true) {} 异同.

for(;;) {} 和 while(true) {} 这两段代码转成汇编是一样的, 不一样 的是 强加的意愿. 第一个 希望 跳过 检测步骤 速度更快一点.

再扩展一点.

```
//另一种 循环语句, goto 还是 很强大实用的__for_loop:

    if(false)

        goto __for_break;

goto __for_loop;

__for_break:
```

可以再扩展深一点, 还有一种 api 比 这个goto 还NB. 有机会分享. 特别强大, 是异常处理程序本质.

对于c.

```
//通过 上一个 scr->pix[scr->width*(idx-1)] => scr->pix[scr->width*idx]//下面的宏 规定 雪花左右摇摆 0 向左一个像素

{

    intwidth = scr->width;

    char* psnow = scr->pix + width*(idx -1);

    char* snow = psnow + width;

    inti, j, t;// i索引, j保存下一个瞬间雪花的位置,t 临时补得,解决雪花重叠问题

                //为当前行重置memset(snow,0, width);

    //通过上一次雪花位置 计算下一次雪花位置for(i =0; i

        for(t = psnow[i]; t>0; --t) {// 雪花可以重叠

                                        // rand()%_INT_SWING - 1 表示 雪花 横轴的偏移量,相对上一次位置j = i + ran

            j = j<0? width -1: j >= width ?0: j;// j如果越界了,左边越界让它到右边,右边越界到左边++snow[j];

        }

    }

}
```

下一行雪花 依赖 上一行雪花, 这里 有点像插入排序.

整体的绘制代码 如下

```
/**
* 屏幕绘制函数,主要生成一个雪花效果
*
* struct screen* : 屏幕数据
* return          : 0表示可以绘制了,1表示图案不变*/intscreen_draw_snow(structscreen* scr)
{
    // 静态变量,默认初始化为0,每次都共用staticint__speed =0;
    int idx;
    if(++__speed != _INT_VSNOW)
        return1;
    //下面 就是 到了雪花飘落的时刻了 既 __speed == _INT_VSNOW__speed =0;
    //这里重新构建雪花界面,先构建头部,再从尾部开始构建for(idx = scr->height -1; idx >0; --idx)
        __snow_next(scr, idx);
    //构建头部__snow_head(scr->pix, scr->width);
    return0;
}
```

绘制了一个屏幕对象的雪花. __speed 记录 绘制次数, _INT_VSNOW 控制绘制速率

d 的实现代码 如下

首先实现一个 销毁屏幕代码和 绘制代码

```
//buf 保存scr 中pix 数据,构建后为 (width+1)*height, 后面宏是雪花图案#define_CHAR_SNOW '*'staticvoid__flash_snow

{

    int i, j, rt;

    intheight = scr->height, width = scr->width;

    intfrate = scr->frate;//刷新的帧频率

                            //每次都等一下for (;;sleep_ms(frate)) {

        //开始绘制屏幕rt = screen_draw_snow(scr);

        if (rt)

            continue;

        for(i =0;i

            char* snow = scr->pix + i*width;

            for(j =0; j

                buf[rt++] = snow[j] ? _CHAR_SNOW :'';

            buf[rt++] ='\n';

        }

        buf[rt -1] ='\0';

        //正式绘制到屏幕上         puts(buf);

        //清空老屏幕,屏幕光标回到最上面         __curup(height);

    }

}#undef_CHAR_SNOW
```

这里 sleep_ms(frate); 是等待时间,否则太快, 人眼看不见.

绘制原理是 让屏幕转成控制台能够认识的字符. 塞入到buf 中.

__curup(height); 让绘制光标回到开头.

后面还有一段 代码实现

```
/**
* 屏幕绘制动画效果, 绘制雪花动画
*
* struct screen* : 屏幕结构指针*/voidscreen_flash_snow(structscreen* scr)
{
    char* buf = NULL;
    // 初始化随机数种子,改变雪花轨迹    srand((unsigned)time(NULL));
    buf =malloc(sizeof(char)*(scr->width +1)*scr->height);
    if(NULL == buf) {
        cerr("[FATAL]Out of memory!");
        exit(EXIT_FAILURE);
    }
    __flash_snow_buffer(scr, buf);
    //1.这里理论上不会执行到这,没加控制器. 2.对于buf=NULL,这种代码 可以省掉,看编程习惯free(buf);
    buf = NULL;
}
```

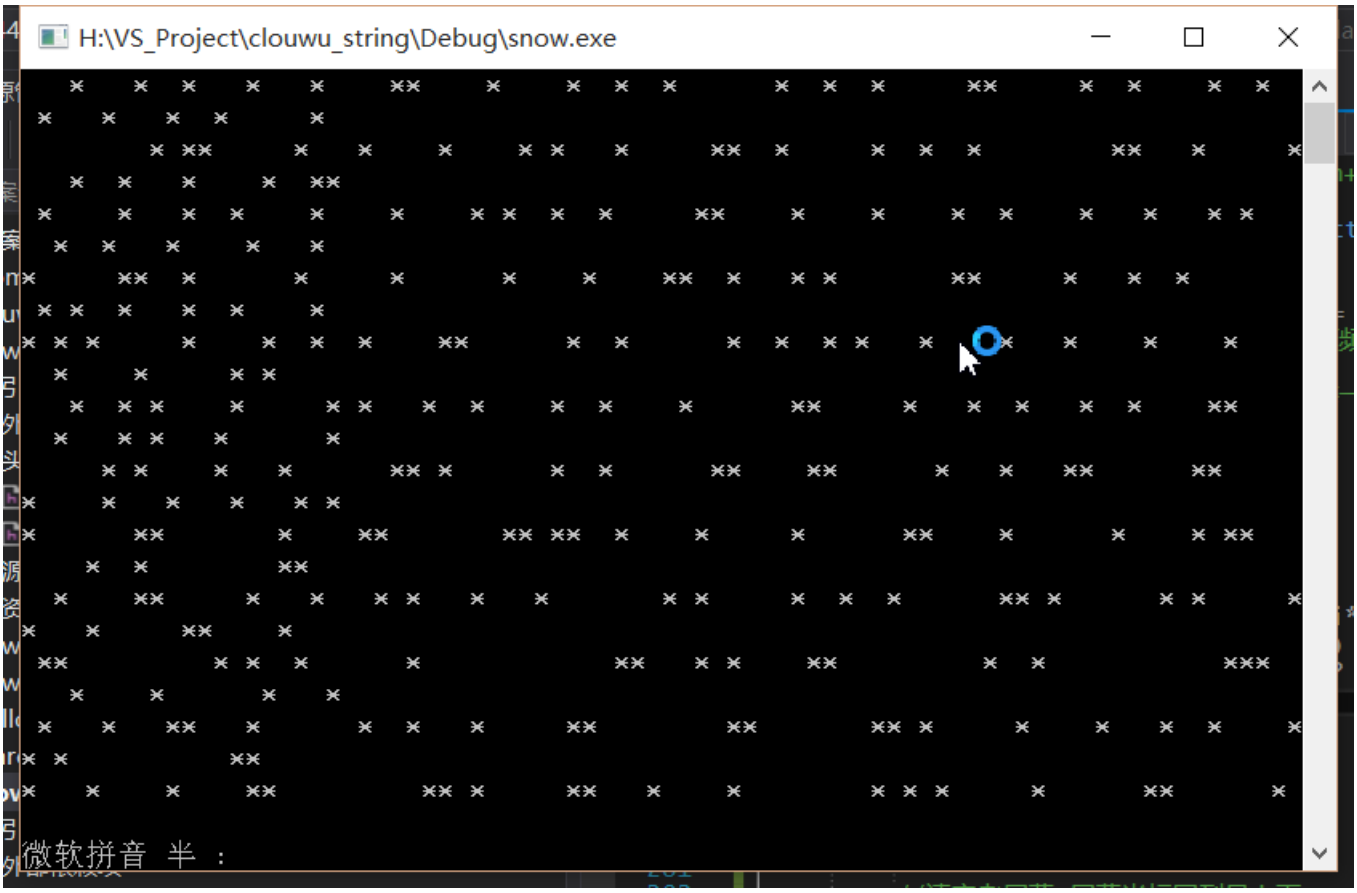
这种双函数实现一个功能技巧用的也很多. 例如写快速排序代码, 就是这样.

到这里 我们 设计和实现都完成了.

2.代码效果展示

2.1 window 上展示

使用VS新建一个控制台项目,F5就可以了效果如下



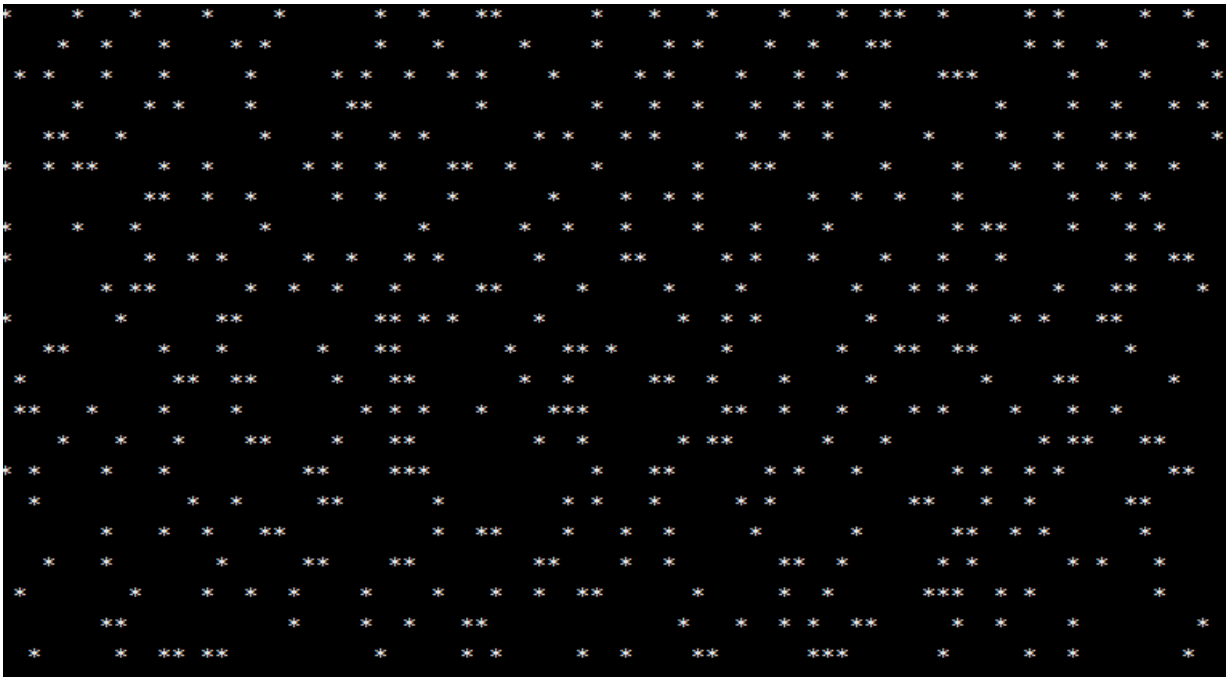
是动态的.

2.2 对于Linux

直接使用

gcc -g -Wall snow.c -o snow.out./snow.out

运行效果如下



到这里, C语言实现雪花效果就如上了.

2.3 完整的代码展示. 感谢 有你,一路同行.

```
#include #include #include #include /** 时间 : 2015年12月26日11:43:22

* 描述 : 应该算过节吧,今天,写了个雪花特效 代码,

*            送给 大学启蒙 芦老师

*                          学生王志 祝福上

* *//** 清除屏幕的shell 命令/控制台命令,还有一些依赖平台的实现

* 如果定义了 __GNUC__ 就假定是 使用gcc 编译器,为Linux平台

*    否则 认为是 Window 平台*/#ifdefined(__GNUC__)//下面是依赖 Linux 实现#include #definesleep_ms(m) \        u

{

    inti = -1;

    while(++i

        printf("\033[1A");//先回到上一行  }#else// 创建等待函数 1s 60 帧 相当于 16.7ms => 1帧, 我们取16ms// 咱么的

{

    COORD cr = {0,0};

    // GetStdHandle(STD_OUTPUT_HANDLE) 获取屏幕对象， 设置光标    SetConsoleCursorPosition(GetStdHandle(STD_OU

}#endif/*__GNUC__ 跨平台的代码都很丑陋 *// 定义初始屏幕的宽高像素宏#define_INT_WIDTH        (100)#define_INT_HEI

* __FILE__        : 文件全路径

* __func__        : 函数名

* __LINE__        : 行数行

* __VA_ARGS__    : 可变参数宏,

* ##表示直接连接, 例如 a##b <=> ab*/#definecerr(msg,...) \    fprintf(stderr, "[%s:%s:%d]"msg"\n",__FILE__,__

*  frate  : 绘制一帧的周期, 单位是 毫秒

*  width  : 屏幕的宽,基于窗口的左上角(0,0)

*  height : 屏幕的高

*  pix    : 用一维模拟二维 主要结构如下

*           0 0 0 1 0 0 1 0 1 0

*           0 1 0 1 0 1 0 1 2 0

*           . . .
```

```
*            => 0表示没像素, 1表示1个像素,2表示2个像素....*/struct screen {
    intfrate;// 也可以用 unsigned 结构int width;
    int height;
    char*pix;
};/** 创建一个 屏幕结构指针 返回
*
* int frate    : 绘制一帧的周期
* int width    : 屏幕宽度
* int height    : 屏幕高度
* return        : 指向屏幕结构的指针
* */structscreen* screen_create(intfrate,intwidth,int height);/** 销毁一个 屏幕结构指针, 并为其置空
* struct screen** : 指向 屏幕结构指针的指针, 二级销毁一级的
* */voidscreen_destory(structscreen** pscr);/**
* 屏幕绘制函数,主要生成一个雪花效果
*
* struct screen* : 屏幕数据
* return          : 0表示可以绘制了,1表示图案不变*/intscreen_draw_snow(structscreen* scr);/**
* 屏幕绘制动画效果, 绘制雪花动画
*
* struct screen* : 屏幕结构指针*/voidscreen_flash_snow(structscreen* scr);// 主函数,主业务在此运行intmain(intar
{
    structscreen* scr = NULL;
    //创建一个屏幕对象scr = screen_create(_INT_FRATE, _INT_WIDTH, _INT_HEIGHT);
    if(NULL == scr)
        exit(EXIT_FAILURE);
    //绘制雪花动画    screen_flash_snow(scr);
    //销毁这个屏幕对象screen_destory(&scr);
    return0;
}/** 创建一个 屏幕结构指针 返回
*
```

```
* int frate    : 绘制一帧的周期
* int width    : 屏幕宽度
* int height    : 屏幕高度
* return        : 指向屏幕结构的指针
* */structscreen*    screen_create(intfrate,intwidth,int height)
{
    structscreen *scr = NULL;
    if(frate<0|| width <=0|| height <=0) {
        cerr("[WARNING]check is frate<0 || width<=0 || height<=0 err!");
        return NULL;
    }
    //后面是 为 scr->pix 分配的内存 width*heightscr =malloc(sizeof(structscreen) +sizeof(char)*width*height);
    if(NULL == scr) {
        cerr("[FATALG]Out of memory!");
        return NULL;
    }
    scr->frate = frate;
    scr->width = width;
    scr->height = height;
    //减少malloc次数,malloc消耗很大,内存泄露呀,内存碎片呀scr->pix = ((char*)scr) +sizeof(struct screen);
    return scr;
}/** 销毁一个 屏幕结构指针, 并为其置空
* struct screen** : 指向 屏幕结构指针的指针, 二级销毁一级的
* */voidscreen_destory(structscreen** pscr)
{
    if(NULL == pscr || NULL == *pscr)
        return;
    free(*pscr);
    // 避免野指针*pscr = NULL;
```

```
}//构建开头 的雪花,下面宏表示每 _INT_SHEAD 个步长,一个雪花,需要是2的幂//static 可以理解为 private, 宏,位操作代码多了
{
    intr =0;
    //数据需要清空memset(snow,0, len);
    for (;;) {
        //取余一个技巧 2^3 - 1 = 7 => 111 , 并就是取余数intt = rand() & (_INT_SHEAD -1);
        if(r + t >= len)
            break;
        snow[r + t] =1;
        r += _INT_SHEAD;
    }
}#undef_INT_SHEAD//通过 上一个 scr->pix[scr->width*(idx-1)] => scr->pix[scr->width*idx]//下面的宏 规定 雪花左右
{
    intwidth = scr->width;
    char* psnow = scr->pix + width*(idx -1);
    char* snow = psnow + width;
    inti, j, t;// i索引, j保存下一个瞬间雪花的位置,t 临时补得,解决雪花重叠问题
                //为当前行重置memset(snow,0, width);
    //通过上一次雪花位置 计算下一次雪花位置for(i =0; i
        for(t = psnow[i]; t>0; --t) {// 雪花可以重叠
                                        // rand()%_INT_SWING - 1 表示 雪花 横轴的偏移量,相对上一次位置j = i + ran
            j = j<0? width -1: j >= width ?0: j;// j如果越界了,左边越界让它到右边,右边越界到左边++snow[j];
        }
    }
}/**
* 屏幕绘制函数,主要生成一个雪花效果
*
* struct screen* : 屏幕数据
* return          : 0表示可以绘制了,1表示图案不变*/intscreen_draw_snow(structscreen* scr)
{
```

```
    // 静态变量,默认初始化为0,每次都共用staticint__speed =0;

    int idx;

    if(++__speed != _INT_VSNOW)

        return1;

    //下面 就是 到了雪花飘落的时刻了 既 __speed == _INT_VSNOW__speed =0;

    //这里重新构建雪花界面,先构建头部,再从尾部开始构建for(idx = scr->height -1; idx >0; --idx)

        __snow_next(scr, idx);

    //构建头部__snow_head(scr->pix, scr->width);

    return0;

}//buf 保存scr 中pix 数据,构建后为 (width+1)*height, 后面宏是雪花图案#define_CHAR_SNOW '*'staticvoid__flash_snou

{

    int i, j, rt;

    intheight = scr->height, width = scr->width;

    intfrate = scr->frate;//刷新的帧频率

                            //每次都等一下for (;;sleep_ms(frate)) {

        //开始绘制屏幕rt = screen_draw_snow(scr);

        if (rt)

            continue;

        for(i =0;i

            char* snow = scr->pix + i*width;

            for(j =0; j

                buf[rt++] = snow[j] ? _CHAR_SNOW :'';

            buf[rt++] ='\n';

        }

        buf[rt -1] ='\0';

        //正式绘制到屏幕上         puts(buf);

        //清空老屏幕,屏幕光标回到最上面         __curup(height);

    }

}#undef_CHAR_SNOW/**

* 屏幕绘制动画效果, 绘制雪花动画
```

```
*

* struct screen* : 屏幕结构指针*/voidscreen_flash_snow(structscreen* scr)

{

    char* buf = NULL;

    // 初始化随机数种子,改变雪花轨迹    srand((unsigned)time(NULL));

    buf =malloc(sizeof(char)*(scr->width +1)*scr->height);

    if(NULL == buf) {

        cerr("[FATAL]Out of memory!");

        exit(EXIT_FAILURE);

    }

    __flash_snow_buffer(scr, buf);

    //1.这里理论上不会执行到这,没加控制器. 2.对于buf=NULL,这种代码 可以省掉,看编程习惯free(buf);

    buf = NULL;

}
```

后记

到这里就结束了,这次分享的比较简单,有兴趣的同学可以 看看, 推荐写一遍. 代码看不懂的时候,多歇歇,看得懂的时候,多写写,

就有套路了. 欢迎吐槽. 错误是在所难免的.

这个冬天,雪花很美,(。◠▽◠)