作业练习2：类与数据结构

面向对象程序设计（C++)

WHUT-CS 2022 Spring

I. 作业目的与要求

本次实验主要在于学习使用C++类结构编程实现基本数据结构的功能，包括链表、栈和队列等。本次作业所涉及的功能代码应满足或遵循以下几点：

- **A**. 本次作业中，各数据结构所管理的数据对象均为整型数据，你可以在以后对代码进行改动和升级后使其能够管理更多可能的数据对象；
- **B**. 本次作业中，需要完成操作符重载的相关功能设计，对此可能存在多种解决方案，可根据设计和功能要求选择你认为合理的解决方案，包括操作符参数、返回值、前缀操作符、后缀操作符等实现细节的选择；
- **C**. 由于还未学习异常处理等相关知识，因此在使用部分库函数需要注意对异常的处理方式以避免程序意外退出。在部分库函数中，C++通过抛出异常对象（exception）而不是通过特定返回值（0或null）来表示错误情况，因此在完成部门功能的编写时，可能需要通过前置条件检查以确保程序不会抛出异常。完成作业时也可以通过查询相关章节或网络资源以正确使用异常；
- **D**. 严格保持整洁的代码格式、统一的命名规则和适当的代码注释以提高代码的整洁性和可读性；

> 注意：作业练习的考察范围涵盖本课程的全部学习内容，不完全遵循授课章节的顺序，也不完全遵循课堂讲授范围，部分知识点的学习和运用需根据教材或参考书内容加以自学

II. 作业内容

Task1: 准备源文件

本次作业任务要求至少包含如下7个源代码文件：

- **list.h**: 用于`List`类的功能声明头文件；
- **list.cpp**: 用于`List`类的功能实现源文件；
- **stack.h**: 用于`ArrStack`类和`ListStack`类的功能声明头文件；
- **stack.cpp**: 用于`ArrStack`类和`ListStack`类的功能实现源文件；
- **queue.h**: 用于`ArrQueue`类和`ListQueue`类的功能声明头文件；
- **queue.cpp**: 用于`ArrQueue`类和`ListQueue`类的功能实现源文件；
- **driver.h**: 包含主函数`main`的入口程序源文件；

Task2: 实现双向链表（doubly-linked list）功能

分别在**list.h**和**list.cpp**文件中编写代码完成双向链表的功能声明和实现，该类应满足以下一些功能要求：

- 使用`Node`类描述双向链表的组成节点，该类应包含如下数据成员：
  - `data`: 用于保存当前节点中的整型数据
  - `prev`: 指向链表中的前置邻接节点（左节点）的指针，如果当前节点没有前置邻接节点，则该数据成员应为空值；
  - `next`: 指向链表中的后置邻接节点（右节点）的指针，如果当前节点没有后置邻接节点，则该数据成员

应为空值；

- 使用`List`类描述双向链表的结构和功能，该类应包含如下组成部分：
  - private成员：
    - `head`：指向链表中的头节点（最左节点）的指针，如果当前链表是空链表，则`head`应为空值；
    - `tail`：指向链表中的尾节点（最右节点）的指针，如果当前链表是空链表，则`tail`应为空值；
    - `len`：用于存储链表的长度，即链表中共包含多少节点，如果当前链表是空链表，则`len`应为0；
  - public成员：
    - `List()`：缺省构造函数，用于初始化一个空的双向链表；
    - `List(int arr[], int num)`：构造函数，基于一个整型数组中所有元素构建一个双向链表，参数`arr`表示指定的整型数组，参数`num`表示整型数组的元素个数，整型数组中的所有元素被依次拷贝到双向链表中的各个节点；
    - `~List()`：析构函数，用于释放双向链表中的所有节点（可通过**delete**操作符释放节点对象）；

      > 注意：双向链表中的所有节点应通过**new**操作符在内存堆空间中动态创建，因此在双向链表的生命周期结束时，所有节点对象所占据的内存空间应通过**delete**操作进行释放

    - `void append(int n)`：功能函数，用于在双向链表的尾节点之后添加一个新节点，该节点保存参数`n`中传递的数据，并以该新节点作为链表新的尾节点；
    - `void prepend(int n)`：功能函数，用于在双向链表的头节点之前添加一个新节点，该节点保存参数`n`中传递的数据，并以该新节点作为链表新的头节点；
    - `bool delete_left(int &n)`：功能函数，用于从双向链表上删除其头节点（最左节点），并将该节点中的数据保存到参数`n`，如操作成功，则更新当前链表对象相应的数据成员状态并返回`true`，如果链表为空，则直接返回`false`；
    - `bool delete_right(int &n)`：功能函数，用于从双向链表上删除其尾节点（最右节点），并将该节点中的数据保存到参数`n`，如操作成功，则更新当前链表对象相应的数据成员状态并返回`true`，如果链表为空，则直接返回`false`；
    - `int length(void)`：功能函数，返回链表对象的长度；
    - `void print(void)`：功能函数，以适当形式打印输出链表对象的数据内容；
    - `operator<<(std::ostream ot, const List &lis)`：操作符<<重载，以适当形式打印输出链表对象的数据内容；

      > 例如：
      >
      > ```
      > std::out << lis; // prints all integers of the List lis.
      > ```

    - 继续声明更多功能函数实现以下功能：
      - 在链表中搜索指定的整型元素值，并返回其节点序号；
      - 从链表中删除指定的整型元素值；
      - 从链表中删除所有节点，将链表置为空链表；
      - 基于链表中的所有节点创建一个包含其中全部整型数据的数组；
      - ...

Task3: 基于数组实现数据栈（ArrStack）功能

分别在**stack.h**和**stack.cpp**文件中编写代码完成`ArrStack`类的功能声明和实现，该类的底层基于数组结构存储数据栈中的元素，并应满足以下一些功能要求：

> 注意：在**C++ Primer Plus**的第10章中有一个基于数组结构的数据栈样例。该样例与本题要求非常相似，可参考该样例完成本题作业

- private成员：
  - `ARR_MAX`：整型类常量，用于表示所有数据栈对象的最大容量。
  - `arr`：保存数据栈各个数据项的整型数组，其元素个数由`ARR_MAX`指定；
  - `size`：保存数据栈中当前数据项的个数，即值应介于0与`ARR_MAX`之间。该数据成员的值还可用于指示栈顶元素的位置，当向数据栈中插入新的数据项时用于确定其在数组中的存放位置；
- public成员：
  - `ArrStack()`：缺省构造函数，用于初始化一个空的数据栈，其`size`成员的值为0；
  - `~ArrStack()`：析构函数，用于释放数据栈对象及相关内存空间；

    > 注意：应根据程序的具体功能逻辑判断该析构函数是否有需要释放的动态内存空间或相关对象

  - `bool push(int n)`：压栈函数，用于在数据栈的栈顶（即`arr(size)`位置）压入参数`n`中传递的数据，并更新`size`以更新栈顶状态并返回`true`，如果数据栈的容量已满，则不进行压栈操作并返回`false`；
  - `bool pop(int &n)`：出栈函数，用于弹出数据栈顶的元素（即`arr(size-1)`位置）并将其值保存到引用参数`n`中，如操作成功，则更新`size`以更新栈顶状态并返回`true`，如果数据栈中不包含任何元素，则直接返回`false`；
  - `bool is_full() const`：功能函数，用于判断数据栈容量是否已满，如容易已满则返回`true`，否则返回`false`；
  - `bool is_empty() const`：功能函数，用于判断数据栈中是否存储有数据元素，如无元素则返回`true`，否则返回`false`；
  - `int get_size() const`：以内联函数形式返回当前数据栈中的元素个数；
  - `void print(void) const`：功能函数，以适当形式打印输出数据栈对象中的数据内容，数据元素从栈底到栈顶排列；
  - `ArrStack & operator+(int n)`：重载操作符`+`，使用该操作符可对参数`n`指定的整数执行压栈操作。压栈操作后，将当前数据栈对象作为操作结果返回值，以便支持操作符`+`的链式操作；

    > 例如：
    >
    > ```
    > ArrStack s; // initialize an empty stack.
    > s + 1; // push 1 onto the stack s
    > s + 100 + 9 + x; // push 3 items into the stack: two integers followed
    >              // by a variable x
    > ```

  - `ArrStack & operator-(int &n)`：重载操作符`-`，使用该操作符可对数据栈执行出栈操作，并将弹出的栈顶元素存入引用参数`n`中。出栈操作后，将当前数据栈对象作为操作结果返回值，以便支持操作符`-`的链式操作；

> 例如：
>
> ```
> ArrStack stk;
> stk + 3 + 4 +5;  // push 3 integers into the stack
> int x, y, z;
> stk - x - y - z; // pop 3 items and save them on x, y and z
> stk - x - x - x; // pop 3 items and all saved on x
> ```

- `std::ostream & operator<<(std::ostream &os, const ArrStack &stk)`：重载操作符`<<`，使用该操作符可对数据栈中保存的数据内容进行打印输出，数据项按照从栈底到栈顶的顺序依次打印，打印时对数据项不执行出栈操作；

> 例如：
>
> ```
> std::cout << someArrStack << " These integers on the stack " << std::endl;
> ```

Task4: 基于链表实现数据栈（ListStack）功能

分别在**stack.h**和**stack.cpp**文件中编写代码完成`ListStack`类的功能声明和实现，该类的底层基于本次作业**Task2**中编写的链表结构存储数据栈中的元素，并应满足以下一些功能要求：

- private成员：
  - `lis`：保存数据栈中各个数据项的整型链表，其元素个数可动态增长；
  - 你可以自行添加你认为必要的数据成员；
- public成员：
  - 所有在`ArrStack`类中声明的公共函数，注意使用`ListStack`替换其中出现的构造函数名、析构函数名、参数类型名等相关字段，以使`ListStack`和`ArrStack`拥有相同的公共接口；
  - 对于`ListStack`类而言，函数`is_full()`的作用并不明显，因为你可以为链表动态增加新的节点以扩充数据栈的容量，但你也可以选择设定一个常量以表示数据栈所允许容纳的最大数据个数；
  - 执行出栈操作时，必要时应该使用`delete`操作符释放出栈节点所对应的内存空间以避免潜在的内存溢出；
  - 构造器函数`ListStack()`可将数据成员`lis`初始化为一个空的链表
  - 当`ListStack`类的对象生命周期结束时，应该确保释放了内部的链表对象所拥有的内存空间

> 可以通过不同的方式释放相关内存空间
>
> 1. 当lis数据成员被销毁时，List类的析构函数会被自动调用，可利用List类的析构函数自动释放数据栈所拥有的内存空间；
> 2. 编写代码显式地释放lis所拥有的全部链表节点内存空间；

Task5: 基于循环数组实现数据队列（ArrQueue）功能

分别在**queue.h**和**queue.cpp**文件中编写代码完成`ArrQueue`类的功能声明和实现，该类的底层基于循环数组结构存储数据栈中的元素，并应满足以下一些功能要求：

> 注意：循环数组首尾相接，当存储到最后一个元素后，下一个元素从第一个位置开始存储

- private成员：
  - `ARR_MAX`：整型类常量，用于表示所有数据队列对象的最大容量。
  - `arr`：保存数据队列各个数据项的整型数组，其元素个数由`ARR_MAX`指定；
  - `size`：保存数据队列中当前数据项的个数，即值应介于0与`ARR_MAX`之间；
  - `front`：保存当前最早进入数据队列的数据项的索引位置，当数据队列为空时，该数据成员的值应为-1；
  - `end`：保存当前最后进入数据队列的数据项的索引位置，当数据队列为空时，该数据成员的值应为-1；

    > 注意：由于使用循环数组存储数据队列中的元素，因此下列公式总是满足
    >
    > End == (front + size -1) % ARR_MAX

- public成员：
  - `ArrQueue()`：缺省构造函数，用于初始化一个空的数据队列，其`size`为0，`front`和`end`都为-1；
  - `~ArrQueue()`：析构函数，用于释放数据栈对象及相关内存空间；

    > 注意：应根据程序的具体功能逻辑判断该析构函数是否有需要释放的动态内存空间或相关对象

  - `bool enqueue(int n)`：入队函数，用于在数据队列的尾部（即`arr(end)`位置）存入参数`n`中传递的数据，并更新`size`和`end`以保持正确的队列状态并返回`true`，如果数据队列的容量已满，则不进行入队操作并返回`false`；

    > 注意：改变end的数值时应注意符合循环数组的容量限制并正确更新其数值

  - `bool dequeue(int &n)`：出队函数，用于弹出数据队列的头部（即`arr(front)`位置）并将其值保存到引用参数`n`中，如操作成功，则更新`size`和`front`以保持正确的队列状态并返回`true`，如果数据队列中不包含任何元素，则直接返回`false`；
  - `bool is_full() const`：功能函数，用于判断数据队列容量是否已满，如容易已满则返回`true`，否则返回`false`；
  - `bool is_empty() const`：功能函数，用于判断数据队列中是否存储有数据元素，如无元素则返回`true`，否则返回`false`；
  - `int get_size() const`：以内联函数形式返回当前数据队列中的元素个数；
  - `void print(void) const`：功能函数，以适当形式打印输出数据队列中的数据内容；
  - `ArrQueue & operator+(int n)`：重载操作符`+`，使用该操作符可对参数`n`指定的整数执行入队操作。入队操作后，将当前数据队列作为操作结果返回值，以便支持操作符`+`的链式操作；
  - `ArrStack & operator-(int &n)`：重载操作符`-`，使用该操作符可对数据队列执行出队操作，并将弹出的队首元素存入引用参数`n`中。出队操作后，将当前数据队列作为操作结果返回值，以便支持操作符`-`的链式操作；

    > 例如：

```
ArrQueue aq; // initialize an empty queue
aq + 1 + 2 + 99;
aq + 100;
int v, w, x, y, z;
aq - x - y - z; // x is 1, y is 2, z is 99.
aq - w; // w is 100, queue is empty now;
aq - v; // v is not changed, because the queue is empty
```

- `std::ostream & operator<<(std::ostream &os, const ArrQueue &q)`：重载操作符`<<`，使用该操作符可对数据队列中保存的数据内容进行打印输出，打印时对数据项不执行出队操作；

  > 例如：
  >
  > ```
  > std::cout << someArrQueue << " These are the integers in the queue" <<
  > std::endl;
  > ```

Task6: 基于链表实现数据队列（ListQueue）功能

分别在**queue.h**和**queue.cpp**文件中编写代码完成`ListQueue`类的功能声明和实现，该类的底层基于本次作业**Task2**中编写的链表结构存储数据队列中的元素，并应满足以下一些功能要求：

- private成员：
  - `lis`：保存数据队列中各个数据项的整型链表，其元素个数可动态增长；
  - 你可以自行添加你认为必要的数据成员；
- public成员：
  - 所有在`ArrQueue`类中声明的公共函数，注意使用`ListQueue`替换其中出现的构造函数名、析构函数名、参数类型名等相关字段，以使`ListQueue`和`ArrQueue`拥有相同的公共接口；
  - 对于`ListQueue`类而言，函数`is_full()`的作用并不明显，因为你可以为链表动态增加新的节点以扩充数据队列的容量，但你也可以选择设定一个常量以表示数据队列所允许容纳的最大数据个数；
  - 执行出队操作时，必要时应该使用`delete`操作符释放出队节点所对应的内存空间以避免潜在的内存溢出；
  - 构造器函数`ListQueue()`可将数据成员`lis`初始化为一个空的链表
  - 当`ListQueue`类的对象生命周期结束时，应该确保释放了内部的链表对象所拥有的内存空间

Task7: 编写主函数功能

在**driver.cpp**文件中编写主函数，用其对之前编写的各个类中的功能函数进行调用，并通过打印输出判断功能调用的结果正确与否。

> 例如：

```
// testing the ArrStack
ArrStack as; //default constructor called
cout << as.size() << endl;
as.push(1);
as + 3;
cout << as.size() << endl;
int x, y, z;
as.pop(x);  // testing pop
as.pop(y);
cout << " x = " << x << "y = " << y << endl;
as + 99 + 100 + 101 - z; // testing the overloaded + and -
cout << as << endl; // testing the overloaded <<
```

Task8: 编译程序

将本次作业所有代码文件的编译指令以注释方式标注在**driver.cpp**文件的开头部分

V. 作业提交

- 所有同学请独立完成本次作业，不允许共同完成和抄袭
- 通过武汉理工大学"理工智课"平台中的本课程页面提交作业结果
  - 课程编号：69274 （面向对象程序设计B）
- 截止日期: 2022 Nov. 30 23:00

VI. 参考文献

[1] C++ Primer Plus (edition 6), Stephen Prata, ISBN: 978-0321-77640-2, Pearson

[2] C++语言程序设计 (第5版), 郑莉, 董渊, ISBN: 978-7302-56691-5, 清华大学出版社