|  |
| --- |
| 哈尔滨工业大学(深圳) |
| **《数据结构》实验报告** |
|  |
| 实验一  栈与队列的应用  学 院: 计算机科学与技术   |  |  | | --- | --- | | 姓 名: | 宗晴 | | 学 号: | 200110513 | | 专 业: | 计算机科学与技术 | | 日 期: | 2021-04-01 | |

# 一、问题分析

题目要求用数组实现栈的基本操作，包括入栈、出栈、获取栈顶元素、判断栈是否为空的操作；以及用栈实现先进先出队列的基本操作，包括入队、出队、获取队头元素、判断队列是否为空、获取队列的一个数组拷贝等操作。

基于栈和队列的性质，可以用顺序存储的线性表即数组来模拟实现栈（结构体中包含一个数组以及栈顶指针），用栈（两个栈，一个用来模拟入队、另一个用来模拟出队）来模拟实现先进先出的队列。

# 二、详细设计

## 2.1 设计思想

1、栈：

用数组以及一个用来记录栈顶元素下标的栈顶指针来实现栈。

1. 判断栈是否为空：由于栈在初始化时栈顶指针即栈顶元素的下标默认为-1，所以判断栈是否为空也就是判断栈顶元素下标是否为-1。若是-1，则代表栈为空，返回1；否则代表栈中有元素，返回0。
2. 向栈插入一个元素：首先判断栈是否已满，即判断栈顶元素下标是否为栈的最大容量-1（因为从0开始存储）。若栈满，则代表操作失败，返回0；否则先将栈顶指针+1，然后将待插入元素存储在该位置，此时操作成功，返回1。
3. 从栈中弹出一个元素：首先调用上述第一个函数判断栈是否为空，若栈为空，则代表操作失败，返回0；否则，先将栈顶指针所指位置的元素赋给指针e（用来带回元素），然后将栈顶指针-1，此时操作成功，返回1。
4. 获取栈顶元素（不删除该元素）：操作同第（3）点，但是最后不需要改变栈顶指针的指向。
5. 队列：

用两个栈来实现队列，栈1用来入栈，栈2用来出栈。

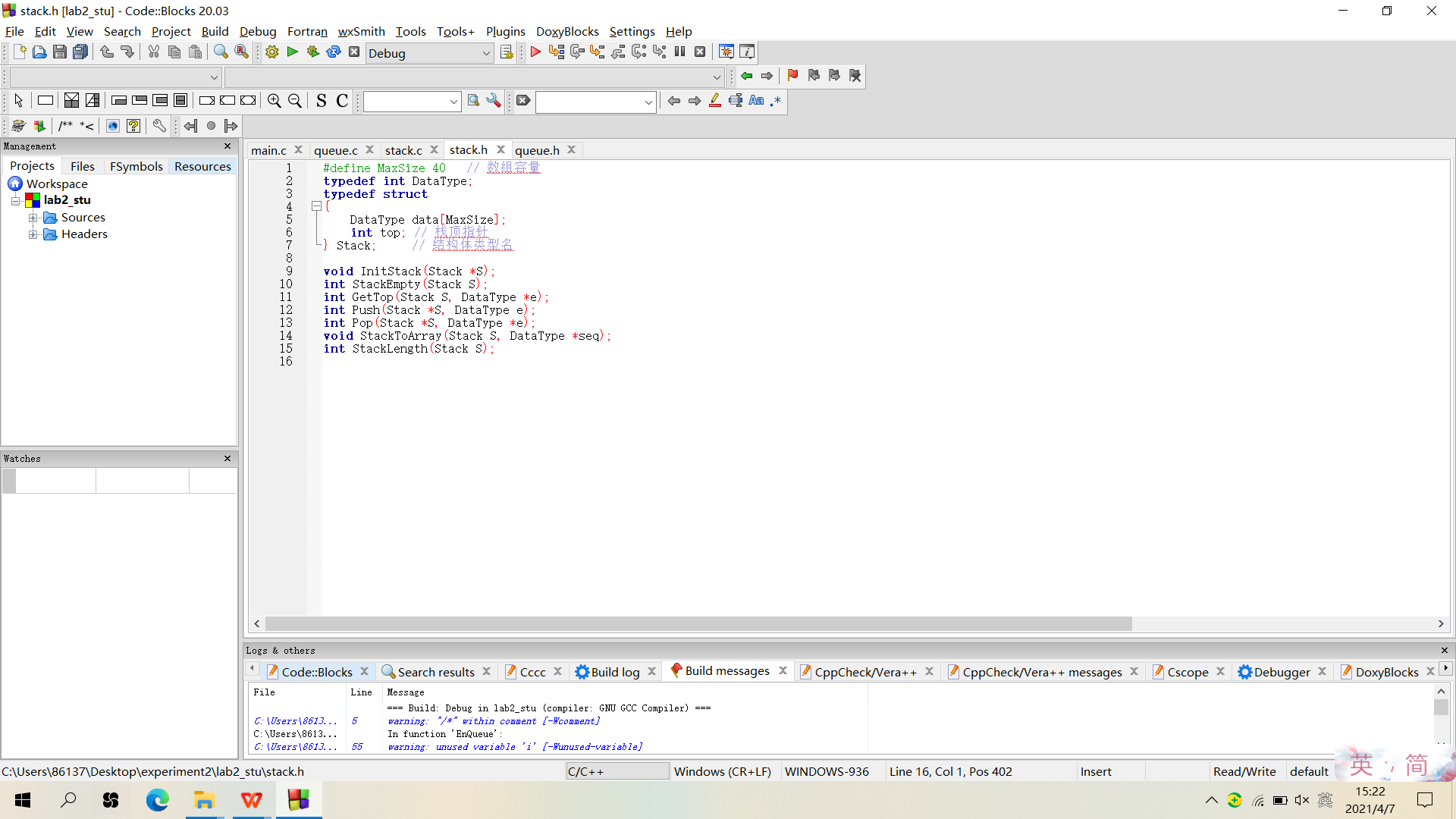
若栈1中有元素，则栈1的栈顶元素总代表队尾元素，否则栈2的栈底元素代表队尾元素；若栈2中有元素，则栈2的栈顶元素代表队头元素，否则栈1的栈底元素代表队头元素。

1. 判断队列是否为空：调用求队列已使用长度的函数求出队列长度（该函数调用了两次求栈已使用长度的函数求出队列的两个栈的总长），若长度不为0，则队列不空，返回0，否则队列为空，返回1。
2. 入队操作：调用求队列已使用长度的函数求出队列长度。若该长度等于最大存储长度，代表队列已满，无法再入队，操作失败，返回0；否则调用向栈插入一个元素的函数，向栈1中压入待入队元素，此时操作成功，返回1。
3. 出队操作：调用求判断队列是否为空的函数。若队列为空，则无法再出队，操作失败，返回0；否则，调用判断栈是否为空的函数判断栈2是否为空，若栈2为空，则意味着栈1的栈底元素是队头元素，应该将栈1中的元素用出栈和入栈函数转移到栈2中，再用出栈函数取出栈2的栈顶元素（即队头元素），若栈2不空，则直接用出栈函数取出栈2的栈顶元素（即队头元素）。
4. 获取队列头（不删除元素）：操作同第（3）点，但是最后调用获取栈顶元素（不删除该元素）的函数获取栈2的栈顶元素（即队头元素）。
5. 获取队列的一个数组拷贝，顺序为从队列头到队列尾：首先判断栈2中是否有元素，当栈2中有元素时，利用出栈函数将栈2中的元素从栈顶到栈底依次拷贝到数组中。然后调用获取栈的一个数组拷贝（顺序为从栈底到栈顶）的函数，将队列的栈1中的元素接下去拷贝到数组中。

## 2.2 存储结构及操作

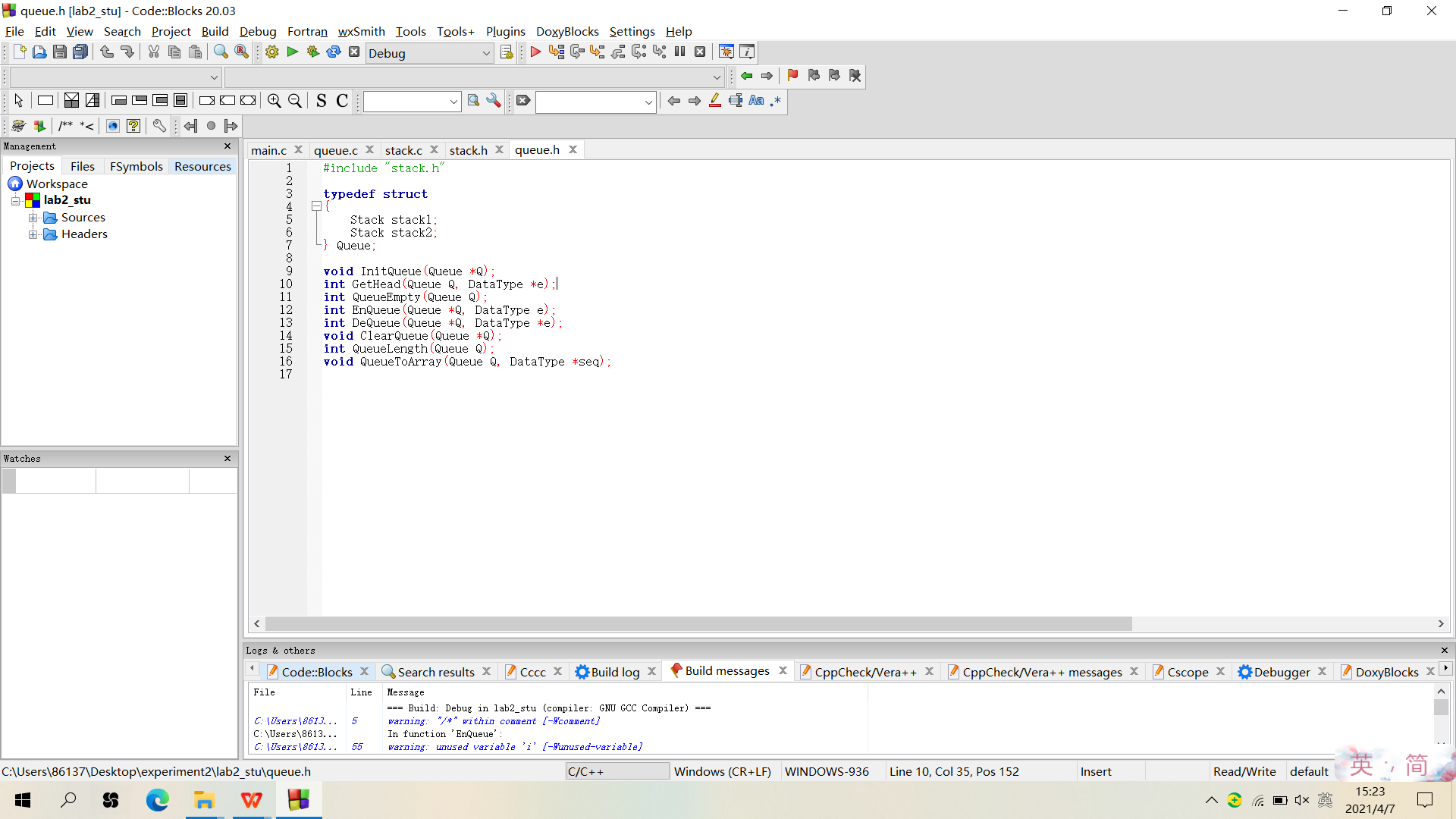
(1) 存储结构

栈：



用数组以及一个用来记录栈顶元素下标的栈顶指针来实现栈。

队列：



用两个栈来实现队列，栈1用来入栈，栈2用来出栈。

若栈1中有元素，则栈1的栈顶元素总代表队尾元素，否则栈2的栈底元素代表队尾元素；若栈2中有元素，则栈2的栈顶元素代表队头元素，否则栈1的栈底元素代表队头元素。

1. 涉及的操作（一般为自定义函数，可不写过程，但要注明该函数的含义。）

栈：

* int StackEmpty(Stack S)

参数：传入整个栈的结构体

功能：判断栈是否空

返回值：栈为空则返回1，否则返回0

* int Push(Stack \*s, DataType e)

参数：传入整个栈的结构体指针以及待插入的元素

功能：向栈插入一个元素

返回值：插入成功则返回1，否则返回0

* int Pop(Stack \*S, DataType \*e)

参数：传入整个栈的结构体指针以及用来带回栈顶元素的指针

功能：从栈中弹出一个元素

返回值：弹出成功则返回1，否则返回0

* int GetTop(Stack S, DataType \*e)

参数：传入整个栈的结构体以及用来返回栈顶元素的指针

功能：获取栈顶元素，不删除该元素

返回值：获取成功则返回1，否则返回0

队列：

* int QueueEmpty(Queue Q)

参数：传入整个队列的结构体

功能：判断队列是否为空

返回值：如果队列为空则返回1，否则返回0

* int EnQueue(Queue \*Q, DataType e)

参数：传入整个队列的结构体指针以及待入队的元素

功能：入队操作，将元素插入队列

返回值：如果插入成功则返回1，否则返回0

* int DeQueue(Queue \*Q, DataType \*e)

参数：传入整个队列的结构体指针以及用来带回队头元素的指针

功能：出队操作，从队列中取出一个元素

返回值：如果成功取出则返回1，否则返回0

* int GetHead(Queue Q, DataType \*e)

参数：传入整个队列的结构体以及用来返回队头元素的指针

功能：获取队列头（不删除元素）

返回值：如果成功获取则返回1，否则返回0

* void QueueToArray(Queue Q, DataType \*seq)

参数：传入整个队列的结构体以及用来传递拷贝数组的数组指针

功能：获取队列的一个数组拷贝，顺序为从队列头到队列尾

返回值：如果队列为空则返回1，否则返回0

## 2.3 程序整体流程



核心算法流程:

栈：

* int StackEmpty(Stack S)



* int Push(Stack \*s, DataType e)



* int Pop(Stack \*S, DataType \*e)



* int GetTop(Stack S, DataType \*e)



队列：

* int QueueEmpty(Queue Q)



* int EnQueue(Queue \*Q, DataType e)



* int DeQueue(Queue \*Q, DataType \*e)



* int GetHead(Queue Q, DataType \*e)



* void QueueToArray(Queue Q, DataType \*seq)

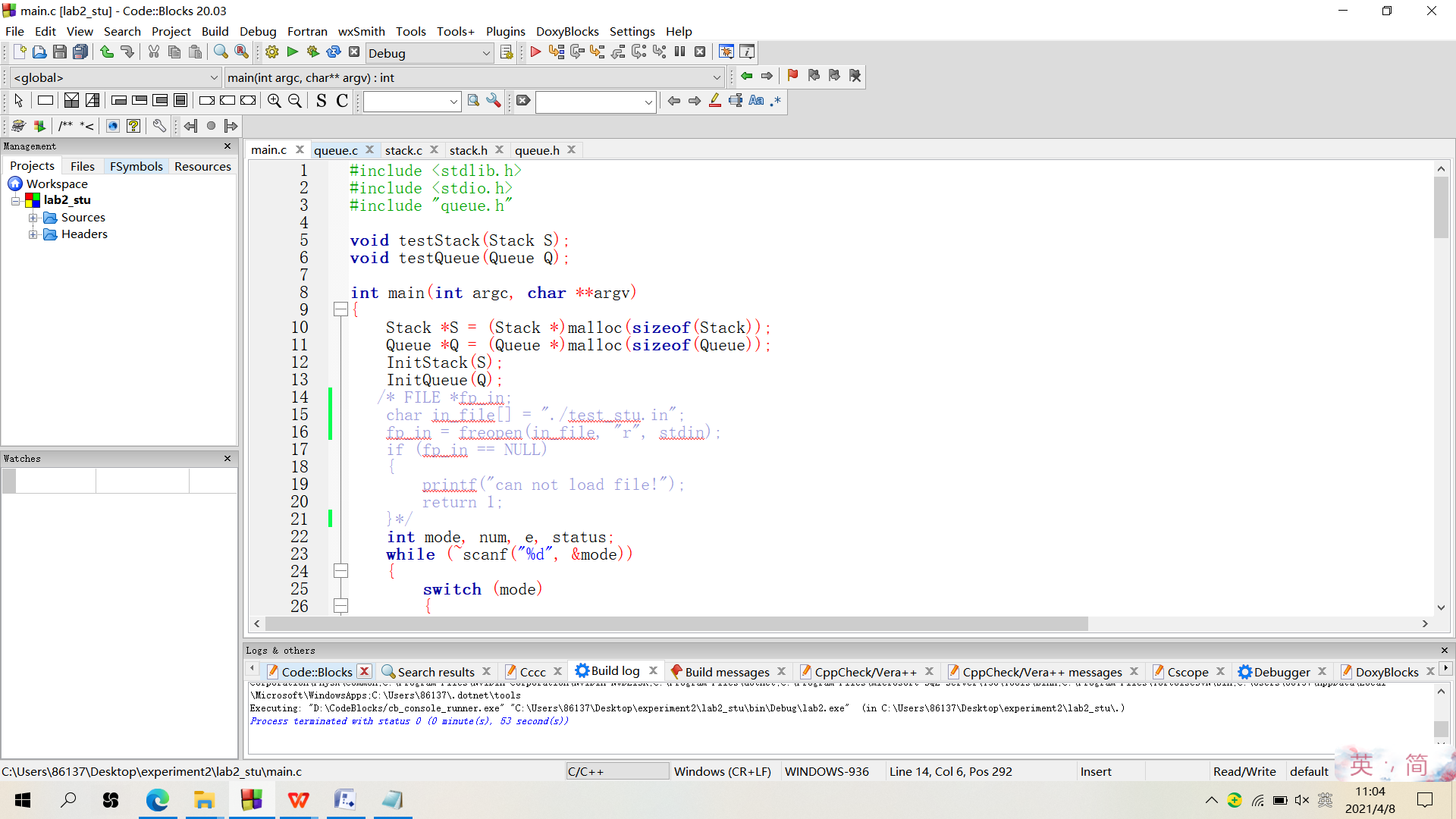


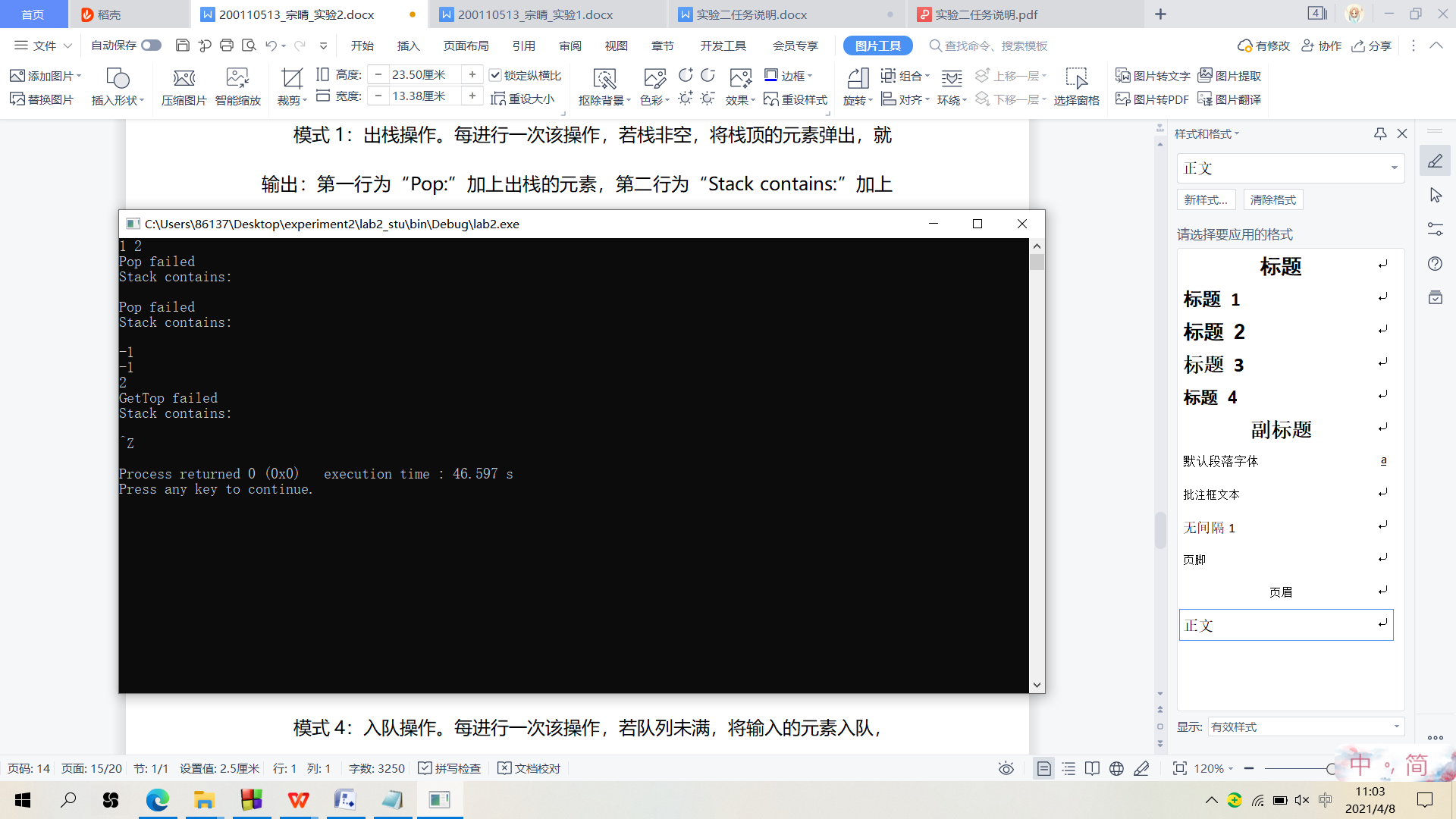
# 三、用户手册

输入数据的方式以及实现各种功能的操作方式：

程序共有8个模式可供选择。输入时每行的第一个整数（0-7）代表所选择的模式；每行第二个整数代表执行该模式的次数（只有入栈、出栈、入队、出队操作有这个参数）；对于入栈和入队操作，每行第三个参数开始为栈或队列的具体元素，元素个数应当与第二个整数的值相同。同一行中输入的数据之间应该用空格隔开。用户需要按行输入每次要执行的模式，如果输入0-7之外的模式，程序会自动忽略；如果想退出程序，则在一个模式运行完毕后按ctrl+z或者ctrl+d即可（同文件读取时的end of file）。

（如下图，注释掉读文件操作）当输入-1（在0-7之外的数）时，程序自动忽略该数据，输入ctrl+z时，程序结束。





模式0：入栈操作。每进行一次该操作，若栈未满，将输入的元素入栈，就输出：第一行为“Push:”加上入栈的元素，第二行为“Stack contains:”加上栈中的全部元素。若栈已满，就输出：第一行为“Push failed”，第二行为“Stack contains:”加上此时栈中的全部元素。

模式1：出栈操作。每进行一次该操作，若栈非空，将栈顶的元素弹出，就输出：第一行为“Pop:”加上出栈的元素，第二行为“Stack contains:”加上出栈后栈中的全部元素。若栈为空，就输出：第一行为“Pop failed”，第二行为“Stack contains:”加上此时栈中的全部元素（即没有元素）。

模式2：获取栈顶元素。每次进行该操作，若栈非空，获取栈顶的元素，就输出：第一行为“GetTop:”加上获取的元素，第二行为“Stack contains:”加上此时栈中的全部元素。若栈为空，就输出：第一行为“GetTop failed”，第二行为“Stack contains:”加上此时栈中的全部元素（即没有元素）。

模式3：判断栈是否为空。每次进行该操作，若栈为空，就输出：第一行为“The Stack is Empty”，第二行为“Stack contains:”加上此时栈中的全部元素（即没有元素）；若栈非空，就输出：第一行为“The Stack is not Empty”，第二行为“Stack contains:”加上此时栈中的全部元素。

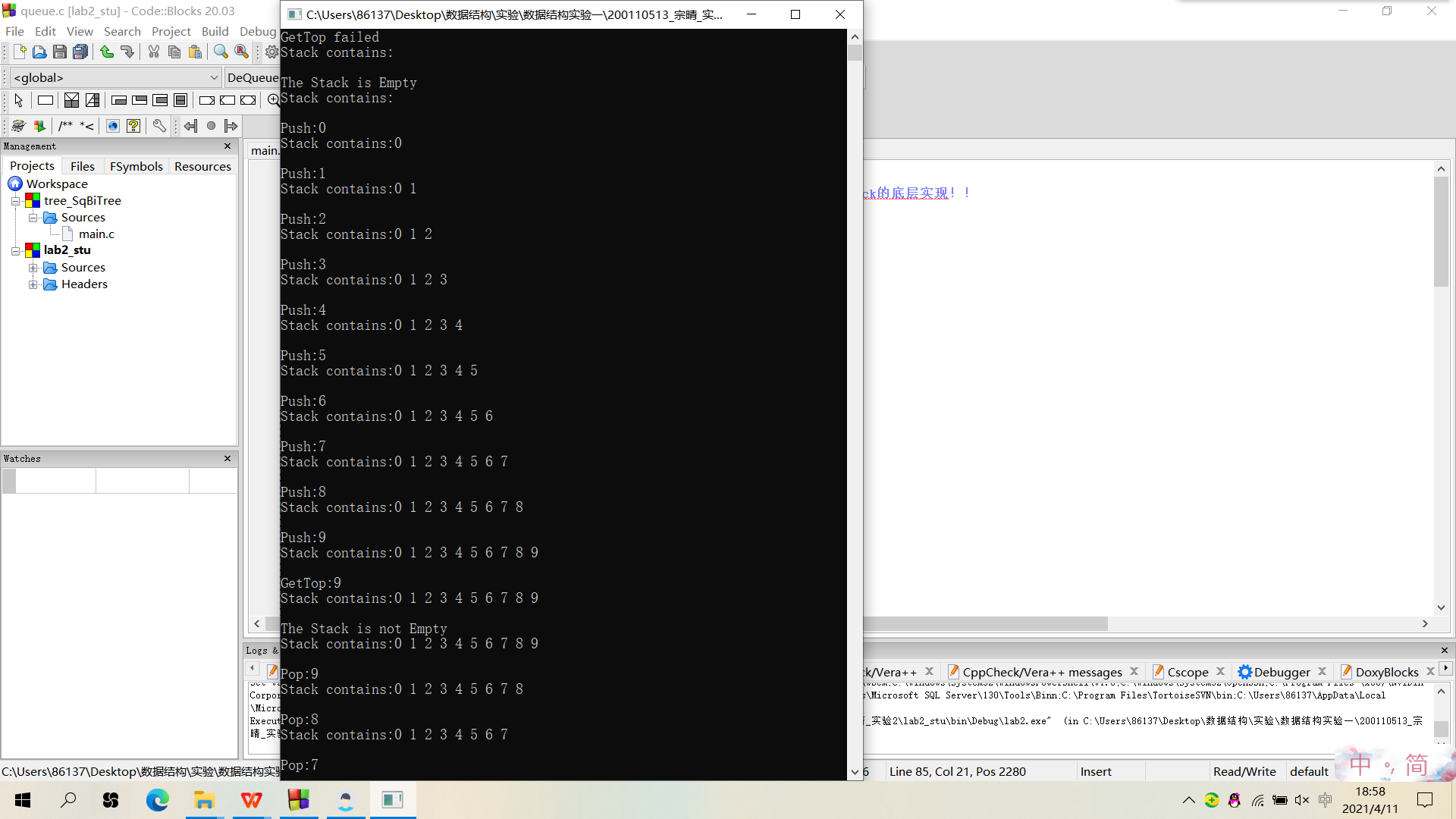
模式4：入队操作。每进行一次该操作，若队列未满，将输入的元素入队，就输出：第一行为“EnQueue:”加上入队的元素，第二行为“Queue contains:”加上队列中的全部元素。若队列已满，就输出：第一行为“EnQueue failed”，第二行为“Stack contains:”加上此时队列中的全部元素。

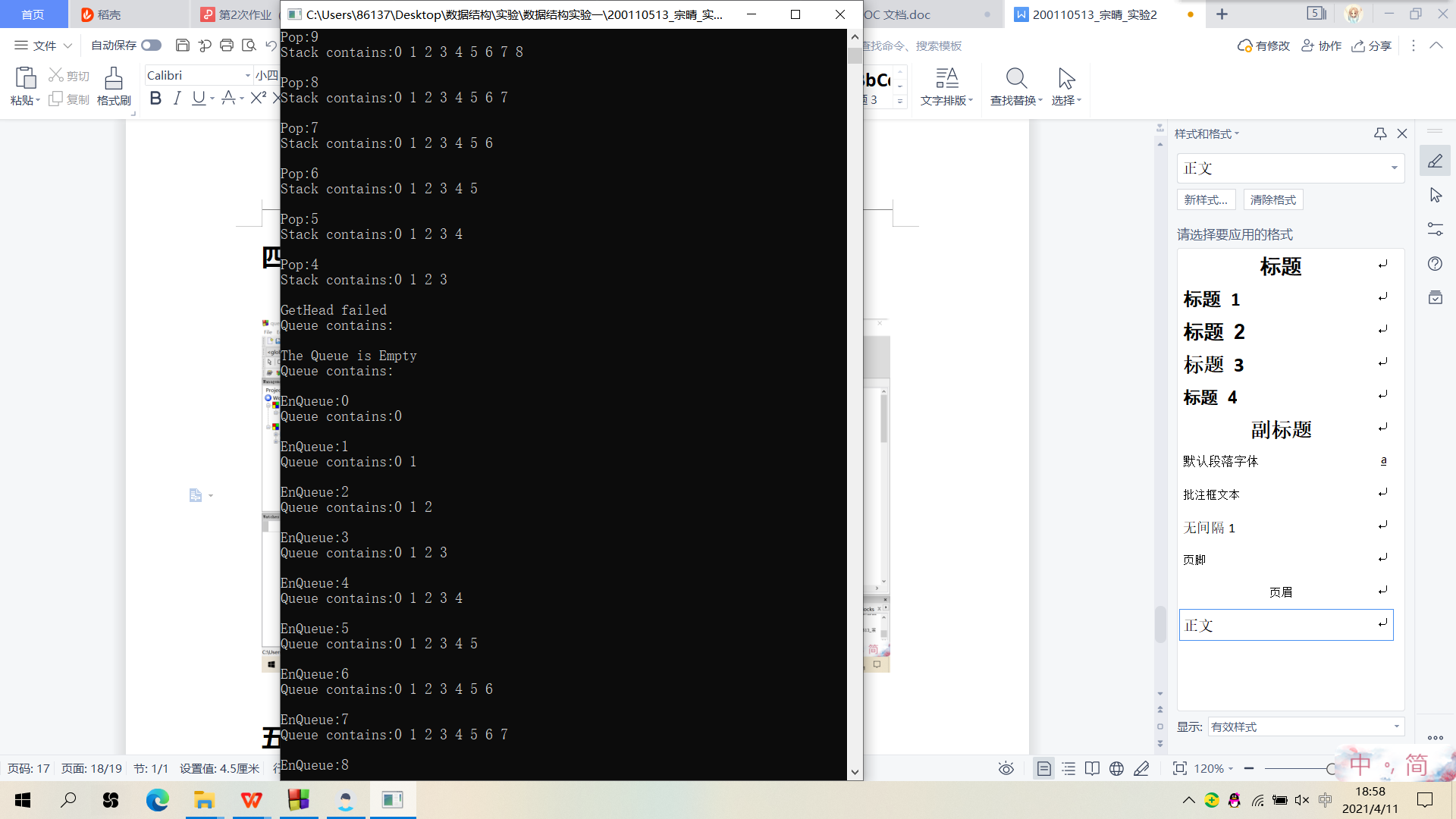
模式5：出队操作。每进行一次该操作，若队列非空，将队头的元素弹出，就输出：第一行为“DeQueue:”加上出队的元素，第二行为“Queue contains:”加上出队后队中的全部元素。若队列为空，就输出：第一行为“DeQueue failed”，第二行为“Queue contains:”加上此时队列中的全部元素（即没有元素）。

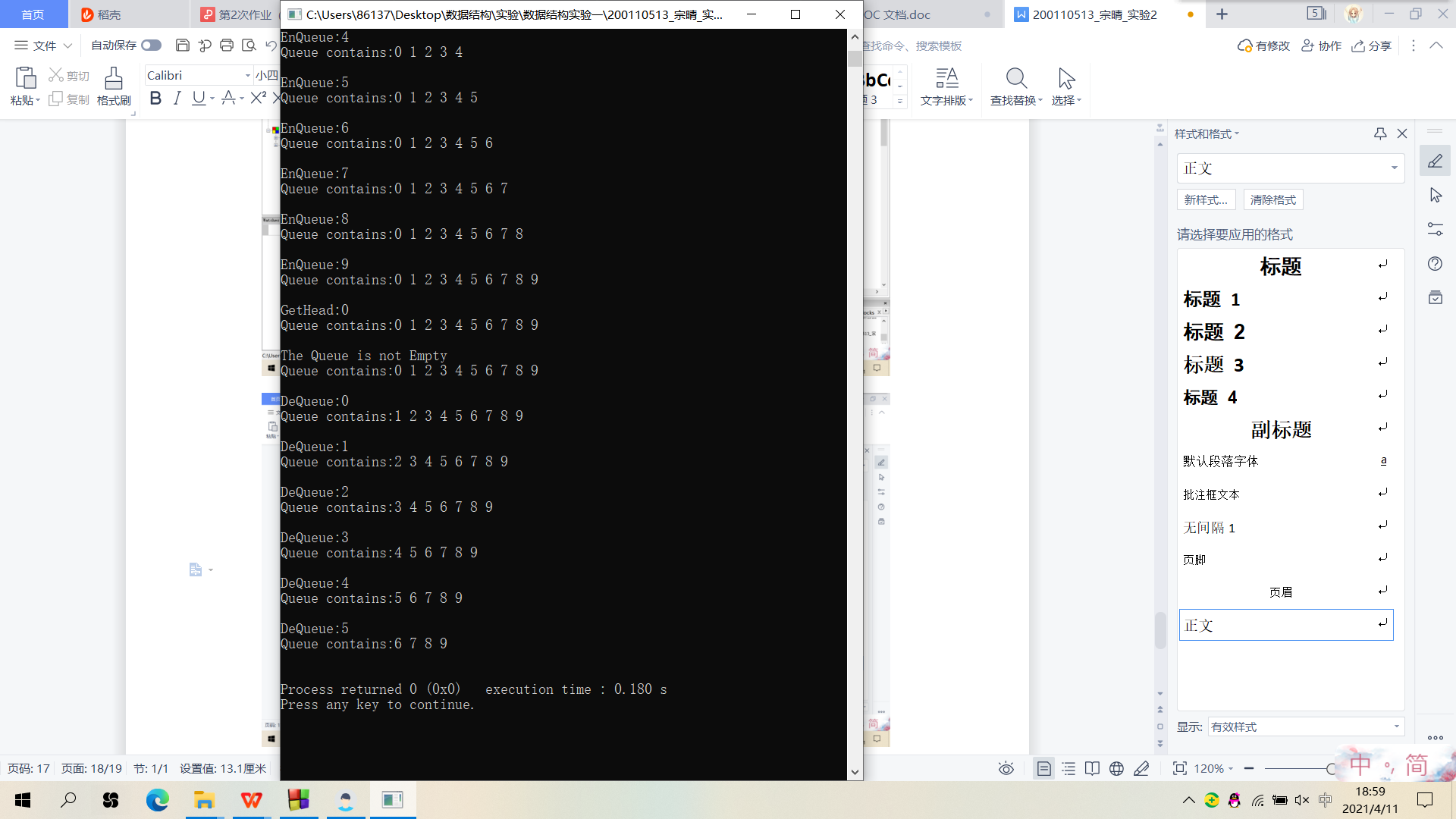
模式6：获取队头元素。每次进行该操作，若队列非空，获取队头的元素，就输出：第一行为“GetHead:”加上获取的元素，第二行为“Queue contains:”加上此时队列中的全部元素。若队列为空，就输出：第一行为“GetHead failed”，第二行为“Queue contains:”加上此时队列中的全部元素（即没有元素）。

模式7：判断队列是否为空。每次进行该操作，若队列为空，就输出：第一行为“The Queue is Empty”，第二行为“Queue contains:”加上此时队列中的全部元素（即没有元素）；若队列非空，就输出：第一行为“The Queue is not Empty”，第二行为“Queue contains:”加上此时队列中的全部元素。

# 四、结果







# 总结

该实验涉及到的数据结构主要是栈和队列，涉及的算法主要是入栈、出栈、获取栈顶元素、判断栈是否为空，以及用栈实现队列、入队、出队、获取队头元素、判断队列是否为空、获取队列的一个数组拷贝等操作。

在这次实验中，我第一次系统地学习了用两个栈实现队列的操作方式，起初我设计的算法是将所有数据存在一个栈中，另一个栈用来在入队和出队时临时用来反转栈的元素顺序，后来我发现这样的算法时间复杂度很高，于是我改成了用两个栈同时存储，一个用来入栈，另一个用来出栈，很好地降低了时间复杂度。

由于对队列中栈的操作只能通过已有函数实现，因此在获得一个队列拷贝数组时，由于栈拷贝数组的顺序固定，因此我拷贝出来的队列数组的顺序总是 出错，好在最终我发现了这一点，在函数中加上了将元素先转入栈1中的操作。

通过这次实验，我认识到了先封装基本操作函数对于实现整个程序的便利性以及它在实际程序书写时的意义。我熟悉了栈和队列的基本操作，对它们有了更深刻的认识。