数据结构 2022秋 Lab2

吴逸昕 19302010013　张皓捷 19302010021

说明

- 若无特殊情况，请使用 C++ 完成本次实验。
- 将基于测试用例通过情况评分；若未按要求完成，视具体情况将该部分得分乘以 0~0.5 计分。请勿修改函数接口、类名、文件名等，若因此导致测试用例无法通过或编译失败，后果自负。
- **截止日期 2022年10月7日 23:59:59，只需提交** `MyStack.cpp` `MyQueue.cpp` `Postfix.cpp` 共 3 个文件，将代码打包为 zip，命名为 学号_姓名_Lab2.zip，提交于 elearning。

任务

1. 双向链表实现栈

基于双向链表，实现栈类 `MyStack.cpp` ，并通过 `MyStackTest.cpp` 测试。请勿创建其他属性或函数，或是修改函数接口、类名、文件名。以下为缩略版，详见代码文件。

```
template<typename T> class MyStack {
private:
    const unsigned int CAPACITY;
    int curSize{};
    Node<T>* top;

public:
    explicit MyStack(int capacity) : CAPACITY(capacity) {
        this->curSize = 0;
        this->top = nullptr;
    }

    bool push(Node<T>* node);

    Node<T>* pop();

    Node<T>* peek();

    int getSize();

    bool isEmpty();

    bool isFull();
};
```

2. 数组实现循环队列

基于数组，实现循环队列类 `MyQueue.cpp` ，并通过 `MyQueueTest.cpp` 测试。请勿创建其他属性或函数，或是修改函数接口、类名、文件名。以下为缩略版，详见代码文件。

```
template<typename T> class MyQueue {
private:
```

```
    const unsigned int CAPACITY;
    int front{};
    int rear{};
    bool full{};
    T* data;

public:
    explicit MyQueue(int capacity) : CAPACITY(capacity) {
        this->front = 0;
        this->rear = 0;
        this->full = (capacity == 0);
        this->data = new T[capacity];
    }

    bool enqueue(T item);

    T dequeue();

    T getHead();

    int getSize();

    bool isEmpty();

    bool isFull();
};
```

3. 应用：中缀表达式转后缀表达式

使用你实现的栈或队列，在 `Postfix.cpp` 实现一个中缀表达式到后缀表达式的转换器，并通过 `PostfixTest.cpp` 。仅要求实现 `*` `/` `+` `-` 操作，不要求实现括号操作。请勿修改函数接口、类名、文件名。

```
static string postfix(string infix) {
    // TODO
    return "";
}
```

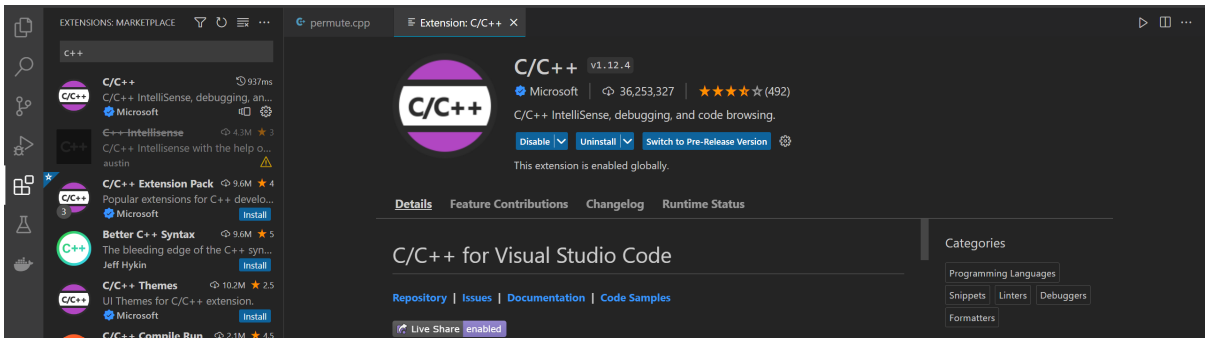
调试器 Debugger 的使用

在编写比较复杂的代码时，常常难以一次性求出完全正确的解答。此时，可以借助调试器 Debugger 来协助发现代码中存在的问题，而无需自己手动逐步计算。

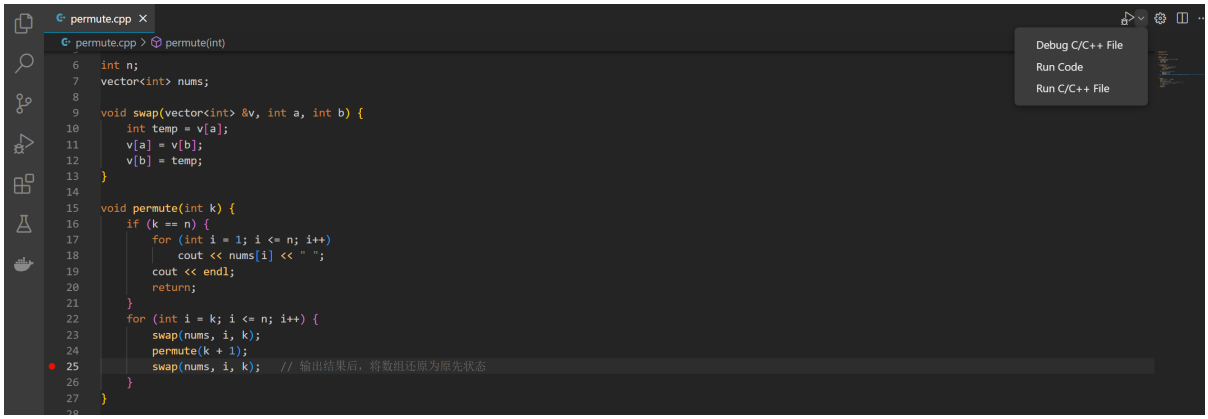
以下以 VScode 与 CLion 为例，简要介绍如何使用调试器调试 C++ 代码。若你使用的是其他的集成环境，也可以参考此教程或是在网上找到相应的教程。

1. VScode

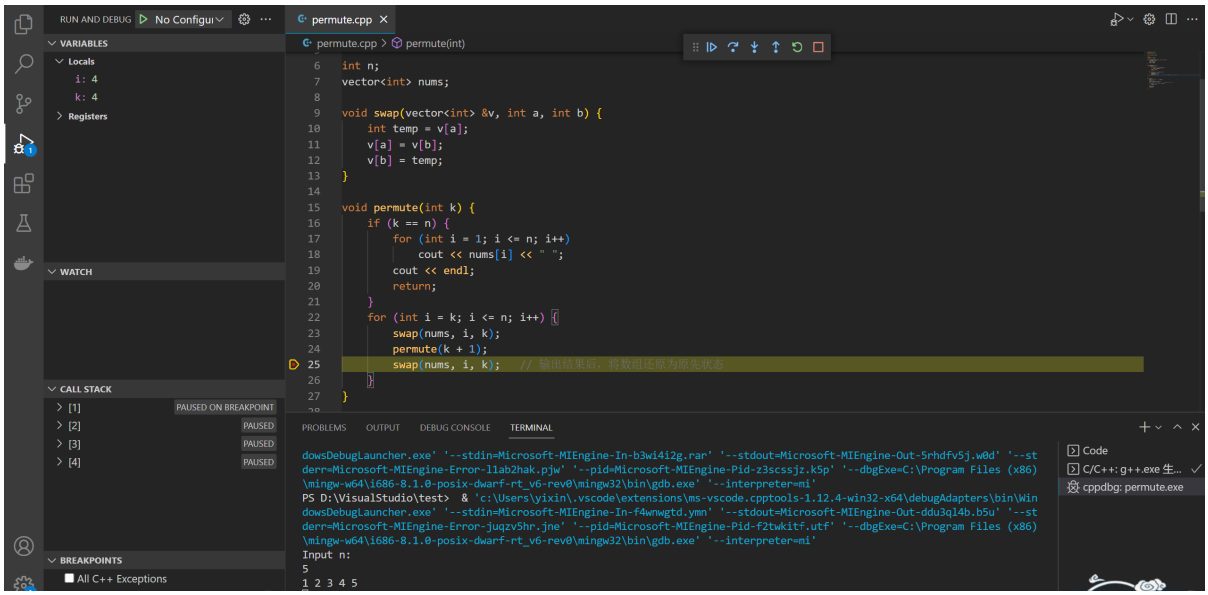
首先在 Extensions 中下载 C/C++（如果你还未下载的话），否则将无法打断点或进行调试。



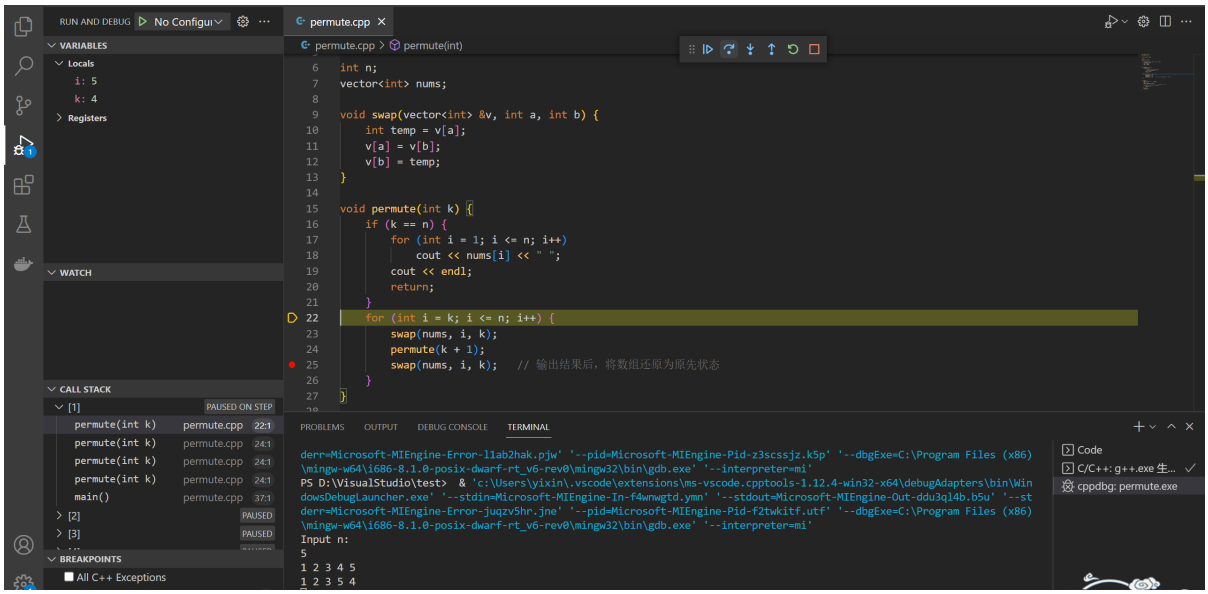
在需要调试的代码行左侧点击，打一个断点（如第25行），并在右上角点击 Debug C/C++ File 开始调试。



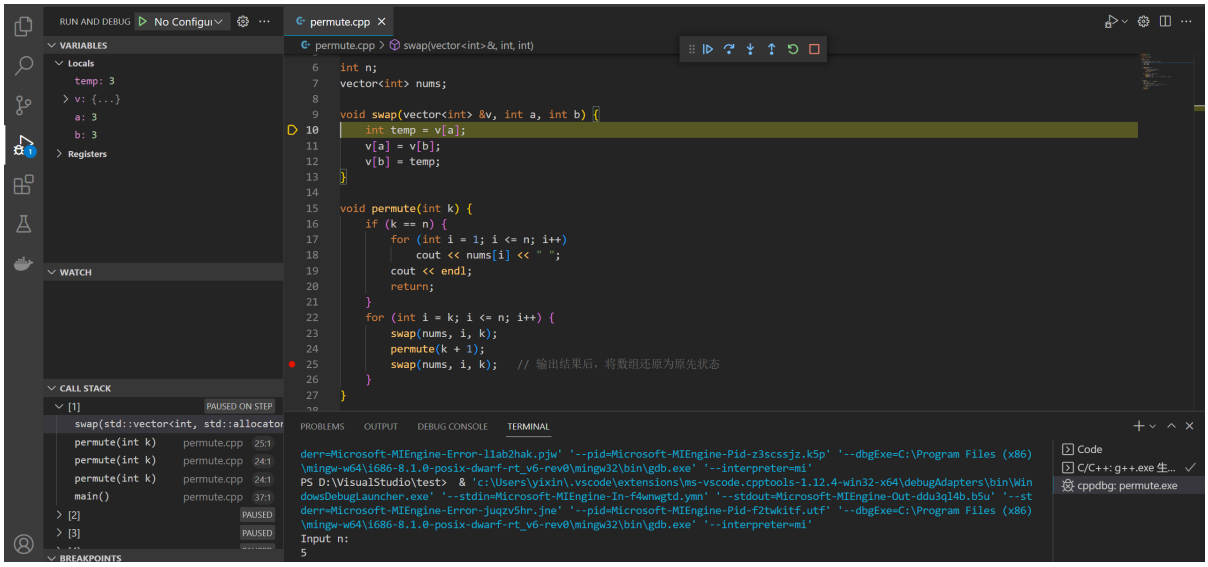
可以看到屏幕上方多出了一行按钮，从左到右分别是 `Continue` `Step Over` `Step Into` `Step Out` `Restart` 与 `Stop` 。屏幕左侧可以看到当前各变量或对象的值。



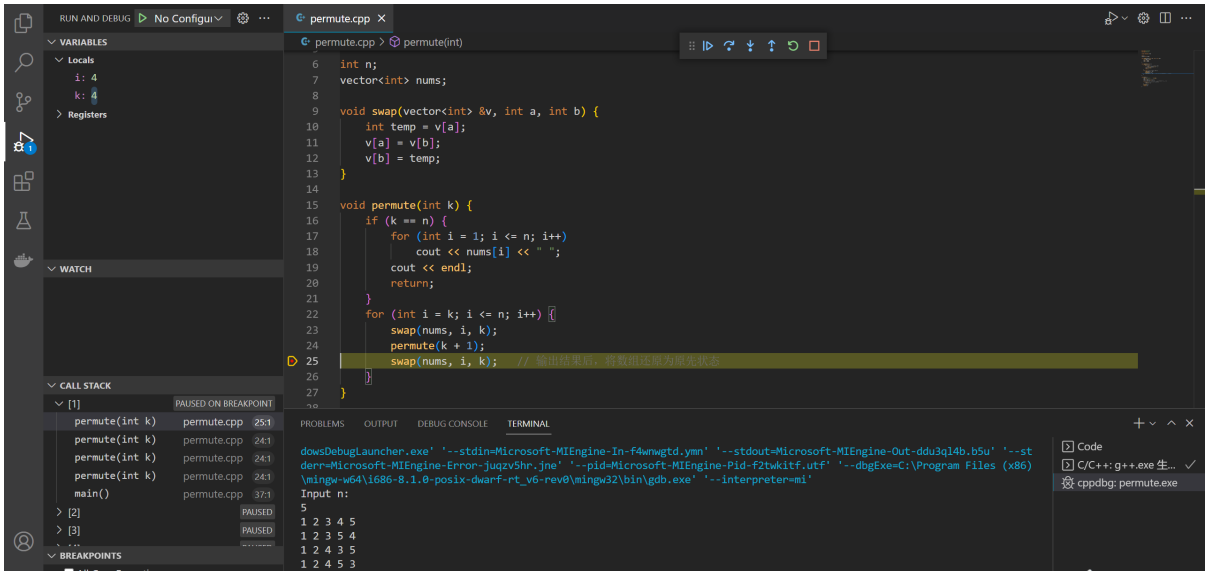
一般最常使用 `Step Over` ，逐行运行代码。



可以使用 `Step Into` 从第 23 行进入调用的这个 `swap` 函数。可使用 `Step Out` 返回调用者所在行。

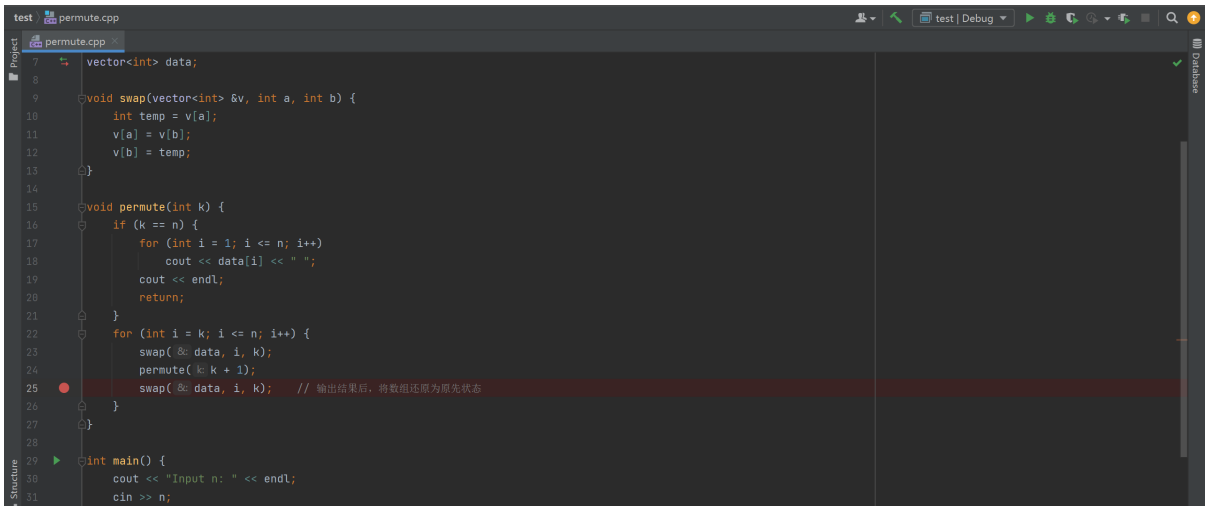


按 `Continue` 直接运行到下一个断点处或下一次执行断点所在行的代码时。当调试完成后，可以选择点击 `Restart` 从头开始调试，或是 `Stop` 退出调试模式。

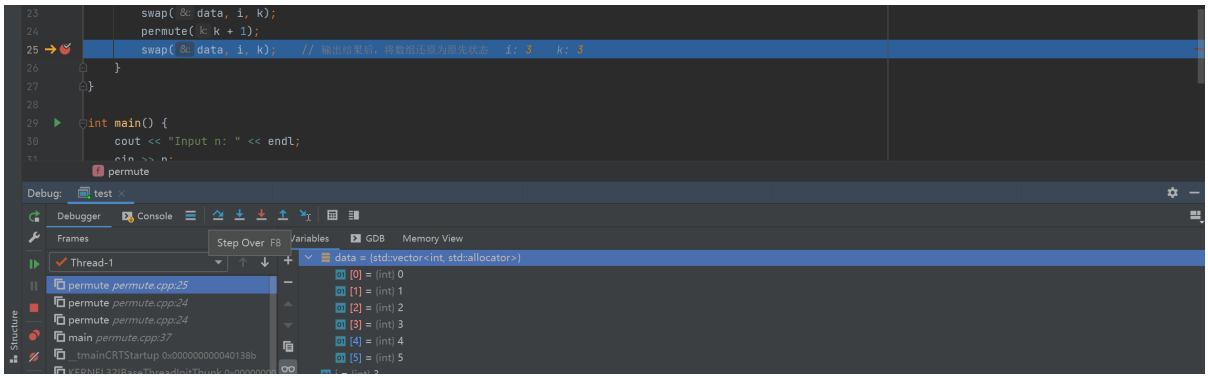


2. CLion

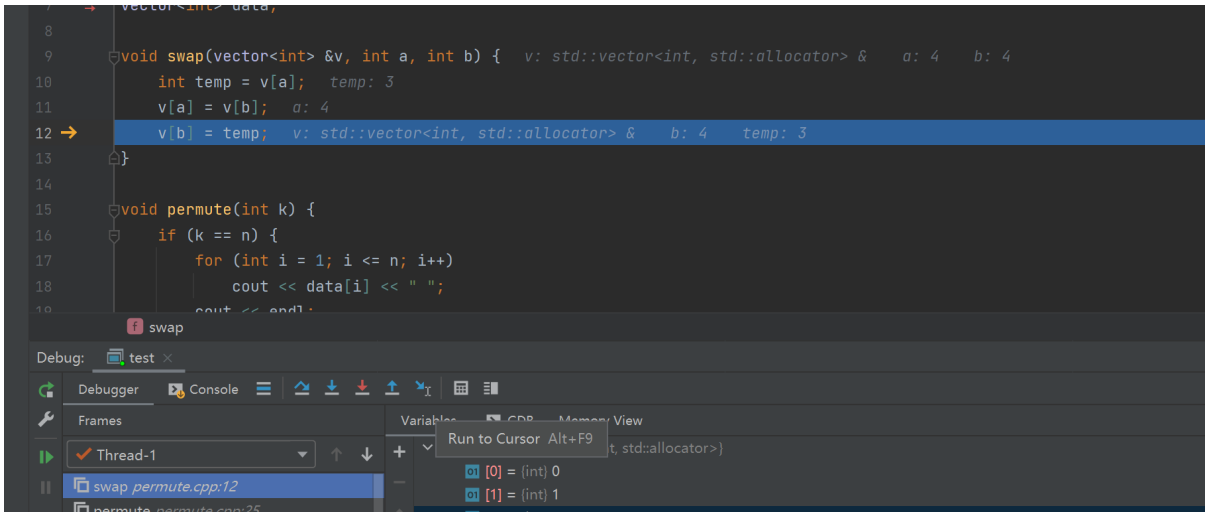
CLion 中，直接在代码行左侧打断点并点击右上角运行按钮旁的 Debug 按钮即可。



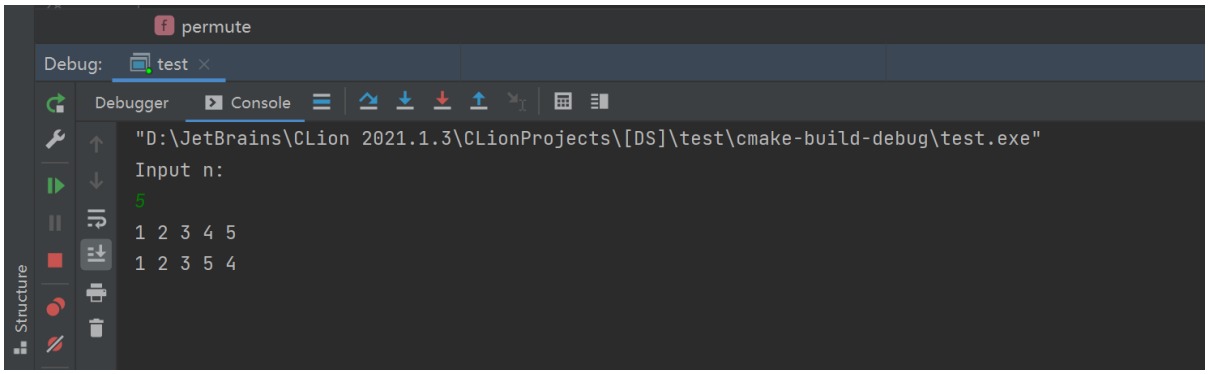
底部 `Console` 标签右侧的前三个蓝色箭头按钮分别是 `Step Over` `Step Into` `Step Out`（每个的作用见 VScode 部分）。`Debugger` 标签左侧一列两个绿色按钮分别是 `Restart` `Continue`，第一个红色按钮为 `Stop`。



最右侧的蓝色箭头 `Run to Cursor` 提供了一种十分方便的调试方式：无需打断点，光标选择哪里，点击按钮后就能直接运行到那里停下。



可以通过 `Console` 标签与 `Debugger` 标签，在调试器与控制台之间切换。


permute
Debug:
test
Debugger
Console
"D:\JetBrains\CLion 2021.1.3\CLionProjects\[DS]\test\cmake-build-debug\test.exe"
Input n:
5
1 2 3 4 5
1 2 3 5 4
Structure