LAB SESSION 2
ADVANCED PROCESSING in LINKED LIST

3. EXERCISES

Linked List

Consider the file ListSample.cpp attached. Use this initial code to accomplish the following tasks.

3.1. Use the defined List, build a linked list of integers as follows \{12, 5, 79, 82, 21, 43, 31, 35, 57\}.

```cpp
void main() {
    List<int> aList;
    Error_code err;
    int x;
    x = 57;   //"insert(int position, List_entry &x)" requires a variable for the input x
    err = aList.insert(0, x);  //can be called without the return value (err)
    x = 35;                        //i.e. only aList.insert(0, 57);
    err = aList.insert(0, x);
    x = 31;
    err = aList.insert(0, x);
    x = 43;
    err = aList.insert(0, x);
    x = 21;
    err = aList.insert(0, x);
    x = 82;
    err = aList.insert(0, x);
    x = 79;
    err = aList.insert(0, x);
    x = 5;
    err = aList.insert(0, x);
    x = 12;
    err = aList.insert(0, x);
}
```

Listing 8

3.2. Uncomment the method printAll in the file, implement it and use it to display the list built in Exercise 3.1.
3.3. Uncomment the rest of commented methods and implement them. Write some pieces of code in the main function to test your implemented methods.

Listing 9

template<class List_entry>
void List<List_entry>::printAll() {
    Node<List_entry> *tmp = head;
    while (tmp != NULL) {
        cout << tmp->entry;
        tmp = tmp->next;
    }
}

//-------------------------------
template<class List_entry>
Error_code List<List_entry>::remove(int position, List_entry &x) {
    if ((position >= count) || (position < 0))
        return underflow;

    Node<List_entry> *pDel;
    if (position == 0) {
        pDel = head;
        head = pDel->next;
        delete pDel;
    } else {
        Node<List_entry> *pPre = set_position (position-1);
        pDel = pPre->next;
        pPre->next = pDel->next;
        x = pDel->entry;
        delete pDel;
    }
    count--;
    return success;
}

//-------------------------------
template<class List_entry>
Error_code List<List_entry>::retrieve(int position, List_entry &x) {

```cpp
if ((position >= count) || (position < 0))
    return underflow;
Node<List_entry>* pLoc = set_position(position);
x = pLoc->entry;
return success;
```

**Listing 10**

3.4 Write an additional method to remove the last element in a linked list. If the list is empty, nothing happens.

```cpp
template<class List_entry>
void List<List_entry>::removeLast() {
    if (count == 0)
        return;
    Node<List_entry>* pPre = NULL, *pDel = head;
    while (pDel->next != NULL) {
        pPre = pDel;
        pDel = pDel->next;
    }
    if (pPre == NULL) //only one element in the list
        head = NULL;
    else
        pPre->next = NULL;
    delete pDel;
count--;
}
```

**Listing 11**

3.5 Write an additional method to remove the last element, which is equal to an input data, in a linked list.

**Example**

```
Example aList = {1,2,3,4,5,6,7,3,8,9,3,0,0,2} 
Remove the last 3 in aList => {1,2,3,4,5,6,7,3,8,9,0,0,2}
```

```cpp
template<class List_entry>
void List<List_entry>::removeLastX(List_entry x) {
    if (count == 0)
        return;
    Node<List_entry>* pPre = NULL, *pPre_save = NULL, *tmp = head;
    bool found = false;
    while (tmp != NULL) {
```
if (tmp->entry == x) {
    pPre_save = pPre; //remember the element before the last x
    found = true;
}
pPre = tmp;
tmp = tmp->next;
}
if (found) {
if (pPre_save == NULL) { //remove the first
    tmp = head;
    head = tmp->next;
} else {
    tmp = pPre_save->next;
    pPre_save->next = tmp->next;
}
delete tmp;
count--;
}

Listing 12

3.6 Write an additional method to remove the first three elements in a linked list. If the list has less than 4 elements, the resulted list is empty.

template<class List_entry>
void List<List_entry>::removeFirstThree() {
    Node<List_entry> *tmp = head;
    int i=0;
    while ((tmp != NULL) && (i < 3) {
        head = tmp->next;
        delete tmp;
        count--;
        tmp = head;
        i++;
    }
}

Listing 13

3.7 Write an additional method to reverse a linked list.
template<class List_entry>
void List<List_entry>::reverse() {
    Node<List_entry> *tmp1 = NULL, *tmp = NULL;
    while (head != NULL) {
        tmp = head;
        head = head->next;
        tmp->next = tmp1;
        tmp1 = tmp;
    }
    head = tmp;
}

Listing 14

3.8. Write an additional method to remove all occurrences of an input in a linked list.

Example  
aList = \{1,2,3,4,5,6,7,3,8,9,3,0,0,2\}

Remove all 3 in aList => \{1,2,4,5,6,7,8,9,0,0,2\}

template<class List_entry>
void List<List_entry>::removeAllX(List_entry x) {
    Node<List_entry> *pPre = NULL, *tmp = head;
    while (tmp != NULL) {
        if (tmp->entry == x) {
            if (pPre == NULL) {
                head = tmp->next;
                delete tmp;
                tmp = head;
            } else {
                pPre->next = tmp->next;
                delete tmp;
                tmp = pPre->next;
            }
            count--;
        } else {
            pPre = tmp;
            tmp = tmp->next;
        }
    }
}

Listing 15
cuu duong than cong . com

cuu duong than cong . com