

# 大作业——CLiTBot 命令行版机器人点灯

## 〇、声明

本文档旨在说明大作业基本要求。**请勿外传。**

设置大作业的目的是希望同学们活用课程学到的知识，锻炼解决实际问题的能力，体验团队协作过程，并从过程中学到更多课堂很难涉及的知识。

“程序设计基础”课程教学团队保留对本文档内容的最终解释权。

# 目录

〇、声明 .....	1
一、任务概述 .....	4
1.1 绘制模块 .....	4
1.2 执行模块 .....	4
1.3 交互模块 .....	5
二、功能任务 .....	6
2.0 公共部分 .....	6
2.0.1 数据结构定义.....	6
2.0.2 地图文件格式.....	8
2.0.3 指令序列文件格式.....	9
2.1 绘制模块任务 .....	10
2.1.1 坐标系约定 .....	10
2.1.2 机器人绘制 .....	10
2.1.3 单元格及背景绘制.....	11
2.1.4 参考接口 .....	12
2.2 执行模块任务 .....	12
2.2.1 行动指令 .....	12
2.2.2 过程调用指令.....	13
2.2.3 结束条件 .....	13
2.2.4 参考接口 .....	14
2.3 交互模块任务 .....	14
2.3.1 加载地图 .....	15
2.3.2 设置自动保存.....	15
2.3.3 设置步数限制.....	15
2.3.4 输出配置 .....	16
2.3.5 新建指令序列.....	18
2.3.6 执行指令序列.....	18
2.3.7 结束程序 .....	19
三、提交要求及计分说明.....	20
3.1 组队要求 .....	20
3.2 提交要求 .....	20
3.3 计分说明 .....	21



# 一、任务概述

机器人点灯 (LightBot) 是一款移动端的闯关小游戏。你的任务是设计一系列指令，控制一个机器人点亮地图上的所有“灯”。在每一关里，地图、机器人的初始位置、灯的位置、指令数量限制都是不一样的。因此，为了完成任务，你需要仔细观察地图，充分利用有限的指令，合理利用过程调用机制，甚至使用递归调用形式的循环。这也使得游戏更具趣味性与难度。

本次大作业希望同学们分组完成一个面向命令行界面简化的机器人点灯 CLiTBot (Command-Line-Interface Tsinghua's Bot, 谐音 C Light Bot)。原本的游戏涉及到复杂的 UI 设计、动画处理等，超出了《程序设计基础》课程的核心知识和能力范围。因此，本次大作业对其进行了简化和调整，希望同学们完成关键逻辑设计和简单的图形呈现。同学们**应组成三人小组完成**。整体任务分成 3 个模块，推荐每位同学负责一个模块，各个模块分别完成并通过测试后，再整合为最终的大作业成果。

## 1.1 绘制模块

绘制模块负责将机器人和地图的状态，使用 2.5D 视角进行绘制，并保存为 24 位色 .bmp 图像文件。详见 [2.1 绘制模块任务](#) 说明。

## 1.2 执行模块

执行模块负责执行用户给出的机器人指令序列，正确修改机器人

和地图状态，期间可能需要调用绘制模块进行输出。详见 [2.2 执行模块任务](#)说明。

### 1.3 交互模块

交互模块负责与用户的交互操作，通过命令行形式接受用户操作命令，完成对应操作，并通过命令行形式显示结果。可能需要通过文件进行输入输出，需要调用执行模块的功能。详见 [2.3 交互模块任务](#)说明。

各模块特点及工作量评估如下，谨供小组内分工参考。实际工作量与个人水平及风格相关，如与实际工作量不符，课程团队概不负责。

模块名	逻辑设计	踩坑指数	代码量
绘制模块	☆☆☆	☆☆	☆☆☆☆☆
执行模块	☆☆☆☆	☆☆☆☆	☆☆
交互模块	☆☆	☆☆☆☆	☆☆☆☆

## 二、功能任务

### 2.0 公共部分

#### 2.0.1 数据结构定义

各模块公用的全局变量定义为以下结构和变量：

<pre>// 位置类型，可用来表达机器人或灯所在位置 struct Position {     int x, y; // x 表示列号, y 表示行号 };</pre>
<pre>// 方向枚举类型，可用来表达机器人朝向，只有四种取值 enum Direction {     UP, DOWN, LEFT, RIGHT };</pre>
<pre>// 机器人类型 struct Robot {     Position pos; // 机器人位置     Direction dir; // 机器人朝向 };</pre>
<pre>// 灯类型 struct Light {     Position pos; // 灯位置     bool lighten; // 是否被点亮 };</pre>
<pre>// 单元格类型 struct Cell {     int height; // 高度     int light_id; // 灯标识, -1表示该单元格上没有灯     bool robot; // true/false分别表示机器人在/不在该单元格上 };</pre>
<pre>// 指令类型 enum OpType {     TL, TR, MOV, JMP, LIT, CALL }; // TL为左转, TR为右转, MOV为向前行走, JMP为跳跃, LIT为点亮灯; // 使用CALL表示调用MAIN, CALL + 1表示调用P1, 以此类推。</pre>
<pre>// 过程类型 struct Proc {     OpType ops[MAX_OPS]; // 指令记录, MAX_OPS为合理常数     int count; // 有效指令数 };</pre>

```

// 指令序列类型
struct OpSeq {
    // 过程记录, MAX_PROCS为合理常数
    // procs[0]为MAIN过程, procs[1]为P1过程, 以此类推
    Proc procs[MAX_PROCS];
    int count; // 有效过程数
};

// 地图状态类型
struct Map {
    // 单元格组成二维数组, MAX_ROW、MAX_COL为合理常数
    Cell cells[MAX_ROW][MAX_COL];
    int row, col; // 有效行数、有效列数
    // 灯记录, MAX_LIT为合理常数
    Light lights[MAX_LIT];
    int num_lights; // 有效灯数
    // 地图上同时只有一个机器人
    Robot robot;
    // 每个过程的指令数限制
    int op_limit[MAX_PROCS];
};

// 游戏状态类型
struct Game {
    char map_name[MAX_PATH_LEN]; // 当前地图的文件路径名
    Map map_init; // 地图初始状态
    Map map_run; // 指令执行过程中的地图状态
    // 自动保存的文件路径名, MAX_PATH_LEN为合理常数
    char save_path[MAX_PATH_LEN];
    int auto_save_id; // 自动保存标识
    int limit; // 执行指令上限 (用来避免无限递归)
};
Game game; // 全局唯一的Game变量

```

上述结构和变量定义仅供参考，直接复制可能遇到的编译问题，课程团队不负责解释。上述结构和变量的定义详细意义见各模块功能描述。可以根据小组情况对上述结构和变量的定义进行修改，但应注意，其中的一些数据结构是各模块实现过程中均须使用的，请在小组充分协商、取得共识的情况下进行修改。由于数据结构不统一导致三个模块的程序无法整合，由小组共同负责，**不允许小组三人分别提**

交一份不能整合的代码。

### 2.0.2 地图文件格式

地图保存为一个文本文件。一幅地图实际上是一个关卡，它包含地图信息与指令序列的限制，各行内容如下：

第一行有空格隔开的四个整数 `row`、`col`、`num_lights`、`num_procs`，表示地图共有 `row` 行 `col` 列、有 `num_lights` 个灯需要点亮、最多允许 `num_procs` 个过程。

接下来 `row` 行中，每行有 `col` 个空格隔开的非负整数，正整数表示地图各行各列的高度，0 表示该单元格不存在。

接下来 `num_lights` 行中，每行有空格隔开的两个整数 `x` 和 `y`，表示各个灯的坐标。

接下来的一行里有空格隔开的 `num_procs` 个整数，表示各个过程指令数限制。其中，第一个数表示 MAIN 过程允许的指令数，后面的数依次表示 P1、P2 等过程允许的操作数。

接下来的一行里有空格隔开的三个整数 `x`、`y`、`d`，其中 `x` 和 `y` 表示机器人初始位置的坐标，`d` 表示朝向。朝向用 0、1、2、3 分别表示机器人的初始朝向为上、下、左、右。

地图文件由小组成员使用其他文本编辑工具生成，大作业程序可以假设地图文件为合法的输入，不必考虑异常情况。

例如，某个关卡的地图文件内容及对应的地图如下，根据这一地图绘制的 2.5D 图像如[图 1](#)。



```

3 3 3 4
2 2 3
2 0 0
1 1 1
2 0
0 1
2 2
3 4 4 4
2 0 2

```

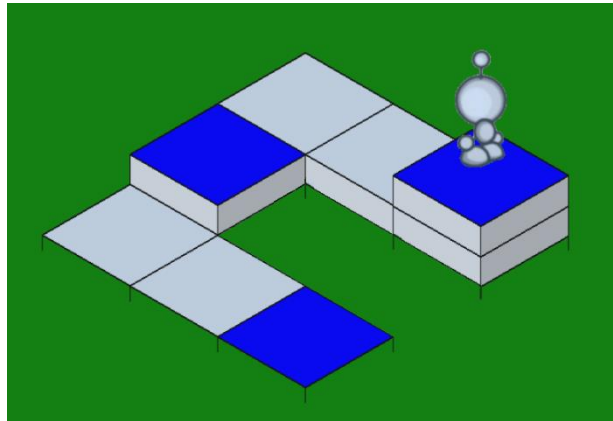


图 1、2. 5D 地图示例

### 2.0.3 指令序列文件格式

指令序列保存为一个文本文件。第一行为一个正整数，表示后续行数。接下来的每行对应指令序列的一个过程。其中的第一行对应 MAIN 过程，第二行对应 P1 过程，第 3 行对应 P2 过程，以此类推。每行以一个整数  $n$  开始，表示该过程由  $n$  条指令组成，接下来为空格分隔的  $n$  条指令。过程可以为空，此时  $n=0$ 。

一个合理的操作序列文件如下所示，该指令序列可以点亮图 1 中的所有灯（上表面为蓝色的单元格表示有待点亮的灯）。

```

4
3 P1 P2 P3
4 LIT JMP MOV MOV
4 TL MOV LIT JMP
4 TL MOV MOV LIT

```

## 2.1 绘制模块任务

绘制模块的任务是，根据地图状态绘制形如[图 1](#)的 2.5D 地图。

### 2.1.1 坐标系约定

借[图 1](#)约定机器人的方向：图中机器面朝方向为“左”，机器人的左手方向为“下”，右手方向为“上”，后方为“右”。

图中最上方的单元格为(0, 0)点，沿机器人背后方向为 x 轴正方向，沿机器人左手方向为 y 轴正方向。同一行的单元格，x 变化、y 不变；同一列的单元格，x 不变、y 变化。

### 2.1.2 机器人绘制

大作业将提供一张四个视角下的机器人图像（robot.bmp，24 位色.bmp 图像）作为素材供使用，如图 2 所示（仅供预览，请使用作业附件中的原图）。

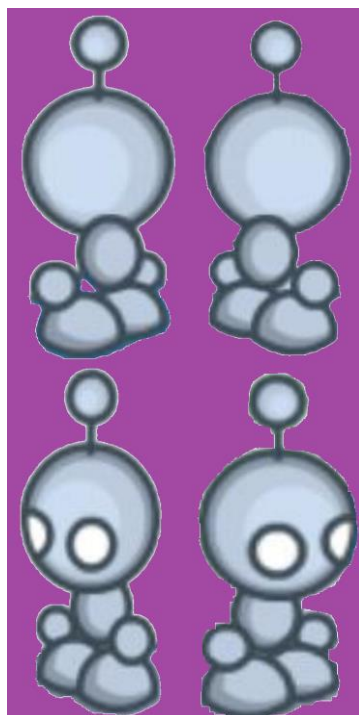


图 2、机器人素材预览

由于简单的.bmp 图不支持透明，这里采用一种比较老式的方法表达透明，即，用一个素材中不会出现的特定颜色表示透明。这里约定使用紫色（红绿蓝分别取值：164、73、163）表示透明。

素材图尺寸为  $700 \times 1400$ ，分别放置了机器人在四个朝向时的图像。使用时，需要将它拆分为  $350 \times 700$  的四张图，并根据需要进行缩放后，“贴”到合适的位置。

素材是助教自行从 LightBot 游戏中截图、抠图获得的。可以预先约定尺寸大小，自行用“画图”等软件将原图进行缩放，从而在程序中不必处理缩放问题。可以自行收集其他喜欢的素材，但必须满足 2.5D 的视角要求并正确体现出机器人的四个朝向。

### 2.1.3 单元格及背景绘制

每一个单元格为一个长方体，而且俯视图为正方形。在 2.5D 视角下这些单元格的上表面投影成了一个菱形，可以认为这些菱形都是两个等边三角形组成的。绘制时，单元格的下表面、上侧面和左侧面被单元格自身遮挡，只需绘制上表面、下侧面和右侧面，注意为这三个面选择有一定差异的颜色以体现空间感。

整体配色方案（包括背景、线条、单元格各面、待点亮的灯、已点亮的灯等）不做限制，可以参考原作，可以采用自己喜欢的方案，也可以直接参考[图 1](#)中的配色。

注意，如果线条只绘制一个像素宽，视觉效果可能很差，请尝试绘制有一定宽度的线条。另外，如果发现绘制的线条有很明显的锯齿感，可以选择尝试自学抗锯齿相关知识，能够让线条看上去更平滑。

### 2.1.4 参考接口

建议实现如下接口：

```
void save(const char * path);  
void auto_save();
```

在 `save()` 函数中，利用全局变量 `game.map_run` 获取到当前地图状态并进行绘制，将结果保存在参数 `path` 所指定的文件中。

在 `auto_save()` 函数中，首先检查全局变量 `game.save_path` 是否为空字符串，来确定是否要自动保存。如果需要，则生成正确的保存文件路径名，并调用 `save()` 函数。自动保存时的文件路径名由全局变量 `game.save_path` 和 `game.auto_save_id` 拼接而成。拼接时，将全局变量 `game.save_path` 字符串中形如“%nd”的部分替换为 `n` 位十进制数，其数值为 `game.auto_save_id`，数值不足 `n` 位时补 0，数值超过 `n` 位时按数值存储。例如，`game.save_path` 为“`imgs/%4d.bmp`”时，则多次调用 `auto_save()` 函数会将地图状态依次保存为“`imgs/0000.bmp`”、“`imgs/0001.bmp`”、“`imgs/0002.bmp`”等，以此类推。

## 2.2 执行模块任务

执行模块的任务是，执行指定的指令序列文件中的指令序列。指令序列文件的格式，详见 [2.0.3 指令序列文件格式](#)。

### 2.2.1 行动指令

机器人共有 5 种行动指令：

- 1、TL，原地左转 90° ；
- 2、TR，原地右转 90° ；

3、MOV，向前移动一格，前方单元格的高度必须与当前单元格的高度相等，否则原地不动并在命令行输出警告；

4、JMP，向前跳跃一格，前方单元格的高度必须比当前单元格的高度大 1 或小 1，否则原地不动并在命令行输出警告；

5、LIT，点亮当前位置的灯，如果当前位置没有灯、或者灯已被点亮，则不行动并在命令行输出警告。

每执行一条上述指令（不论是否成功），都记一次行动步数。

### 2.2.2 过程调用指令

除上述指令外，过程名称可以作为指令，意义为调用该过程。过程名称只能是 MAIN 或者  $P_n$  ( $n \geq 1$ ，例如 P1、P2 等) 的形式。MAIN 过程即为执行的入口（它也可以被调用）。如果过程不存在或过程中没有指令（可以视为同一种情况），则视为调用直接返回，但应在命令行输出警告。每次过程调用（不论是否成功），都记一次行动步数。

调用过程结束后，应返回调用处。为实现这一功能，应维护调用栈。

### 2.2.3 结束条件

机器人操作序列开始执行后，当且仅当以下三种情况下会结束执行：

1、地图中的所有灯都被点亮；

2、不满足 1，但步数已达到上限（即，全局变量 `game.limit` 的值）；

3、不满足 1 和 2，但 MAIN 过程的全部指令执行完毕。

#### 2.2.4 参考接口

建议实现如下结构与接口：

```
// 执行结果枚举类型
enum ResultType {
    LIGHT, // 结束条件1, 点亮了全部灯, 干得漂亮
    LIMIT, // 结束条件2, 到达操作数上限
    DARK   // 结束条件3, MAIN过程执行完毕
};
// 执行结果类型
struct Result {
    int steps; // 记录总步数
    ResultType result; // 用enum记录结束原因
};
Result robot_run(const char * path);
```

在 `robot_run()` 函数中，参数 `path` 为指令序列文件的文件路径名。该函数应读取 `path` 指定的指令序列文件，并执行其中的指令序列。在执行指令序列时，首先应该将地图的初始状态（即，全局变量 `game.map_init`）复制一份作为地图的当前执行状态（即，全局变量 `game.map_run`）。此时，应重置自动保存标识，并触发一次自动保存（即，调用绘制模块提供的 `auto_save()` 函数）。在执行过程中，指令执行的效果会修改地图的当前执行状态（即，修改 `game.map_run`）。每次执行一条记步数的指令后，触发一次自动保存。

### 2.3 交互模块任务

交互模块的任务是，接受并执行用户的交互命令。交互命令由命令行输入（即，通过 `cin` 输入）。交互命令共包括 7 种：

### 2.3.1 加载地图 读取文件中的数据到程序中

命令格式: **LOAD <MAP\_PATH>**

该命令尝试加载指定的地图文件。其中<MAP\_PATH>为地图文件的文件路径名。

在程序启动时，游戏未加载任何地图（即，全局变量 `game.map_name` 为空字符串）；如加载地图成功，则全局变量 `game.map_name` 应变为<MAP\_PATH>，地图初始状态（即，全局变量 `game.map_init`）应与地图文件内容一致；如加载地图失败，则游戏恢复为未加载任何地图的状态。

例如，“**LOAD maps/main.map**”命令尝试加载 `maps/main.map` 的地图文件。

### 2.3.2 设置自动保存

命令格式: **AUTOSAVE {<SAVE\_PATH>|OFF}**

该命令设置自动保存路径，或关闭自动保存。其中<SAVE\_PATH>为自动保存的文件路径名，参见 [2.1.4 参考接口](#)。

例如，“**AUTOSAVE OFF**”关闭自动保存。

再如，“**AUTOSAVE imgs/%4d.bmp**”设置开启自动保存功能，并将地图状态自动保存为“`imgs/0000.bmp`”、“`imgs/0001.bmp`”、“`imgs/0002.bmp`”等。

### 2.3.3 设置步数限制

命令格式: **LIMIT <OPS>**

该命令设置机器人执行指令序列时的步数上限（即，全局变量

game.limit 的值)。其中<OPS>应当是一个合理的正整数。这一设置主要用于防止机器人执行指令序列时出现死循环的情况。安全起见，程序启动时，应将这一数值设置为 100。

例如，“**LIMIT 500**”将设置步数上限修改为 500。

### 2.3.4 输出配置

命令格式：**STATUS**

该命令输出当前配置，如果已成功加载地图，则显示当前地图。

样例输出如下（格式可调，但信息应完整）：

```
Map Name: maps/main.map
Autosave: imgs/%4d.bmp
Step Limit: 500
  3
  2
  1
Robot is facing left.
Proc Limit: [3, 4, 4, 4]
```

在这一样例中：第 1 行显示当前加载的地图名；第 2 行显示自动保存的文件路径名；第 3 行显示步数限制；第 4-6 行显示当前地图；第 7 行显示机器人朝向；第 8 行显示各过程的指令数限制。

输出的当前地图与 [2.0.2 地图文件格式](#) 的样例一致。数字表示单元格高度（注意高度为 0 表示单元格不存在，跳过不显示）。存在的单元格上有待点亮的灯时，应以背景蓝色输出；有已点亮的灯时，应以背景黄色输出；否则以背景灰色输出。机器人所在的单元格以数字红色输出，否则以数字绿色输出。

没有灯：背景灰色  
有灯未点亮：背景蓝色  
有灯点亮：背景黄色  
有机器人：数字红色  
无机器人：数字绿色

在命令行输出（即，cout 输出）时，可以通过 ESC 转义字符串实现调整背景色和字符颜色，其格式为：“\e[{a}m”。其中，{a}在使用



时替换为不同的样式代码。ESC 转义字符串还可以包含任意多项的样式代码，如 “\e[{a};{b};{c}m” 为三项的情况。特别注意 m 是一个紧跟在最后一个代码后的终止符，不能忽略。

样式代码一般是一个数字串，某些数字串被定义了对应的含义。本作业中用到的样式代码及其含义包括：

样式代码	含义
0	全部样式恢复默认
1	加粗字体
91	亮红色前景（机器人所在格子的高度）
92	亮绿色前景（普通格子的高度）
100	亮灰色背景（普通格子的背景）
103	亮黄色背景（已点亮的灯的背景）
104	亮蓝色背景（待点亮的灯的背景）

通过对上述样式代码的组合，可以实现不同的单元格的格式要求。例如，“\e[91;104;1m” 表示前景亮红色、背景亮蓝色且字体加粗，用于表示有机器人和待点亮的灯的单元格。

可以先尝试下面的样例代码，应能够获得上面的样例输出中的地图状态。

```
#include <iostream>
using namespace std;
int main() {
    cout << "\e[92;100;1m2\e[92;100;1m2\e[91;104;1m3\e[0m " << endl;
    cout << "\e[92;104;1m2\e[0m" << endl;
    cout << "\e[92;100;1m1\e[92;100;1m1\e[92;104;1m1\e[0m" << endl;
    return 0;
}
```

如果在 Windows 环境下看不到上述效果，可尝试在终端中输入如下指令修改注册表，打开 ANSI 颜色输出的支持。

```
reg add HKEY_CURRENT_USER\Console /v VirtualTerminalLevel /t REG_DWORD /d 0x00000001 /f
```

更多信息可参考：[https://en.wikipedia.org/wiki/ANSI\\_escape\\_code](https://en.wikipedia.org/wiki/ANSI_escape_code)。

### 2.3.5 新建指令序列

命令格式: **OP <OP\_PATH>**

该命令开启编辑指令序列模式，其中<OP\_PATH>为指令序列文件的文件路径名。如该文件已存在，则覆盖该文件，否则创建该文件。执行该命令后，**输入指令序列**，格式与 [2.0.3 指令序列文件格式](#) 中描述的指令序列文件格式一致。如果输入的指令序列中使用的过程数或任一过程的指令数超出了地图指定的上限，则提示错误。

例如，下列命令则**将指令序列存入 ops/1.ops 文件中**。

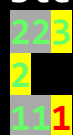
```
OP ops/1.ops
4
3 P1 P2 P3
4 LIT JMP MOV MOV
4 TL MOV LIT JMP
4 TL MOV MOV LIT
```

### 2.3.6 执行指令序列

命令格式: **RUN <OP\_PATH>**

该命令表示机器人开始按照指令序列执行，其中<OP\_PATH>为指令序列文件的文件路径名。如果文件不存在，则给出错误信息。文件存在时，**执行指令序列**（即，调用执行模块提供的 robot\_run() 函数），**并输出执行结果**（包括结束原因、总步数、结束时的地图状态）。

例如，“**RUN ops/1.ops**”表示尝试执行文件 ops/1.ops 中的指令序列。样例输出如下：

```
Run ops/1.ops, result: LIGHT
Step(s) used: 15

Robot is facing right.
```

### 2.3.7 结束程序

命令格式：**EXIT**

该命令表示用户要求结束程序。

## 三、提交要求及计分说明

### 3.1 组队要求

每位同学独立完成自己所应负责的模块部分，之后同组员合作进行整合。

**要求 3 人一组。**如出现人数不足的情况，可由同学自愿报名参与两组的大作业组队完成部分，并酌情考虑予以加分。分组方案确定后，**每组应选出一名队长。**

### 3.2 提交要求

每组**由队长**在网络学堂提交一个.zip 压缩包。

组员可以(但不必须)在网络学堂提交作业时说明队长姓名学号，**但组员不要提交附件。**发现组员在网络学堂提交附件的，将单独扣分。

压缩包内应包括但不限于：

1) 程序源文件。放置在 src 文件夹下，可以有多级目录，应包括全部工程编译所需文件以及图片、地图等资源文件

2) 说明文档。一个.doc 或.pdf 文档，放置在压缩包根目录下，内容应包括但不限于：小组人员（姓名、学号、班级）、基本功能完成情况、扩展功能说明（可选）、分工情况（注意：这部分将作为小组内同学评分依据）。

3) 演示视频。放置在压缩包根目录下，内容应包括但不限于：小组人员展示、基本功能演示、扩展功能演示（可选）。要求长度不超过 3 分钟，大小不超过 30MB，推荐使用.mp4 格式。

如果上述内容过大，导致.zip压缩包无法上传至网络学堂，可将上述内容上传至清华云盘分享，并提交一个内含有效分享链接的.txt文件代替压缩包的提交。

如果实现了扩展功能，则必须在文档中说明，并在视频中演示，否则视为没有实现扩展功能。

### 3.3 计分说明

1、要求主体使用 C/C++ 实现。满分 100 分，完整实现第二章规定的全部任务就可以获得 100% 的大作业分值。同小组内的同学获得的小组成果评分相同，个人模块得分与具体分工有关。

2、允许调用 C/C++ 语言提供的库以外的其他库，允许调用其他语言编写的函数。但如这样做，应在说明文档中给出说明，并扣除由此节省的工作量所对等的分值。特别的，如使用操作系统相关的库和头文件（如 system.h、windows.h、unistd.h 等）会进行扣分处理。C++ 标准库包含的功能可参考：<https://en.cppreference.com/w/cpp/header>。

3、实现扩展功能或扩展任务（即，除第二章中所提及任务以外的其他有意义的扩展功能）有额外加分，分值视重要程度及工作量而定。如果为实现某些扩展功能不得不使用操作系统相关库，推荐使用 Linux 系统配套库，非常不推荐使用 Windows 系统配套的库。

4、请注意程序内存的合理使用。过大的内存消耗、内存泄漏等可能会导致助教批改作业时电脑崩溃。助教保留此情况下进行额外扣分的权利。

5、“**抑制内卷条款**”：实现扩展功能或扩展任务并不能获得超过100%的大作业分值，但可以用来补足由于第2条规定导致的扣分。请各位同学根据兴趣、能力和时间安排，选择是否实现扩展功能或扩展任务。