FRAGMENT LINKED LIST

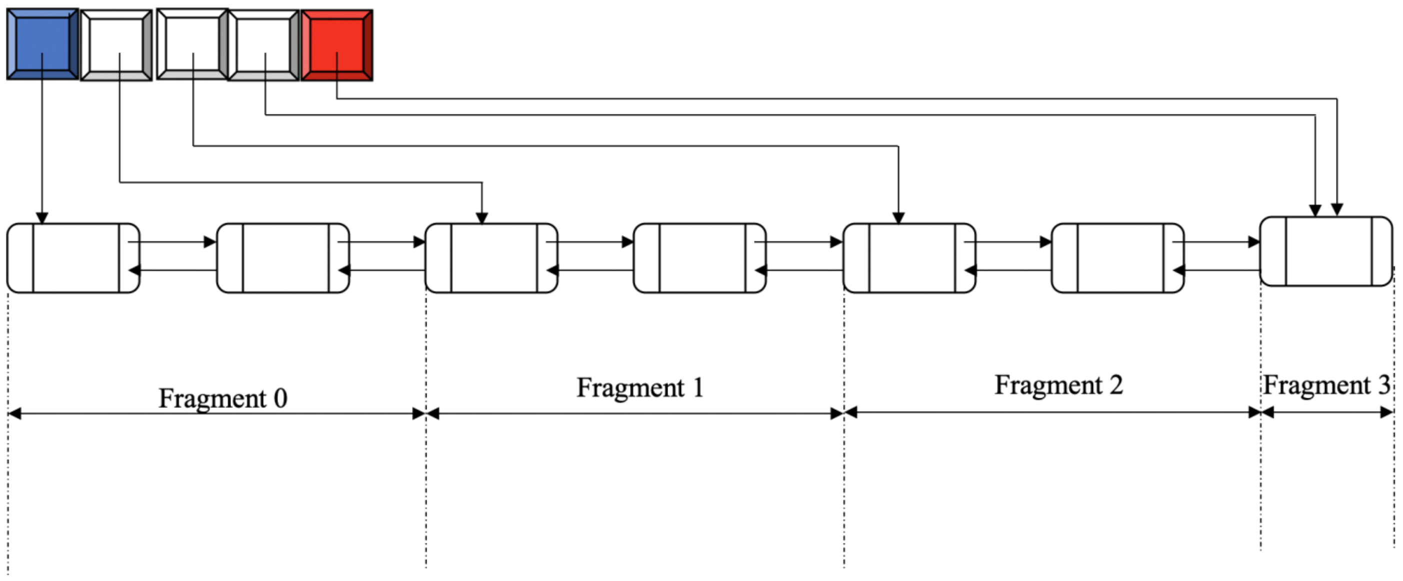
• Outcome

After successfully completing this project, I will be able to:

- Implement a derivative of doubly linked list, especially fragment linked list.
- Know how to use a list data structure.

• Tasks

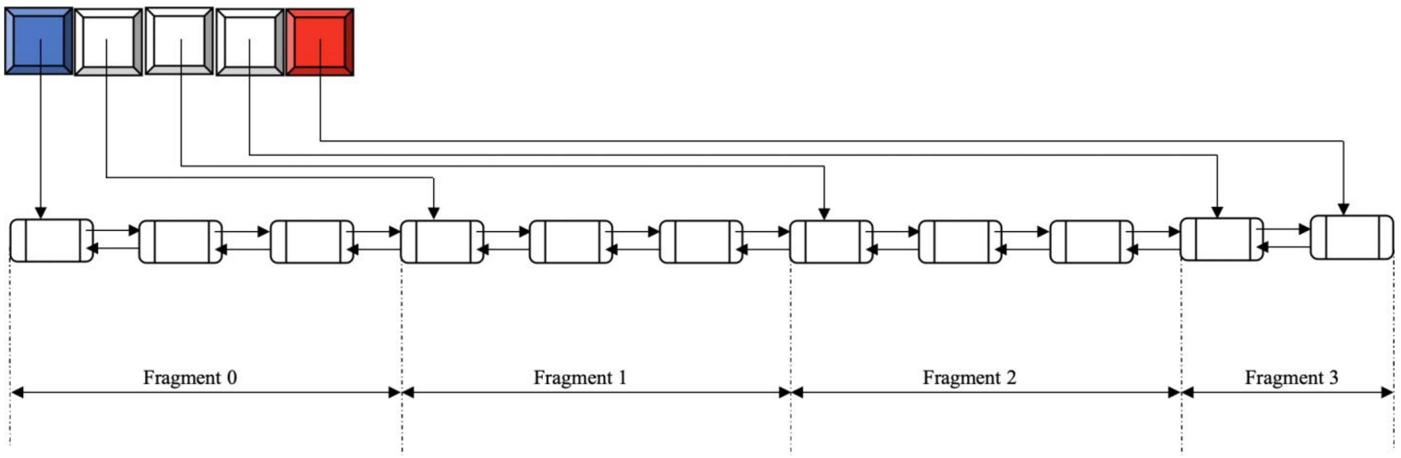
The job required to build a C++ program to implement the idea of fragment linked list, which is illustrated in Picture 1 and Picture 2.



Picture 1: FLL with 7 elements, each fragment has the maximum size of 2 elements

A fragment linked list is implemented based on the doubly linked list, in which the list is divided into *fragment* with maximum size*fragment_max_size*.

In order to store these fragments, list of pointers containing the address of nodes at the beginning of each fragment, where:first pointer (in blue) in this list points to the first node of the list, also the first node in the first fragment.The last pointer (in red) in this list points to the last node of the list.



Picture 2: FLL with 11 elements, each fragment has the maximum size of 3 elements

The other pointers (in white) in this list points to the first node of each fragment in the order of fragments.

- ## File structure:

- Class IList(with template T) demonstrates interface (as a blueprint) of general list data structure, containing following methods:
  - void add(T e)   : add a new element into the end of the list.
  - void add(int index,T e)   : add a new element into index index position.
  - T removeAt(int index): delete the element at indexposition and return that element.
  - bool removeItem(T item): delete the element has value itemand return whether that element can be found.
  - bool empty(): checking whether the list is empty.
  - int size(): return the size (the number of elements) of the current list.
  - void clear(): delete all current elements in the list.
  - T get(int index): find and return the element at index position.
  - void set(int index,const T& element): set new value for element index.

    - int indexOf(T item): find and return the position of the element which has value item in the list.
    - bool contains(T item): check whether the list contains element that has value item.
    - string toString(): return the list in the form of string.
- Class FragmentLinkedList (with template T) is the fragment list data structure needs implementation:
    - Nested class Node and Iterator respectively illustrate the nodes in the list and the object used for iterating actions in the list. The methods are overridden based on the interface IList.
    - Two methods for class Iterator: Iterator:
        - Iterator begin(int fragmentIndex = 0): return the first Iterator corresponding to fragmentIndex.
          E.g: with fragmentIndex=1, return Iterator corresponding to the first element in fragment 1.
        - Iterator end(int fragmentIndex = -1): return the next of the last Iterator corresponding to fragmentIndex. With fragmentIndex = -1, return Iterator corresponding to the next of last element in the list (NULL element).
          E.g: with fragmentIndex=1, return Iterator corresponding to the next of last element in fragment 1.
    - Methods in class Iterator:
        - Iterator(FragmentLinkedList<T>*,bool) : set pNodeto the first node (index = 0) in the list pointed by pList when begin =true , otherwise points to NULL
          (index = pList->size()).
        - Iterator(int,FragmentLinkedList<T>*,bool): pNode points to the first node in fragment in list pointed by pList when begin =true , otherwise points the next node of the fragment's last node.
        - Iterator& operator= (const Iterator&iterator): assignment operator in Iterator to do assign the corresponding attributes with input iterator, andre- turns this Iterator.
        - T& operator*() : return datain pointed node. In case of NULL, throw an exception std::out_of_range("Segmentation fault!")
        - bool operator!= (const Iterator& iterator): inequality operator in Iterator, returns true if there is different in pointed memory or index.

- void remove(): remove node which iterator points. After removal, node points to the previous node. In case of head, pNodeis assigned to NULLwith index=-1.
- void set(constT& element) : set the new value for pNode.
- Iterator& operator++() : prefix operator++ which sets pNode to the next element and increase index by 1.
- Iterator operator++(int): postfix operator++ which sets pNode to the next element and increase index by 1.