Homework #2 B09705039 劉惟恩

Solutions:

1.

```
// aBag is given as a bag of integers so template is no needed in this case
// BagInterface is defined in class (P.18)
// applying Array-based Implementation

class aBag : public BagInterface
{
private:
    static const int DEFAULT_CAPACITY = 6;
    int items[DEFAULT_CAPACITY];
    int itemCount;
    int maxItems;

    // member function for question b
    int getIndexOf(const int &beReplaced) const;

public:
    // is already defined in class (P.24)
    ArrayBag();
    int getCurrentSize() const;
    bool isEmpty() const;
    bool add(const int &newEntry);
    bool remove(const int &anEntry);
    void clear();
    bool contains(const int &anEntry) const;
    int getFrequencyOf(const int &anEntry) const;

    // Answers for question 1

        // question a
    int sum() const;

        // question b
    bool replace(const int &toReplace, const int &beReplaced);
};


// question a
int aBag::sum() const
{
    int sum = 0;
    for(int i = 0; i < itemCount; i++)
        sum += items[i];
    return sum;
};
```

```
// question b
int aBag::getIndexOf(const int &beReplaced) const
{
    bool found = false;
    int result = -1;
    for(int index = 0; index < itemCount; index++)
    {
        if(items[index] == beReplaced)
        {
            result = index;
            break;
        }
    }
    return result;
};

bool aBag::replace(const int &toReplace, const int &beReplaced)
{
    int index = getIndexOf(beReplaced);
    bool canBeReplaced = (index != -1);
    if(canBeReplaced)
    {
        items[index] = toReplace;
    }
    return canBeReplaced;
};
```

2.

```
/*
ADT design Rectangle
characteristics(data): length, width
behaviors(function): area(), perimeter()
*/

class RectangleInterface
{
public:
    virtual int area() const = 0;
    virtual int perimeter() const = 0;
};

class Rectangle : public RectangleInterface
{
private:
    int length = 0;
    int width = 0;
public:
    Rectangle(int len, int wid);
    int area() const;
    int perimeter() const;
};
```

```
//setting and retrieving the dimensions of the rectangle
Rectangle::Rectangle(int len, int wid)
{
   length = len;
   width = wid;
};

//finding the area of the rectangle
int Rectangle::area() const
{
   return (length * width);
};

//finding the perimeter of the rectangle
int Rectangle::perimeter() const
{
   return ((length + width) * 2);
};
```

3.

a.

```
template <typename ItemType>
class Node
{
private:
   ItemType item;
   Node<ItemType> *next;
   Node<ItemType> *previous;

public:
   Node();
   Node(const ItemType &anItem, Node<ItemType> *previousNodePtr);
   Node(const ItemType &anItem, Node<ItemType> *nextNodePtr);
   Node(const ItemType &anItem, Node<ItemType> *previousNodePtr, Node<ItemType>
*nextNodePtr);
   void setItem(const ItemType &anItem);
   void setPrevious(Node<ItemType> *previousNodePtr);
   void setNext(Node<ItemType> *nextNodePtr);
   ItemType getItem() const;
   Node<ItemType> *getPrevious() const;
   Node<ItemType> *getNext() const;
};

//constructors
//default constructor
template <class ItemType>
Node<ItemType>::Node() : previous(nullptr), next(nullptr)
{
};
//initial Item and previous pointer constructor
template <class ItemType>
```

```
Node<ItemType>::Node(const ItemType &anItem, Node<ItemType> *previousNodePtr) :
item(anItem), previous(previousNodePtr), next(nullptr)
{
};
//initial Item and next pointer constructor
template <class ItemType>
Node<ItemType>::Node(const ItemType &anItem, Node<ItemType> *nextNodePtr) :
item(anItem), previous(nullptr), next(nextNodePtr)
{
};
//initial Item, previous pointer and next pointer constructor
template <class ItemType>
Node<ItemType>::Node(const ItemType &anItem, Node<ItemType> *previousNodePtr,
Node<ItemType> *nextNodePtr) : item(anItem), previous(previousNodePtr),
next(nextNodePtr)
{
};

//setter
//setItem
template <class ItemType>
void Node<ItemType>::setItem(const ItemType &anItem)
{
   item = anItem;
}
//setPrevious
template <class ItemType>
void Node<ItemType>::setPrevious(Node<ItemType> *previousNodePtr)
{
   previous = previousNodePtr;
}
//setNext
template <class ItemType>
void Node<ItemType>::setNext(Node<ItemType> *nextNodePtr)
{
   next = nextNodePtr;
}

//getter
//getItem
template <class ItemType>
ItemType Node<ItemType>::getItem() const
{
   return item;
}
//getPrevious
template <class ItemType>
Node<ItemType> *Node<ItemType>::getPrevious() const
{
   return previous;
}
//getNext
```

```
template <class ItemType>
Node<ItemType> *Node<ItemType>::getNext() const
{
   return next;
}
```

b.
step1: Use newNodePtr to new a Node by the default constructor, setItem(newEntry), setNext(headPtr) for the new node.
step2: headPtr = newNodePtr, setPrevious(newNodePtr) for the second node.
(step3: If need to count the items : itemCount ++, and return true as finished.)

c.
step1: Use the function getPointerTo(anEntry) to find the location of the item that we wanted to remove and put it in a pointer entryNodePtr.
step2: If it can be removed, we use the item of the beginning node to rewrite the item of the node to be removed (which the entryNodePtr pointed at).

// if the node item to remove is not the first node we need to do the above first (if it is a bag ADT) else we can start from the third step

step3: Use a pointer nodeToDeletePtr to point to the beginning node.
step4: Point the head pointer to the second node and setPrevious(nullptr).
step5: Delete the node that nodeToDeletePtr pointed at, and set nodeToDeletePtr as a nullptr.
(step6:If need to count the items : itemCount- -, and return true as finished.)