【注意:】
1、 本次作业只允许使用到目前为止讲授过的内容（第 7 章及以前）及已完成作业中的补充概念
2、 已学过的知识中，不允许使用 goto，不允许使用 C++的 string 变量
3、 C++方式不允许使用 scanf/printf 进行输入/输出
4、 所有输入均需要考虑输入错误的情况，包括同型数据不在指定范围内（例：要求输入[1..12]但输入-2/13 等）以及输入了异型数据（例：需要正整数但输入字符）的情况
5、 整个程序，不允许使用任何形式的全局变量/数组/指针，允许使用全局宏定义或常变量
6、 作业必须符合相应的缩进格式，格式分占 10%

综合题 3：配置文件工具函数集的实现及前次综合题从配置文件中读取信息

【背景描述:】在 Windows 和 Linux 操作系统中，很多应用程序都有相应的配置文件，用来设定程序运行过程中的各个选项，配置文件的结构说明如下：

```
;这是某程序的配置文件
;2018.04.14 修订

[VideoProperties]
Title=属性设置
Title_V=10

[SpecialEffect]
Title=特效
EffectBlock=12.3 #版本
ZoomBlock=

[FaceTrack]
   Title =    人脸追踪
FaceTrackingBlock=y

#FaceTrack=3
```

```
;这是某程序的配置文件
;2018.04.14 修订

Title_V=10
EffectBlock=12.3 #版本
ZoomBlock=
   Title =    人脸追踪
FaceTrackingBlock=y
#FaceTrack=3
```

★ 配置文件分为若干组，每组用[***]表示组名，组名各不相同

★ 每组有若干项，每项的基本格式是“项目名=值”，同组的项目名不相同，不同组可能相同
- 项目名也可能是中文
- 每个项目一行，不允许多项目一行

★ 值的可能取值有：整数、浮点数、单字符、字符串、空
- 字符串可能为字母、数字、中文、符号等
- 字符串不含空格，tab 键等不可显示字符
- 字符串不含 TAB、;、#、″、’、{}、[]、()、=等特殊含义字符(均为半角字符)
- 项目名及值得前后允许有空格、tab 等，不包含在内，也不算错误(左侧例子中[FaceTrack]仍为 Title=人脸追踪)

★ 如果某行出现;或#(均为半角)，则表示该符号出现至本行尾部均为注释(左侧红色)，不需要符合语法要求，也不被读取

★ 某些配置文件，可能只有项目名，没有组名，下文中称为简单配置文件

【工具函数的定义(每个定义均为两个，一个是 C++方式，一个是 C 方式)】

★ int group_add(fstream &fp, const char *group_name)

★ int group_add(FILE *fp, const char *group_name)

示例说明：

在 main 函数中调用 group_add(fp, "test");，则表示在配置文件的加入[test]组，组中暂时无内容

- 增加成功返回 1，否则返回 0
- 如果[test]组已存在，则不能重复增加，直接返回 0 即可
- 加入的组放在文件的最后

★ int group_del(fstream &fp, const char *group_name)

★ int group_del(FILE *fp, const char *group_name)

示例说明：

在 main 函数中调用 group_del(fp, "test");，则表示在配置文件中删除[test]组及该组下存在的全部项

- 删除成功返回 1，否则返回 0
- 如果[test]组不存在，直接返回 0 即可
- 如果[test]组重复存在（例如：手工修改使两组同名），则要删除所有同名组并返回 1

★ int item_add(fstream &fp, const char *group_name, const char *item_name, const void *item_value, const enum ITEM_TYPE item_type)

★ int item_add(FILE *fp, const char *group_name, const char *item_name, const void *item_value, const enum ITEM_TYPE item_type)

说明： item_add 函数，C++方式还可以用类似于 int item_add(fstream &fp, const char *group_name, const char *item_name, const T item_value)这种更简单的形式（类型 T 为 int/double/char/char */NULL 等),具体实现方式可以是函数重载或函数模板，但为了保证 C/C++ 方式的工具函数集对使用者呈现出一致性，此处仍采用兼容 C 方式的函数定义

示例说明：

1、假设在 main 函数中有：

```
int i = 12345;
item_add(fp, "test", "起始值", &i, TYPE_INT);
```

则表示在配置文件的[test]组最后加入 "起始值=12345" 项

2、假设在 main 函数中有：

```
double d = 123.45;
item_add(fp, "test", "起始值", &d, TYPE_DOUBLE);
```

则表示在配置文件的[test]组最后加入 "起始值=123.45" 项

3、假设在 main 函数中有：

```
char *s="今天是个好日子";
item_add(fp, "test", "起始值", s, TYPE_STRING);
```

则表示在配置文件的[test]组最后加入 "起始值=今天是个好日子" 项

4、假设在 main 函数中有：

```
char c = 'Y';
item_add(fp, "test", "起始值", &c, TYPE_CHARACTER);
```

则表示在配置文件的[test]组最后加入 "起始值=Y" 项

5、假设在 main 函数中有：

```
item_add(fp, "test", "起始值", NULL, TYPE_NULL);
```

则表示在配置文件的[test]组最后加入 "起始值=" 项

- 增加成功返回 1，否则返回 0

- 如果[test]组不存在，直接返回 0 即可
- 如果存在多个[test]组（例如：手工修改使存在多个[test]），则在位置靠前的组中增加本项并返回 1 即可
- 如果[test]组中的"起始值"已存在，则不能重复增加，直接返回 0 即可
- 如果组名为 NULL（例：item_add(fp, NULL, "起始值", NULL, TYPE_NULL);则表示在简单配置文件的最后加入 "起始值=" 项
- item_type 的类型定义为：enum ITEM_TYPE { TYPE_INT = 0, TYPE_DOUBLE, TYPE_STRING, TYPE_CHARACTER, TYPE_NULL};
- 因为数据类型错误导致运行出错，不算错误（例如：TYPE_STRING 给了一个 double 地址，因为无尾零将配置文件写乱；TYPE_INT 给了一个 short 型地址，导致写入的 int 型数据不正确等）

★ int item_del(fstream &fp, const char *group_name, const char *item_name)
★ int item_del(FILE *fp, const char *group_name, const char *item_name)
- 示例说明：假设在 main 函数中有：item_del(fp, "test", "起始值");，则表示在配置文件的[test]组中删除"起始值=***" 项
- 删除成功返回 1，否则返回 0
- 如果[test]组不存在，直接返回 0 即可
- 如果[test]组存在，但要删除的"起始值"项不存在，则直接返回 0 即可
- 如果[test]组存在，但要删除的"起始值"项重复存在（例如：手工修改使存在多个"起始值"），则删除该组所有同名项并返回 1（注意：不能删除其它组的同名项）
- 如果组名为 NULL（例：item_del(fp, NULL, "起始值");则表示在简单配置文件中删除所有同名项

★ int item_update(fstream &fp, const char *group_name, const char *item_name, const void *item_value, const enum ITEM_TYPE item_type)
★ int item_update(FILE *fp, const char *group_name, const char *item_name, const void *item_value, const enum ITEM_TYPE item_type)

说明：item_update 函数,C++方式还可以用类似于 int item_update(fstream &fp, const char *group_name, const char *item_name, const T item_value)这种更简单的形式（类型 T 为 int/double/char/char */NULL 等），具体实现方式可以是函数重载或函数模板，但为了保证 C/C++ 方式的工具函数集对使用者呈现出一致性，此处仍采用兼容 C 方式的函数定义

示例说明：

1、假设在 main 函数中有：
```
int i = 12345;
item_update(fp, "test", "起始值", &i, TYPE_INT);
```
则表示在配置文件的[test]组将"起始值=***"项更新为"起始值=12345"；
若"起始值"项不存在，则在最后加入"起始值=12345"项

2、假设在 main 函数中有：
```
double d = 123.45;
item_update(fp, "test", "起始值", &d, TYPE_DOUBLE);
```
则表示在配置文件的[test]组将"起始值=***"项更新为"起始值=123.45"；
若"起始值"项不存在，则在最后加入"起始值=123.45"项

3、假设在 main 函数中有：
```
char *s="今天是个好日子";
item_update(fp, "test", "起始值", s, TYPE_STRING);
```
则表示在配置文件的[test]组将"起始值=***"项更新为"起始值=今天是个好日子"；
若"起始值"项不存在，则在最后加入"起始值=今天是个好日子"项

4、假设在 main 函数中有：
char c = 'Y';
item_update(fp, "test", "起始值", &c, TYPE_CHARACTER);
则表示在配置文件的[test]组将"起始值=***"项更新为"起始值=Y"；
若"起始值"项不存在，则在最后加入"起始值=Y"项

5、假设在 main 函数中有：
item_update(fp, "test", "起始值", NULL, TYPE_NULL);
则表示在配置文件的[test]组将"起始值=***"项更新为"起始值="；
若"起始值"项不存在，则在最后加入"起始值="项

- 更新/新增成功返回 1，否则返回 0
- 如果[test]组不存在，直接返回 0 即可
- 如果存在多个[test]组（例如：手工修改使存在多个[test]），则在位置靠前的组中更新/增加本项并返回 1 即可
- 如果[test]组存在，但要更新的项"起始值"不存在，则加入在该组最后，返回 1 即可
- 如果[test]组存在，但要更新的项"起始值"重复存在（例如：手工修改使存在多个"起始值"），则更新位置靠前的一项并删除本组其它同名项，返回 1 即可
- 更新项前后的数据类型允许不同（例："起始值=Y"更新为"起始值=12345",反之亦可）

★ int item_get_value(fstream &fp, const char *group_name, const char *item_name, void *item_value, const enum ITEM_TYPE item_type)

★ int item_get_value(FILE *fp, const char *group_name, const char *item_name, void *item_value, const enum ITEM_TYPE item_type)

示例说明：

1、假设在 main 函数中有：
int i;
item_get_value(fp, "test", "起始值", &i, TYPE_INT);
如果配置文件的[test]组有"起始值=12345"，则调用后 i 值是 12345

2、假设在 main 函数中有：
double d;
item_get_value(fp, "test", "起始值", &d, TYPE_DOUBLE);
如果配置文件的[test]组有"起始值=123.45"，则调用后 d 值是 123.45

3、假设在 main 函数中有：
char s[80];
item_get_value(fp, "test", "起始值", s, TYPE_STRING);
若配置文件的[test]组有"起始值=今天是个好日子"，则调用后 s 值是"今天是个好日子"

4、假设在 main 函数中有：
char c;
item_get_value(fp, "test", "起始值", &c, TYPE_CHARACTER);
如果配置文件的[test]组有"起始值=Y"，则调用后 c 的值是'Y'

5、假设在 main 函数中有：
item_get_value(fp, "test", "起始值", NULL, TYPE_NULL);
如果配置文件的[test]组有"起始值=***"（任意项），则调用后函数返回 1，否则返回 0

- 取值成功返回 1，否则返回 0
- 如果[test]组不存在，直接返回 0 即可，不要改变传入的 void *item_value 的值
- 如果存在多个[test]组（例如：手工修改使存在多个[test]），则在位置靠前的组中取本项的值，根据存在与否返回相应值，不再考虑后续同名组
- 如果[test]组中"起始值"不存在，直接返回 0 即可，不要改变传入的 void *item_value 的值
- 如果[test]组中"起始值"存在多项，则取位置考前的项即可，不再考虑后续项

- 当取值类型为 TYPE_NULL 时，不做任何操作，返回 1/0 即可
- 因为数据类型错误导致运行出错，不算错误（例如：TYPE_INT 给了一个 short 型地址，导致系统弹窗报错；TYPE_STRING 给个一个未指向确定空间的 char *值/不足以容纳整个字符串的一维字符数组；TYPE_INT 给出 int 型地址，但是同名项有多个，第一个不是 int 型数据但第三个是 int 型等）

【公共函数集实现要求：】

1、用 C++/C 语言的文件读写方式分别完成

2、本次作业项目名称为 90-b3，与之前的 90-b0、90-b1、common 并列，且本次作业的源文件要放到两个目录中

```
解决方案目录（名称可自行定义）
|-- 90-b0              （“合成十”项目的目录，必须是此名称）
|   |-- 文件与原来相同，略
|   `-- 其他 VS 产生的文件及子目录
|-- 90-b1              （“消灭星星”项目的目录，必须是此名称）
|   |-- 文件与原来相同，略
|-- 90-b2              （“数字俄罗斯方块”项目的目录，必须是此名称）
|   |-- 文件与原来相同，略
|   `-- 其他 VS 产生的文件及子目录
|-- 90-b3              （本次作业所建项目的目录，必须是此名称）
|   |-- 90-b3.cpp      （为了测试本次作业的公共函数集而写的程序）
|   |                   注意：C++方式与 C 方式实现互斥，不能同时存在于一个项目中
|   |                     例 1：90-b3.cpp 中 #include "../common/common_readconfig.h"
|   |                           项目中包含 "../common/common_readconfig.c"
|   |                          即可使用 C 方式的工具函数集
|   |                     例 2：90-b3.cpp 中 #include "../common/common_readconfig.hpp"
|   |                           项目中包含 "../common/common_readconfig.cpp"
|   |                          即可使用 C++方式的工具函数集
|   |                     要求：90-b3.cpp 其余部分不做修改的情况下，程序运行结果不变
|   `-- 其他 VS 产生的文件及子目录
|-- common             （公共函数目录，必须是此名称）
|   |-- cmd_console_test.cpp
|   |-- cmd_console_tools.cpp
|   |-- cmd_console_tools.h
|   |-- common.h
|   |-- common_adv.cpp
|   |-- common_base.cpp
|   |-- common_readconfig.h       本次作业公共函数集的头文件说明（C 方式）
|   |-- common_readconfig.c       本次作业公共函数集的具体实现（C 方式）
|   |-- common_readconfig.hpp     本次作业公共函数集的头文件说明（C++方式）
|   |-- common_readconfig.cpp     本次作业公共函数集的具体实现（C++方式）
|   |                             再次申明：C/C++方式的公共函数集不同时出现在一个项目中
`   `-- 其他 VS 产生的文件及子目录
```

3、90-b3.cpp 要求能够同学间双向验证（即其他人的 90-b3.cpp 和你的公共函数集放入同一个项目中，运行结果与你自己的 90-b3.cpp + 工具函数集的运行结果应一致，每人的公共函数集需要验证至少 5 人的 90-b3.cpp，将验证名单放在 90-b3.cpp 源程序的第 2 行用注释说明即可（如果查验不正确则要连环扣分）

【注意：】90-b3.cpp 源程序可以提供给别人，工具函数集不可以

4、main 函数中的测试用例要涵盖所有函数的使用，包括各种错误情况

【大作业通过公共函数集读配置文件：】

1、90-b0、90-b1、90-b2 三个大作业均要求可以通过公共函数集读取配置文件

- 90-b0/90-b2 要求用 C++方式的公共函数集
- 90-b1 要求用 C 方式的公共函数集

2、具体使用方法要求（以 90-b2 项目生成了 90-b2.exe 为例）

- 在集成环境中运行，遵循原方式，用菜单供选择
- 双击生成的 90-b2.exe 文件，同集成环境运行的要求
- 在 cmd 窗口下输入：90-b2 -f my_tetris.cfg，则表示所有配置从 my_tetris.cfg 中读取，只运行菜单的最后一项（完整游戏版）
- 配置文件的名称可变，既允许是直接跟纯文件名（表示和 exe 在同一目录下），也可以跟带绝对/相对路径的文件名

【示例程序的提供：】

提供 90-b3-demo.cpp/90-b0.cfg/90-b1.cfg/90-b2.cfg/90-b2.exe 供参考**（因时间安排问题，稍晚几天提供）**

【作业要求：】

1、 仅需要在VS2017下编译通过即可，要做到“0 errors, 0 warnings”

2、 **5月1日前**网上提交本次作业

3、 每题所占平时成绩的具体分值见网页

4、 超过截止时间提交作业会自动扣除相应的分数，具体见网页上的说明

5、 **如果前两次大作业完成度不理想，此次只要能补齐菜单最后一项的实现（完整版），可以将此前作业的分数做提升，最高不超过满分的85%**