作业九
田永铭-221900180

概念题 ：

1. 请简述什么是事件驱动的程序设计?

Windows 应用程序采用的是一种基于事件（消息）驱动的交互式流程控制结构:
程序的任何一个动作都是由某个事件激发的。
事件可以是用户的键盘、鼠标、菜单等操作。
每个事件都会向应用程序发送一些消息。
每个应用程序都有一个消息队列
“取消息-处理消息”的过程称为消息循环。
每个窗口都有一个消息处理函数。
主程序:

```
//初始化
.....
//进入消息循环
while (取消息)
{  .....
    处理消息
}
```

2. 请简述什么是“文档-视”结构，它有什么作用?

视类（CView）

- 实现程序数据的显示功能以及操作数据时与用户的交互功能。
- 视窗口（简称视）通常位于单文档应用主窗口（CFrameWnd）和多文档应用子窗口（CMDIChildWnd）的客户区（可显示区）。

文档类（CDocument）

- 对程序要处理的数据进行管理，包括磁盘文件输入/输出。

“文档-视”结构将这两类结合。

作用:
“文档-视”结构将文档注释和代码视图结合起来，使得开发者可以在代码中直接查看和编辑文档注释，并且可以通过代码视图快速浏览和定位到各种元素的定义和实现。这样可以提高代码的可读性和可维护性，同时也方便生成 API 文档和开发者文档。

3. 请简述什么是图形用户接口（GUI）？与基于控制台的接口相比它有什么优点?

图形用户接口（GUI）是一种通过图形元素（如窗口、按钮、文本框、滚动条等）和鼠标、键盘等输入设备进行交互的用户界面。

优点:

用户友好：GUI 提供了直观、易于理解和使用的界面，使用户能够更快地完成任务。

可视化：GUI 提供了可视化的反馈，例如通过窗口、图标和菜单等元素来展示程序状态和操作结果。

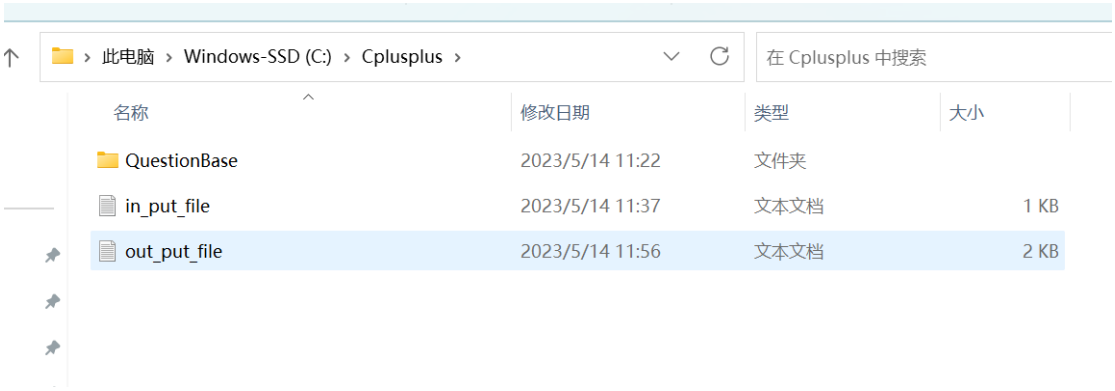
多任务处理：GUI 允许用户同时执行多个任务，例如在一个窗口中打开多个文档，或在一个应用程序中同时运行多个工具。

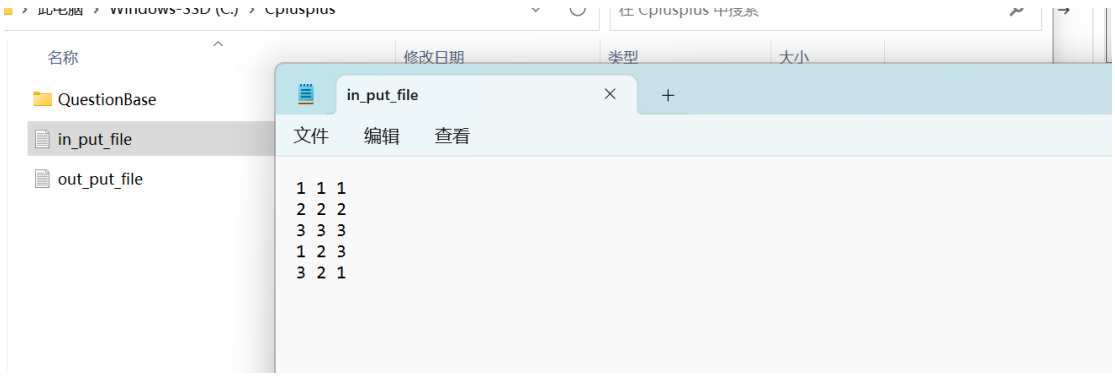
可定制性：GUI 允许用户自定义界面，例如可以改变窗口大小、颜色和字体等。

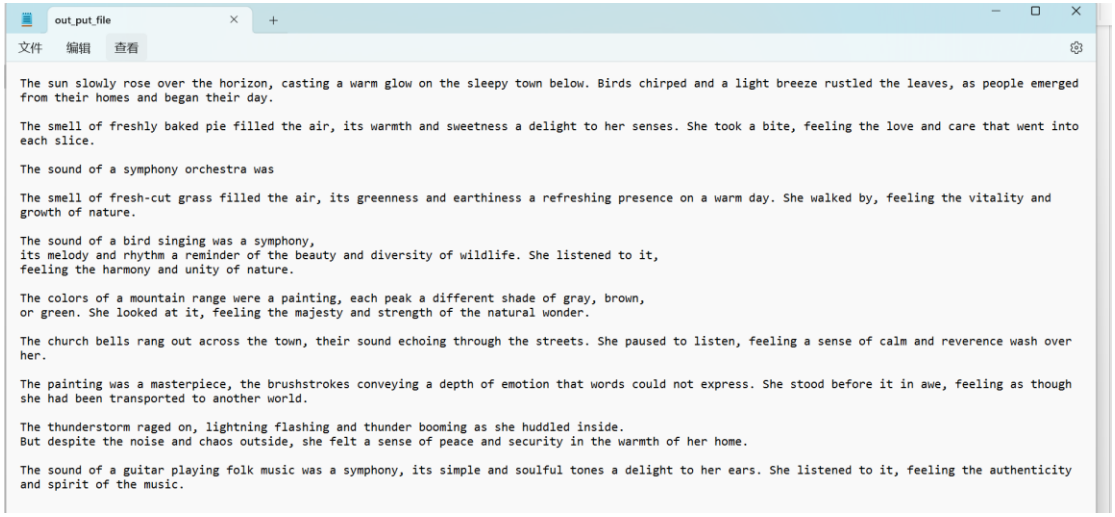
可扩展性：GUI 允许开发者添加新的界面元素和功能，例如通过插件或扩展来增强程序功能。

编程题

1. 试卷生成：







代码实现：

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <string>
using namespace std;

//题目内容
struct Question
{
    string type;
    string level;
    string content;
};

//抽取要求
struct Requirement
{
    string type;
    string level;
    int count;
};

// 获取题库中所有题目
vector<Question> get_questions(const string& question_base,const
Requirement& req)
{
    vector<Question> questions;
    string path = question_base + "/type" + req.type + "/level" +
req.level + "/questions.txt";
    ifstream fin(path);
    if(!fin) exit(-1);
    else
    {
        string line;
        Question question;
        bool reading = false;
        while (getline(fin, line))
        {
            if (line == "[begin]")
            {
                question.type = req.type;
                question.level = req.level;
```

```
                question.content.clear();
                 reading = true;
            }
            else if (line == "[end]")
            {
                reading = false;
                questions.push_back(question);
            }
             if (reading) question.content += (line + '\n');
        }
    }

    fin.close();
    return questions;
}

// 读取要求
vector<Requirement> read_requirements(const string& input_request)
{
    vector<Requirement> requirements;
    ifstream fin(input_request);
    if(!fin) exit(-1);
    else
    {
        string line;
        while(getline(fin, line))
        {
            istringstream iss(line);
            Requirement req;
            iss >> req.type >> req.level >> req.count;
            requirements.push_back(req);
        }
    }
    fin.close();
    return requirements;
}

//generate 函数
void generate(const char * input_request, const char * question_base,
const char * output_file)
{
    vector<Requirement> requirements = read_requirements(input_request);
    ofstream fout(output_file);
    vector<Question> questions;
```

```
    if(!fout) exit(-1);
    else
    {
        for(int i = 0;i < requirements.size(); i++)
        {
            questions = get_questions(question_base,requirements[i]);
            for (int j = 0; j < questions.size(); j++)
            {
                fout << questions[j].content<< endl;
            }
            questions.clear();
        }
    }
}

int main()
{
    char input_request[] = "C:\\Cplusplus\\in_put_file.txt";
    char question_base[] = "C:\\Cplusplus\\QuestionBase";
    char output_file[] = "C:\\Cplusplus\\out_put_file.txt";
    generate(input_request,question_base,output_file);
}
```