# Lab01实验报告

班级：软外1903 姓名：秦政瀚 学号：201906150312

## Task 1a

### Theory:

* Using a static array to handle the AList problem.
* The array implements the list:
  + The memory layout of the array is compact.
  + Implement the list structure in a language that does not provide pointers.
  + Interprocess communication, using index is more reliable than using pointers - the pointer address in one process is meaningless in another process.
  + For some memory shortages, when its value type is integer and the value field matches the array index, you can reuse the next pointer with data, saving some memory.
  + Integer zero-sum to prevent memory fragmentation.

### Function:

* a construtor AList().
* a desctructor ~AList().
* int Size(void) that returns the number of elements in the list.
  + Using the function sizeof().
* void AddFirst(int element) that adds an element first in the list.
  + If the size equals to the capacity of the AList before adding, throw an error.
  + It's necessary to use for-loop to make room for the new first element.
* void AddLast(int element) that adds an element last in the list.
  + If the size equals to the capacity of the AList before adding, throw an error.
  + Add the new element last directly.
* int RemoveFirst(void) that deletes the first element in the list and returns its value.
  + If the size equals to zero before removing, throw an error.
  + Get the value of last element, which should be returned.
  + Decrease the number of elements.
* int RemoveLast(void) that deletes the last element in the list and reutrns its value.
  + If the size equals to zero before removing, throw an error.
  + The same with the RemoveFirst().
* int ReturnNth(int n) that returns the N th element in the list without removing the element from the list (When n=1 the first element is returned, for n = 2 the second element, and so on.).
  + It's necesaary to judge whether the parameter n is less, more than or equals to the size of the AList.
* bool Empty(void) that returns true if the list is empty and returns false otherwise.
* void PrintList(void) that prints the elements of the list.

### Attention:

* In order to make the code more robustness, it's better to add precondition to verify whether the size of array is underflowed or overflowed.
* It suggests that to throw exception in illegal conditions.

## Task 1b

### Theory:

* Using a doubly linked list structure to implement the ADT List.
* Node implements the list:
  + Anywhere in memory, no continuity is required.
  + Each data holds the memory address of the next data, through which the next data is found.
  + It's easy to add data and delete data.
  + Finding data is inefficient because it is not randomly accessible, so accessing data from a location starts with the first data.
  + No size is specified, easy to expand. The list size need not be defined, the data is arbitrarily added and deleted.

### Function:

* Definite the class Node which definites the attributes: T data(T is the template's type), Node\* prev, Node\* next.
* a construtor List().
* a desctructor ~AList().
* int getLength(void) that returns the number of elements in the list.
* void addFirst(int element) that adds an element first in the list.
  + Create the new node contains the parameter element.
  + If the list head is nullptr, set the new node as head, and set the next node nullptr, set the previous node nullptr. If not set the new node's next node as the head, set the head's previous as the new node and set the new node as head finally. **Attention: the order of the steps can't be reverse!**
* void addLast(int element) that adds an element last in the list.
  + Create the new node contains the parameter element.
  + If the list head is nullptr, set the new node as head, and set the next node nullptr, set the previous node nullptr.
  + It's necessary to use for-loop to find the current last node. And link the new node with the current last node.
* int removeFirst(void) that deletes the first element in the list and returns its value.
  + It's necessary to throw an error if the list is empty.
  + It's necessary to create variable to store the current head's pointer to next node and data temporarily.
  + Set the current node's next node as head. Set the pointer to previous node nullptr. Set the pointer to next nullptr. **Attention: the order of the steps can't be reverse!**
* int removeLast(void) that deletes the last element in the list and reutrns its value.
  + It's necessary to throw an error if the list is empty.
  + The similaranalogical theory with the removeFirst().
* int returnNth(int n) that returns the N th element in the list without removing the element from the list (When n=1 the first element is returned, for n = 2 the second element, and so on.).
* bool empty(void) that returns true if the list is empty and returns false otherwise.
* void printList(void) that prints the elements of the list.

## Task 2

### Theory:

* Using an array to implement the Stack.
* The biggest feature of the stack is the last-in, first-out(LIFO). For the data in the stack, all operations are done at the top of the stack, only the elements at the top of the stack can be viewed, only the data can be pressed into the top of the stack, and only the number of data can be ejected from the top of the stack.

### Function:

* a constructor .
* a destructor.
* int Size(void) that returns the number of elements in the stack.
* int Pop(void) that removes and returns the top element.
* void Push(int element) that puts an element on the top of the stack.
* int Peek(void) that returns the element on the top of the stack without removing it from the stack.
* bool Empty(void) that returns true if the stack is empty and false otherwise.

### The characters of Stack:

* Only can store, pop or retrieve the element in stack from its one side.
* A linear data structure.
* Each element in the stack should be FILO(first in last out).

## Task 3

### Theory:

* Using an array to implement the Queue.

### Function:

* a constructor.
* a destructor.
* int Size(void) that returns the number of elements in the queue.
* int Dequeue(void) that removes and returns the first element of the queue.
  + Using the point \*front point to head element and \*back to the last element.
* void Enqueue(int element) that add an element last in the queue.
* int First(void) that returns the first element without removing it from the queue.
* bool Empty(void) that returns true if the queue is empty and false otherwise.

### The characters of Queue:

* Only can store or retrieve the element in queue from its one side and delete or retrieve the element from the another side.
* A linear data structure.
* Each element in the stack should be FIFO(first in first out).

## Task 4

### Theory & Implemetation:

* The same theory with the classic algorithm problem--Joseph Ring.
* Using the single link list to implement.
* Using view counter to count different limit condition.