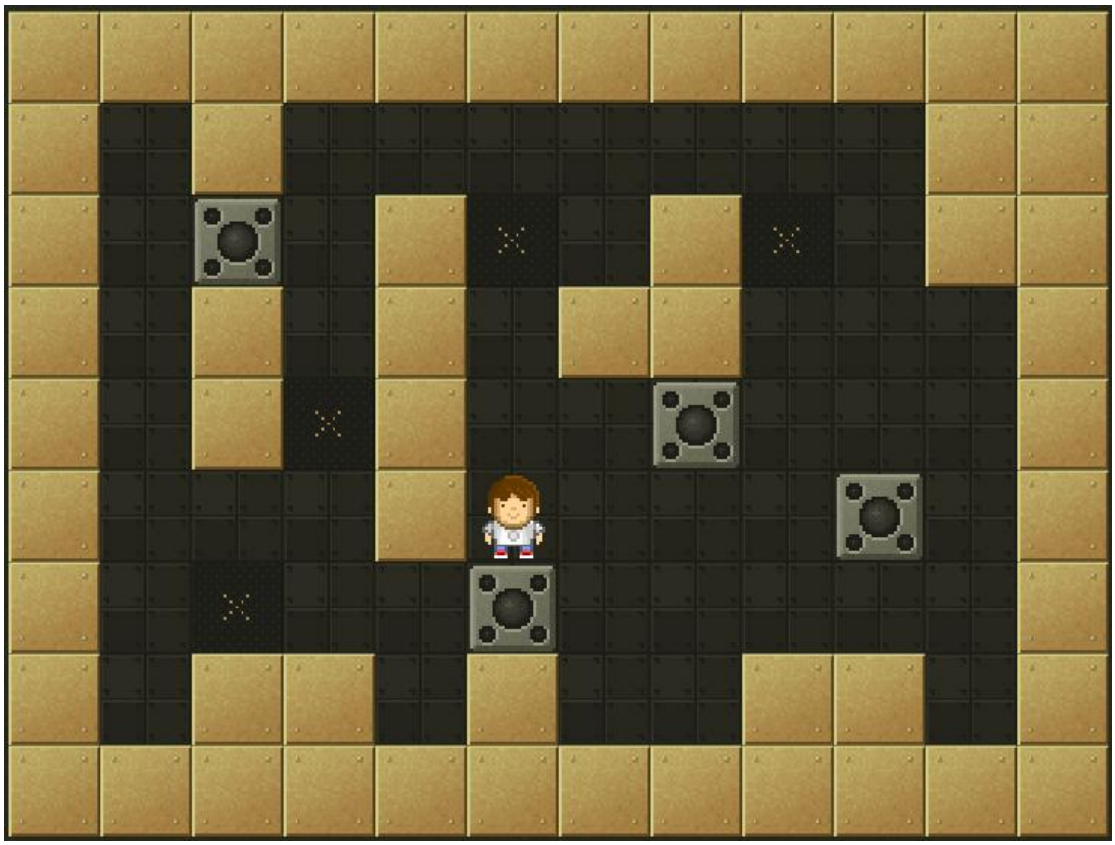
项目经理带你-零基础学习 C/C++
【从入门到精通】

补充项目 - 推箱子游戏

第 1 节 项目需求

实现一款推箱子游戏，效果如下图所示,具体规则：

1. 箱子只能推动而不能拉动；
2. 如果箱子前一格是地板或箱子目的地，则可以推动一个箱子往前走一格，如果箱子已经在箱子目的地则不能再推动；
3. 推箱子的小人不能从箱子目的地上直接穿过；
4. 注意不要把箱子推到死角上，不然就无法再推动它了；
5. 所有箱子都成功推到箱子目的地，游戏结束，过关成功！



美工资源：　链接：https://pan.baidu.com/s/1MZv8pDBXdNDbXxuAAPSM-A
提取码：2syq

图形库: www.easyx.cn

第 2 节　项目实现

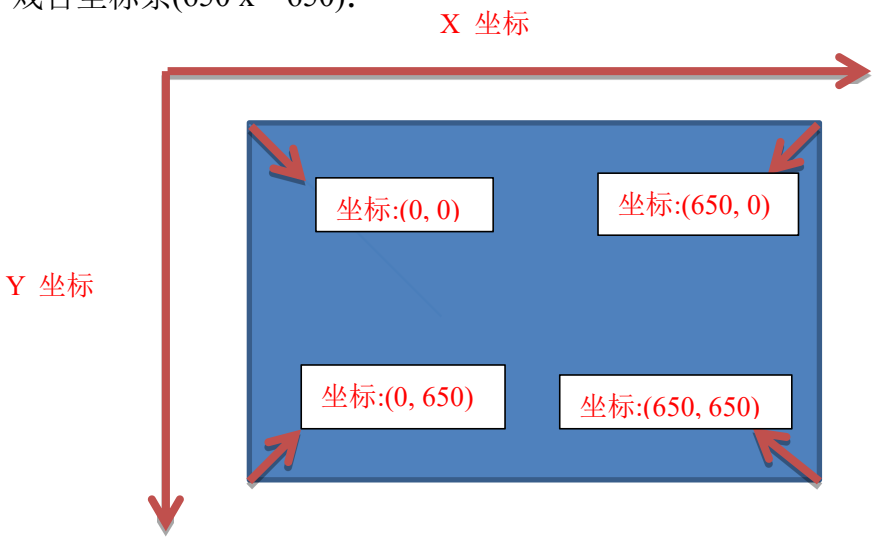
2.1　模块划分

（作用：1.化繁为简　2.适合团队协作　3.高质量代码）

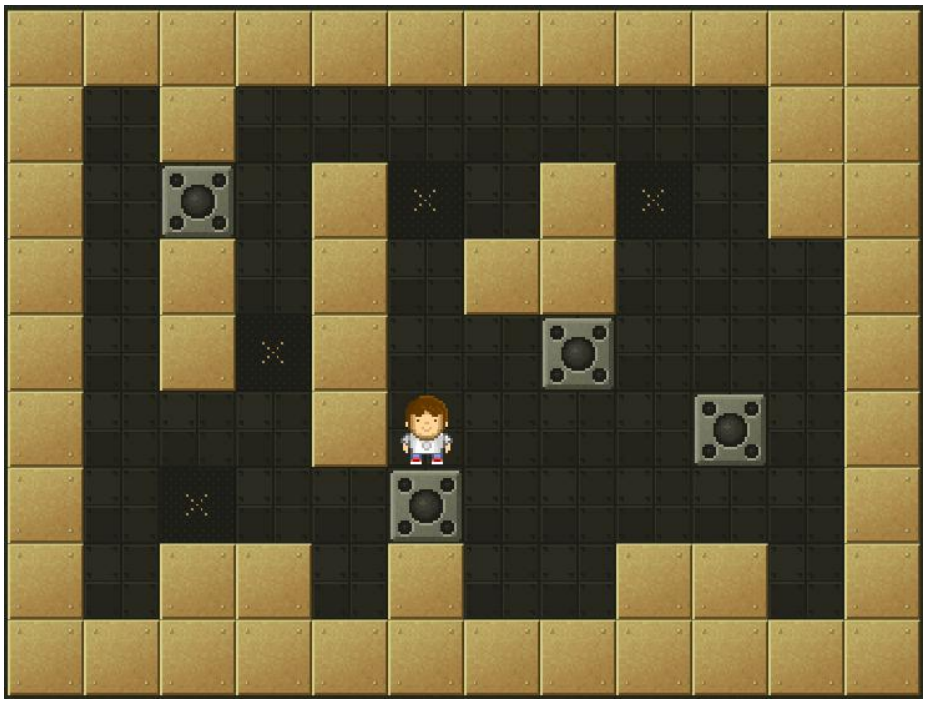
2.1.1 地图初始化

搭台唱戏

戏台坐标系(650 x　650)：



地图表示:



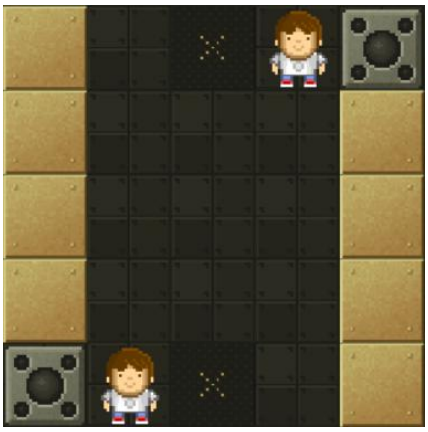
使用二维数组

- 游戏道具显示（墙、箱子、箱子目的地、小人、地板）
- 便于程序在游戏过程中进行逻辑判断和控制小人向前一步的动作控制
- 判断游戏结果

道具表示:

墙: 0，地板: 1，箱子目的地: 2，小人: 3， 箱子: 4，箱子命中目标: 5

```
/*游戏地图*/
int map[5][5] = {
{ 0,  1,  2,  3,  4 },
{ 0,  1,  1,  1,  0 },
{ 0,  1,  1,  1,  0 },
{ 0,  1,  1,  1,  0 },
{ 4,  3,  2,  1,  0 },
};
```



编码实现：

```
#include <graphics.h>
#include <iostream>
#include <stdlib.h>
#include <string>

using namespace std;

#define RATIO 61

#define SCREEN_WIDTH 960
#define SCREEN_HEIGHT 768


#define LINE   9
#define COLUMN 12

#define START_X  100
#define START_Y  150

IMAGE images[6];

/*游戏地图*/
int map[LINE][COLUMN] = {
{ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{ 0,  1,  0,  1,  1,  1,  1,  1,  1,  1,  0,  0 },
```

```
{ 0, 1, 4, 1, 0, 2, 1, 0, 2, 1, 0, 0 },
{ 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0 },
{ 0, 1, 0, 2, 0, 1, 1, 4, 1, 1, 1, 0 },
{ 0, 1, 1, 1, 0, 3, 1, 1, 1, 4, 1, 0 },
{ 0, 1, 2, 1, 1, 4, 1, 1, 1, 1, 1, 0 },
{ 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
};

int main(void){
    IMAGE bg_img;

    //搭台唱戏
    initgraph(SCREEN_WIDTH, SCREEN_HEIGHT);
    loadimage(&bg_img, _T("blackground.bmp"), SCREEN_WIDTH,
SCREEN_HEIGHT, true);
    putimage(0, 0, &bg_img);

    //加载道具图标
    loadimage(&images[0], _T("wall.bmp"), RATIO, RATIO, true);
    loadimage(&images[1], _T("floor.bmp"), RATIO, RATIO, true);
    loadimage(&images[2], _T("des.bmp"), RATIO, RATIO, true);
    loadimage(&images[3], _T("man.bmp"), RATIO, RATIO, true);
    loadimage(&images[4], _T("box.bmp"), RATIO, RATIO, true);
    loadimage(&images[5], _T("box.bmp"), RATIO, RATIO, true);

    for(int i = 0; i< LINE; i++){
        for(int j = 0; j < COLUMN; j++){
            putimage(START_X+j*RATIO, START_Y+i*RATIO,
&images[map[i][j]]);
        }
    }

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

2.1.2 热键控制

热键定义： 左 => a 下=> s 上=> w 右 => d 退出 => q

```
#include <conio.h>

//控制键 上、下、左、右 控制方向，'q' 退出
#define    KEY_UP        'w'    //char  'a'
#define    KEY_LEFT      'a'
#define    KEY_RIGHT     'd'
#define    KEY_DOWN      's'
#define KEY_QUIT      'q'

    //游戏环节
    bool quit = false;

    do {
        if(_kbhit()){ //玩家按键
            char ch = _getch();

            if(ch == KEY_UP){
                gameControl(UP);
            }else if(ch == KEY_DOWN){
                gameControl(DOWN);
            }else if(ch == KEY_LEFT){
                gameControl(LEFT);
            }else if(ch == KEY_RIGHT){
                gameControl(RIGHT);
            }else if(ch == KEY_QUIT){
                quit = true;
            }

        }
        Sleep(100);
    }while(quit==false); //!quit
```

2.1.3 推箱子控制

```
/**********************************************
*实现游戏四个方向（上、下、左、右）的控制
* 输入：
*  direct - 人前进方向
* 输出：  无
**********************************************/
void gameControl(enum _DIRECTION direct){

    struct _POS next_pos = man;
    struct _POS next_next_pos = man;
    switch(direct){
    case UP:
        next_pos.x--;
        next_next_pos.x-=2;
        break;
    case DOWN:
        next_pos.x++;
        next_next_pos.x+=2;
        break;
    case LEFT:
        next_pos.y--;
        next_next_pos.y-=2;
        break;
    case RIGHT:
        next_pos.y++;
        next_next_pos.y+=2;
        break;
    }
    //宏展开 next_pos.x>=0 && next_pos.x<LINE && next_pos.y>=0 &&
next_pos.y <COLUMN
    if( isValid(next_pos) && map[next_pos.x][next_pos.y] == FLOOR ){//
人的前方是地板
        changeMap(&next_pos, MAN); //小人前进一格
        changeMap(&man, FLOOR);
        man = next_pos;
    }else  if(isValid(next_next_pos) && map[next_pos.x][next_pos.y] ==
BOX){//人的前方是箱子
        //两种情况，箱子前面是地板或者是箱子目的地
        if( map[next_next_pos.x][next_next_pos.y] == FLOOR){
            changeMap(&next_next_pos, BOX);
            changeMap(&next_pos, MAN); //小人前进一格
```

```
            changeMap(&man, FLOOR);
            man = next_pos;
        }else if(map[next_next_pos.x][next_next_pos.y] == BOX_DES){
            changeMap(&next_next_pos, HIT);
            changeMap(&next_pos, MAN); //小人前进一格
            changeMap(&man, FLOOR);
            man = next_pos;
        }
    }
}
```

2.1.4 游戏结束

```
/**********************************************
*判断游戏是否结束，如果不存在任何一个箱子目的地，就代表游戏结束
*输入： 无
*返回值：
*     true - 游戏结束   false - 游戏继续
**********************************************/
bool isGameOver(){
    for(int i = 0; i< LINE; i++){
        for(int j = 0; j < COLUMN; j++){
            if(map[i][j] == BOX_DES) return false;
        }
    }
    return true;
}

/**********************************************
```

```
*游戏结束场景，在玩家通关后显示
*输入：
*      bg - 背景图片变量的指针
*返回值：  无
************************************************/
void gameOverScene(IMAGE *bg){
    putimage(0, 0, bg);
    settextcolor(WHITE);
    RECT rec = {0, 0, SCREEN_WIDTH, SCREEN_HEIGHT};
    settextstyle(20, 0, _T("宋体"));
    drawtext(_T("恭喜您~ \n 您终于成为了一个合格的搬箱子老司机！"),
&rec,  DT_CENTER | DT_VCENTER | DT_SINGLELINE);
}


//main 函数中
if(isGameOver()){
                gameOverScene(&bg_img);
                quit = true;
            }
```

第 3 节 项目精讲

1. 宏

3.1 为什么要使用宏

- 提高代码的可读性和可维护性
- 避免函数调用，提高程序效率

3.2 什么是宏

它是一种预处理器指令，在预编译阶段将宏名替换为后面的替换体 。

3.3 宏的定义

由三部分组成　　#define　　WIDTH　　960
预处理指令　宏名　　替换体(多行可用 \ 延续)

```
#include <stdio.h>
#include <stdlib.h>

#define  _width   1024   //宏命名规则同变量名
#define  ADDR     "中华人民共和国湖南\
省平江县"

int main(void){

    printf("width: %d\n", _width);
    printf("我的祖籍:  %s\n", ADDR);
    system("pause");
    return 0;
}
```

3.4 宏定义的使用

1. 不带参数的宏

2. 在宏中使用参数
   (请注意一下的区别:
   #define SQUARE(x) (x)*(x)
   #define SQUARE(x) x*x
)

```
#include <stdio.h>
#include <stdlib.h>

// 不带参数的宏
#define  _width   1024
#define  ADDR     "中华人民共和国湖南\
省平江县"
#define MARTIN 王鹏程

// 带参数的宏
#define SQUARE(x)   (x)*(x)
#define MAX(x,y)    x>y?x:y

int main(void){

    printf("width: %d\n", _width); // 宏展开相当于
printf("width: %d\n", 1024);
    printf("我的祖籍:  %s\n", ADDR);
    //printf("我的名字: %s", MARTIN); // 宏展开相当于 printf("我的名
字: %s", 王鹏程); 会报错

    int i = 10;
    int j = SQUARE(i);  // 宏展开  j = i*i;
    printf("j: %d\n", j);
    printf("MAX(i,j):%d\n", MAX(i, j)); //宏展开
printf("MAX(i,j):%d\n", i>j?i:j);
    int z = SQUARE(2+3);   // (2+3)*(2+3) = 25
    printf("z: %d\n", z);


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

2. 结构体

2.1 为什么要使用“结构”（结构体）

但需要表示一些复杂信息时，使用单纯的数据类型很不方便。

比如：学生信息（学号，姓名，班级，电话，年龄）

2.2 什么是“结构”

结构，就是程序员自定义的一种“数据类型”
是使用多个基本数据类型、或者其他结构，组合而成的一种新的“数据类型”。

2.3 结构的定义

```
struct   结构名 {
    成员类型   成员名;
    成员类型   成员名;
};
```

实例：

```
struct   student {
    char   name[16];
    int    age;
    char   tel[12];
};
```

特别注意：
1）要以 struct 开头
2）最后要使用分号
3）各成员之间用分号隔开

2.4 结构的初始化

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

//结构，就是程序员自定义的一种“数据类型”
struct student {
    char name[16];
    int  age;
    char tel[12];
};

//结构体包含结构体的情况
struct _class{
    struct student  rock;
    struct student  martin;
    struct student  zsf;
};


int main(void){
    //结构体的初始化
    //方式一  定义的时候初始化所有的属性
    struct student rock = {"Rock", 38, "******"};

    printf("rock 的姓名: %s  年龄: %d 电话: %s\n", rock.name, rock.age,
rock.tel);

    //方式二  定义的时候我们可以指定初始化的属性 VS/VC 不支持,但 gcc 是
支持的
    //struct student s1 ={.name="张三丰",.age = 100};

    //方式三  单独初始化每一个属性
    struct student s2;
    strcpy(s2.name, "杨过");
    s2.age = 40;
    s2.tel[0]='\0';

    printf("杨过的姓名: %s  年龄: %d 电话: %s\n", s2.name, s2.age,
s2.tel);

    //结构体中含有结构体
    struct _class  c1={{"Rock", 38, "******"},{"Martin", 38,
"18684518289"},{"张三丰",100,""}};
```

```
    printf("c1 班 martin 同学的姓名: %s  年龄: %d 电话: %s\n",
c1.martin.name, c1.martin.age, c1.martin.tel);

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

使用形式:
结构体变量.成员变量
中间用 . 分隔

2.5 结构的使用

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

struct student {
    char name[16];
    int  age;
};

int main(void){
    struct student  s1, s2;

    printf("请输入第一个学生的姓名: \n");
    scanf_s("%s",s1.name, sizeof(s1.name));
    printf("请输入第一个学生的年龄: \n");
    scanf("%d", &s1.age);

    printf("第一个学生的姓名: %s, 年龄: %d\n", s1.name, s1.age);

    //结构体的小秘密, 结构体变量之间可以直接赋值
    //s2 = s1;
    memcpy(&s2, &s1, sizeof(struct student));
    printf("第二个学生的姓名: %s, 年龄: %d\n", s2.name, s2.age);
    char c1[16]={"martin"}, c2[16];
    //c2 = c1; //数组不能直接赋值
```

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

2.6 结构数组

struct 结构名 变量名[数组大小]

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


struct student {
    char name[16];
    int  age;
};

int main(void){
    struct student s[2];

    printf("请输入第一个学生的姓名：\n");
    scanf_s("%s",s[0].name, sizeof(s[0].name));
    printf("请输入第一个学生的年龄：\n");
    scanf("%d", &s[0].age);

    printf("第一个学生的姓名: %s, 年龄: %d\n", s[0].name, s[0].age);

    //结构体的小秘密，结构体变量之间可以直接赋值
    s[1] = s[0];
    memcpy(&s[1],&s[0], sizeof(struct student));
    printf("第二个学生的姓名: %s, 年龄: %d\n", s[1].name, s[1].age);

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

2.7 指向结构体的指针

使用结构体变量址指针访问结构体成员要使用 -> 符号

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

struct _friend{
    char name[32];
    char sex; // m - 男  f - 女
    int  age;
};

int main(void){
    struct _friend girl = {"小龙女", 'f', 18};

    struct _friend *my_girl = &girl;

    printf("小龙女的名字：%s, 性别：%s  年龄：%d\n", girl.name,
girl.sex=='m'?"男":"女", girl.age);

    //指针访问结构体变量的成员，有两种方式
    //方式 1. 直接解引
    printf("小龙女的名字：%s, 性别：%s  年龄：%d\n", (*my_girl).name,
(*my_girl).sex=='m'?"男":"女", (*my_girl).age);

    //方式 2. 直接使用指针访问  ->
    printf("小龙女的名字：%s, 性别：%s  年龄：%d\n", my_girl->name,
my_girl->sex=='m'?"男":"女", my_girl->age);
    system("pause");
    return 0;
}
```

实际开发中我们常用 -> 符通过指针去访问结构体指针变量的成员

2.8 使用结构体传递值

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

struct programer{
    char  name[32];
    int   age;
    int   salary;
};

//形参是结构体变量，值传递
struct programer add_salary(struct programer p, int num){
    p.salary += num;
    return p;
}

//形参使用结构体指针
void add_salary1(struct programer *p, int num){
    if(!p) return ;
    p->salary += num;
}

//形参使用引用
void add_salary2(struct programer &p, int num){

    p.salary += num;
}

//形参是结构体变量，值传递,返回引用
struct programer& add_salary3(struct programer p, int num){
    p.salary += num;
    return p;
}

int main(void){
    struct programer xiaoniu;

    strcpy(xiaoniu.name, "小牛");
    xiaoniu.age = 28;
    xiaoniu.salary = 20000;

    //结构体变量做为参数传值是值传递,和 int 等基本类型一样
     //xiaoniu = add_salary(xiaoniu, 5000);
```

```
    //指针传值
    //add_salary1(&xiaoniu, 5000);

    //引用传值
    //add_salary2(xiaoniu, 10000);

     //返回引用
     xiaoniu = add_salary3(xiaoniu, 20000);

    printf("姓名: %s, 年龄:%d, 薪水:%d\n", xiaoniu.name, xiaoniu.age,
xiaoniu.salary);

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

注意:
一般不建议把结构体直接作为函数参数。
因为结构体的 size 比较大，直接传递，消耗性能!
解决方案:（使用指针和引用，优先使用引用）

3. 枚举

2.1 枚举的概念

枚举是 C/C++语言中的一种基本数据类型，它可以用于声明一组常数。当一个变量有几个固定的可能取值时，可以将这个变量定义为枚举类型。
比如，你可以用一个枚举类型的变量来表示季节，因为季节只有 4 种可能的取值：春天、夏天、秋天、冬天。

2.2 枚举类型的定义

一般形式为：enum　枚举名　{枚举元素 1,枚举元素 2,……};
enum Season {spring, summer, autumn, winter};

2.3 枚举变量的定义

前面只是定义了枚举类型，接下来就可以利用定义好的枚举类型定义变量，跟结构体一样，有 3 种方式定义枚举变量
1.先定义枚举类型，再定义枚举变量

```
enum Season {
    spring,
    summer,
    autumn,
    winter
};
enum Season s;
```

2.定义枚举类型的同时定义枚举变量

```
enum Season {
    spring,
    summer,
    autumn,
    winter
  } s;
```

3.省略枚举名称，直接定义枚举变量

```
enum {
    spring,
    summer,
    autumn,
    winter
} s;
```

上面三种方式定义的都是枚举变量 s

2.4 枚举使用的注意

- ✓ C 语言编译器会将枚举元素(spring、summer 等)作为整型常量处理，称为枚举常量。

- ✓ 枚举元素的值取决于定义时各枚举元素排列的先后顺序。默认情况下，第一个枚举元素的值为 0，第二个为 1，依次顺序加 1。

```
#include <stdio.h>

int main()
 {
    // 1.定义枚举类型
    enum Season
    {
        spring,
         summer,
         autumn,
         winter
     };

    // 2.定义枚举变量
    enum Season s =    winter;
    printf("%d\n", s);
    return 0;
 }
```

打印结果为：**3**
也就是说 spring 的值为 0，summer 的值为 1，autumn 的值为 2，winter 的值为 3

- ✓ 也可以在定义枚举类型时改变枚举元素的值

```
#include <stdio.h>

int main()   {
       // 1.定义枚举类型
       enum Season{
          spring = 1,
          summer,
          autumn,
          Winter};
```

```
        //2.定义枚举变量
        enum Season s =   winter;
        printf("%d\n", s);
        return 0;
 }
```

打印结果为：4
没有指定值的枚举元素，其值为前一元素加 1。

2.5、枚举变量的基本操作

1.赋值

可以给枚举变量赋枚举常量或者整型值

```
#include <stdio.h>

int main() {
      // 1.定义枚举类型
        enum Season {spring, summer, autumn, winter} s;

        // 2.定义枚举变量
        s = spring; //  等价于  s = 0;
        printf("%d\n", s);
        s = winter;//等价于 s = 3;
        printf("%d\n", s);
        return 0;
 }
```

打印结果　0
　　3

2.遍历枚举元素

```
enum Season {spring, summer, autumn, winter} s;

//  遍历枚举元素
for (s = spring; s <= winter; s++) {
     printf("枚举元素：%d \n", s);
}
```

输出结果：　枚举元素：0
　　枚举元素：1
　　枚举元素：2
　　枚举元素：3

4. 类型定义

3.1 什么是类型定义

typedef 是一个高级数据特性，它可以为某一**类型**自定义名称，即类型的别名。

3.2 为什么要使用类型定义

从一辆豪车说起： 奇瑞捷豹路虎揽胜极光

1. 简化写法
2. 提高程序的可移植性

```
//64 位 linux 系统
#include <stdio.h>
#include <stdlib.h>

typedef    long    int64;

int main(void){

        int64 dream = 10000000000; //梦想一百亿
        printf("dream: %lld\n", dream);
        printf("sizeof(int64): %d\n", sizeof(int64));
        return 0;
}
```

打印结果

```
dream: 10000000000
sizeof(int64): 8
```

```c
//32 window  系统
#include <stdio.h>
#include <stdlib.h>

typedef long long  int64;


int main(void){

    int64 dream = 10000000000; //梦想一百亿
    printf("dream: %lld\n", dream);
     printf("sizeof(int64): %d\n", sizeof(int64));
    system("pause");
    return 0;
}
```

打印结果

```
dream: 10000000000
sizeof(int64): 8
```

3.3 类型定义的使用

```c
#include <stdio.h>
#include <stdlib.h>

typedef char * STRING;
#define  STR   char *

int main(void){
    STRING s1, s2;  //等同于 char *s1; char *s2;
    char name[] = "Martin";
    s1 =  name;
```

```
    s2 = name;

    STR  s3, s4; //  char *  s3, s4;
    s3 = name;
    s4 = name[0];

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

类型定义和宏定义有相似之处，但不能混为一谈

5. 头文件 #include

5.1 什么是 #include

预处理指令 - 编译前包含指定文件内容到当前文件中，即使用包含文件替换源文件中的#include 指令。

#include 指令有两种形式：

#include <stdio.h>　　←文件名在尖括号中　<标准系统目录>
#include "box_man.h"　　←文件名在双引号中 <当前目录>

5.2 头文件的作用

- 代码重用
- 封装 - 把某些具有共性的函数定义放在同一个源文件里
- 提高代码的可维护性

5.3 头文件的使用

test.h

```
//#pragma once    //第一种 文件只包含一次
#ifndef TEST_H    //第二种 #ifndef 和 #endif 包围的代码只包含一次
#define TEST_H


#include <stdio.h>

struct _pos{
    int x;
    int y;
    int z;
    int w;
};

void test_A();
void test_B();
```

```
extern int kkk;
#endif
```

test_A.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include "test.h"

//程序员 A 的代码

int main(void){
    struct _pos  pos;
    pos.x = 0;
    pos.y = 0;
    pos.z = 0;
    pos.w = 0;

    printf("kkk: %d\n", kkk);

    test_A();
    test_B();
    system("pause");
}


void test_A(){
    printf("我是 test_A.cpp => test_A() \n");
    {
        static int time=0;
        if(++time>5)return ;
    }
    test_B();
}
```

test_B.cpp

```
#include <stdio.h>
#include "test.h"

int kkk = 100;

extern void test_B(){
    struct _pos pos;
    pos.x = 0;
```

```
    pos.y = 0;
    pos.w = 0;
    printf("我是 test_B.cpp => test_B() \n");
    test_A();
}
```

5.4 头文件保护措施

#pragma once　　//防止整个头文件被包含多次

#ifndef … #define　…#endif　　//防止#ifndef 和#endif 包围的代码包含多次

第 4 节 项目练习

（一） 在现有推箱子项目实现的基础上实现如下功能：

1. 人可以从箱子目的地上走过
2. 人可以把箱子从箱子目的地上推开

可选： 统计人走的步数，超过指定的步数（比如 40 步）游戏失败，显示失败界面