# Лабораторийн ажил №11

Н. Мөнхжин

ХШУИС, МКУТ, Компьютерийн ухаан, nyamkamunhjin@gmail.com

## 1. ОРШИЛ.

C++ хэлний ойлголт болох загвар класс, загвар функц ашиглан лист өгөгдлийн бүтцийг үүсгэнэ.

## 2. ЗОРИЛГО

Үүсгэх гэж байгаа лист өгөгдлийн бүтэц нь ямар ч төрлийн объект дээр ажиллахаар байх ёстой тул template-ээр хийх.

1. Лист-ийн бүтцийг зохион байгуулах.
2. Шаардлагатай функцуудыг тодорхойлох.

## 3. ОНОЛЫН СУДАЛГАА

### 3.1 Linked-List, Vector

Linked-List нь хоорондоо хаягаар холбогдсон өгөгдлийн дарааллыг хэлдэг. Хүснэгттэй адилхан үйлдэл хийдэг ч нэг гол ялгаа нь Linked-List нь Динамик байдлаараа ялгарна. Нэг ёсондоо хүснэгтийн хэмжээг зарласаны дараа ихэсгэж багасгах боломжтой.[1]

### 3.2 Загвар класс, загвар функц

Загвар класс нь ямар ч класстай ажилж болдог ба ажиллах классуудыг ерөнхийд нь авч үздэг. Ямар ч төрлийн объект дамжуулсан байсан түүндээ тохируулж хувьсан өөрчлөгддөг. [2]

## 4. ХЭРЭГЖҮҮЛЭЛТ

Linked-List-ийн зохион байгуулалт

```
template <class T>

class Node {

public:
```

```
    T value;
    Node *next;
};


template <class T>
class List{
public:
    int length;
    Node<T> *head;
    Node<T> *tail;
};
```

- Node нь дотроо 1 утга болон дараа Node-ний хаягийг хадгална.
- List нь дотроо Эхний Node сүүлийн Node-үүдийн хаягийг хадгалах ба эхний Node болон сүүлийн Node-үүдийн дундах Node-үүд хаягаар холбоосоор холбогдсон байна.
-

**Түүн дээр ажиллах функуудыг хавсралтаас харж болно.**

## 5. ДҮГНЭЛТ

Загвар классаар Linked-List-ийг хийснээр дахин програмчлах шаардлагагүй болж илүү ажлыг хийхээс хэмнэж байна.

## 6. АШИГЛАСАН МАТЕРИАЛ

1. https://www.geeksforgeeks.org/data-structures/linked-list/

2. http://www.cplusplus.com/doc/oldtutorial/templates/

## 7. ХАВСРАЛТ

```
#include <stdio.h>
```

```cpp
#include <iostream>
#include <string>
using namespace std;

template <class T>
class Node {
public:
    T value;
    Node *next;
    Node() {
    }

    Node(T value) {
        this->value = value;
        next = NULL;
    }
};


template <class T>
class List{
public:
    int length;
    Node<T> *head;
    Node<T> *tail;

    List() {
        // printf("Constructor called!");
        length = 0;
        head = NULL;
        tail = NULL;
    }
    void print() {
```

```
    Node<T> *temp = head;
    int index = 0;
    while(temp != NULL) {
        cout << index << ": " << temp->value << endl;
        temp = temp->next;
        index++;
    }
}
void add(T value) {
    if(head == NULL) { // if list is empty
        head = new Node<T>(value);
        tail = head;
    } else { // add node after tail
        tail->next = new Node<T>(value);
        tail = tail->next;
    }

    // increase length by 1
    length++;
}

void insert(T value, int index) {
    // if list is empty or if index higher than length
    if(head == NULL || index >= this->length) {
        this->add(value);
        return;
    }

    if(index == 0) {
        Node<T> *temp = new Node<T>(value);
        temp->next = head;
        head = temp;
```

```
            length++;
            return;
        }

        int i = 1;
        Node<T> *temp = head;
        while(i < index) {
            temp = temp->next;
            i++;
        }
        // cout << "debug: " << temp->value << endl;
        // save next temp
        Node<T> *tempNext = temp->next;

        // assign new node to next temp
        temp->next = new Node<T>(value);
        temp->next->next = tempNext;

        length++;
    }

    auto get(int index) {
        int i = 0; // counter
        Node<T> *temp = head;
        while(i < index) { // counter until index
            temp = temp->next;
            i++;
        }
        return temp->value;
    }

    void deleteNode(int index) {
        if(head == NULL) {
```

```
        cout << "List empty\n";
        return;
    }

    int i = 0;  // counter
    Node<T> *after = head;
    Node<T> *before;
    while(i < index) { // count until previous index
        before = after;
        after = after->next;
        i++;
    }

    if(head == after) {
        head = head->next;
        delete after;
        return;
    }

    if(tail == after) {
        tail = before;
        delete after;
        tail->next = NULL;
        return;
    }


    before->next = after->next; // re connecting the link
    delete after; // deleting the node

    length--; // decrease length
}
```

```
    int getLength() {
        return this->length;
    }
};


int main() {
    List<int> link;
    link.add("1");
    link.add("2");
    link.add("3");
    link.add("4");
    link.add("5");
    link.insert("insert", 5);
    // link.deleteNode(5);
    link.print();


    // cout << link.get(3) << endl << link.getLength() << endl;
    return 0;
}
```