* Singly Linked List 의 Additional 리스트 연산

1) Chain을 역순으로 변환하는 연산(invert)

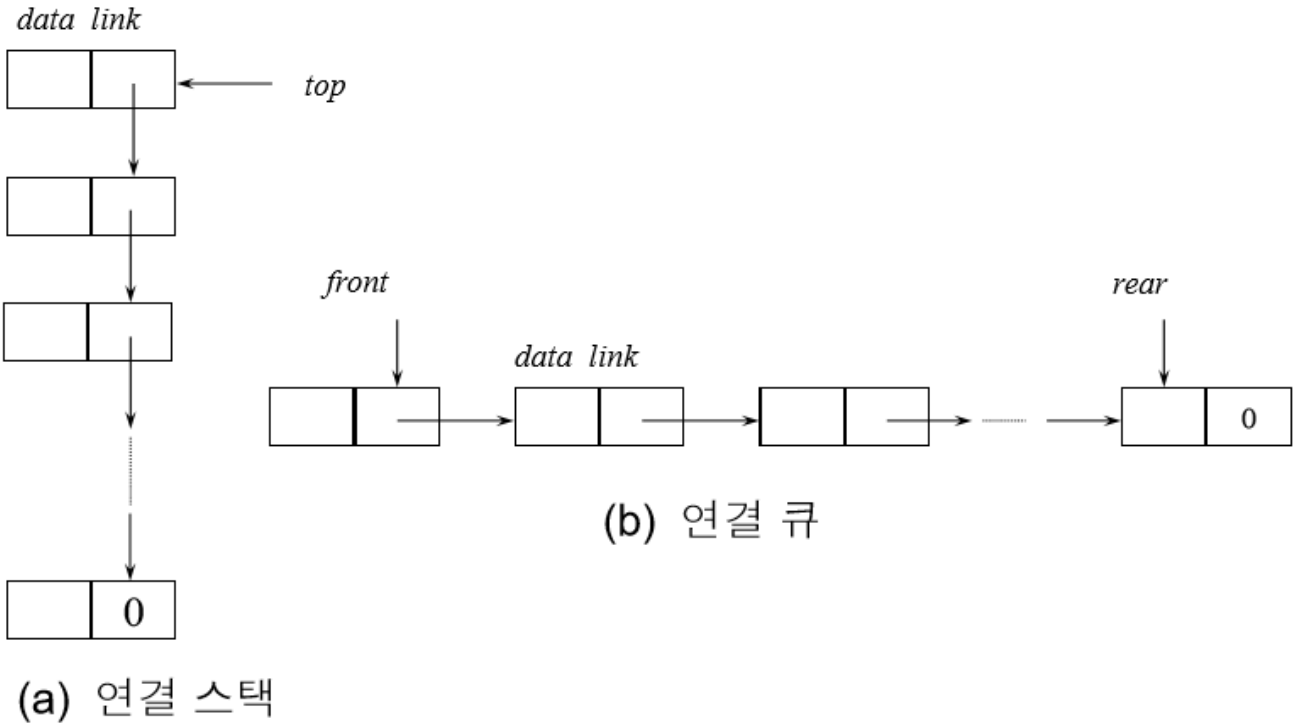
// 체인 $x = (a_1, ..., a_n)$이 $x = (a_n, ..., a_1)$로 변환된다.

```
template <class Type>
 void List<Type>::Invert() {
       ListNode<Type> *p = head, *q = 0, *r;    //
       while(p) {
          r = q;
          q = p;              // r은 q를 따라간다.
          p = p->link;    // p가 다음 노드로 옮겨 간다.
          q->link = r;    // q에 이전 노드를 연결한다.
       }
       head = q;
 }
```

2)List 합치기 (List Concatenation)

```
template <class Type>
void List<Type>:: concatenate(List<Type> ptr2) {
{ // 리스트 ptr1 뒤에 리스트 ptr2가 접합된 새 리스트를 생성한다.

   if (IS_empty(ptr1)    return ptr2;
   else {
        if (!IS_EMPTY(ptr2)) {
             for (temp = ptr1; temp->link; temp = temp->link)
               ;
             temp->link = ptr2;
        }
        return ptr1;
    }
}
```

1.2 Linked Stacks and Queues



(a) 연결 스택

(b) 연결 큐

◆생성자

공백 Stack : top/head 을 0으로 초기화

공백 Queue : front를 0으로 초기화

1) Linked List **STACK**

- **Class** 선언

```
class Node {
    private:
        int data;
        Node *next;
        Node(int value)
          {data = value;
           next = 0;}
        friend class linkedStack;
};

  class linkedStack {
      private:
         Node *head;
      public:
        linkedStack ()
              {head = 0;}
        ~linkedStack();
        void createStack();
        void push(int);
        int pop();
        int isEmpty();
        void displayStack();
        void searchStack(int);
   };
```

- Stack Create 함수

```
void linkedStack::createStack()
 {
       head = 0;
 }
```

* PUSH 함수

```
void linkedStack::push(int data){

   Node *temp = new Node(data);
   if (head == 0)
            head = temp;
   else {
            temp->next = head;
            head = temp;
     }
}
```

* POP 함수

```
int linkedStack::pop() {
      Node *p;
      int num;

   num= head->data;
   p = head;
   head = head->next;
   delete p;
   return num;
}
```

- STACK-EMPTY 함수

```
int linkedStack::isEmpty() {

  if (head == 0) return 1;
  else return 0;
}
```

```
  void linkedStack::searchStack(int num) {
  Node *p;

  if (head != 0) {
     p = head;
     while (p != 0 && p->data != num)      p = p->next;

       if (p)    cout << p->data << " is in the list.";
       else cout << num << " is not in the List.";
  }
  else      cout << "Stack is empty\n";
}
```

```
void linkedStack::displayStack() {
        Node *p;

    if (!isEmpty()) {
       p = head;
       while (p) {
          cout << p->data;    p = p->next;}
    else
         cout << "Stack empty";
    }
```

* Linked Stack sample Code

```
/*****************************/
/* <Stacks with Linked Lists        */
/*****************************/
#include <iostream>
#include <iomanip>

class Node {
   private:
      ……….
};

class linkedStack {
   private:
       Node *head;
   public:
      ……..
};

linkedStack::~linkedStack()
{   Node *p;
   ……..
}

void linkedStack::createStack()
{     head = 0;     }

void linkedStack::push(int data) {
  Node *temp = new Node(data);
   ……..
}

int linkedStack::pop() {
   Node *p;    int num;
   ……
   delete p; return num; }

void LinkedStack::displayStack()
{     Node *p;
     …….
  }

int linkedStack::isEmpty()
{   …….     }

void linkedStack::searchStack(int num)
{   Node *p;
…….
}


void main() {
   linkedStack s1;
   char input[10]; int num; int stopflag = 1;

   while (stopflag) {
      cout << "Command : push, pop, display,
          search, quit => ";
      cin >> input;

      if (strcmp(input,"push")==0) {
         cout<<"Input a number => ";
          cin >> num;    s1.push(num);
      }
       else if (strcmp(input, "pop") == 0) {
          if (!s1.isEmpty()) {
              num = s1.pop();
              cout << num << " has been
                    popped" << endl;
          }
          else   cout << "Stack is empty!\n";
        }
      else if (strcmp(input,"search")==0){
        cout << "Enter a number => ";
        cin >> num;
         s1.searchStack(num);
        }
      else if(strcmp(input,"display") == 0)
              s1.displayStack();

      else if (strcmp(input, "quit") == 0)
              stopflag = 0;
      else
           cout <<"Bad command"<< endl;}
       }
     }
```

2) Linked List Queue

- Class 선언

```
class Node {
   private:
      int data;
      Node *next;
      Node(int value) {data = value; next = 0;}
   friend class linkedQueue;
};

class linkedQueue {
    private:
      Node *front;
      Node *rear;

   public:
      linkedQueue () {front = 0; rear = 0;}
      ~linkedQueue();
      void createQueue();
      void enqueue(int);
      int   dequeue();
      int   isEmpty();

      void displayQueue();
      void searchQueue(int);
};
```

* Queue-empty 함수

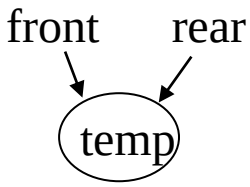
```
int linkedQueue::isEmpty()
{
   if (front == 0)        return 1;
   else                   return 0;
 }
```

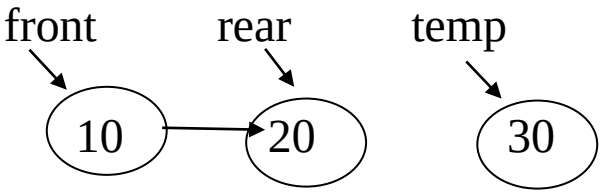
- create 함수

```
 void linkedQueue::createQueue()
 {
    front = 0;      rear = 0;
 }
```

- Enqueue 함수

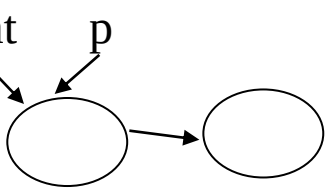
```
 void linkedQueue::enqueue(int data)
 {
   Node *temp = new Node(data);

   if (front == 0) {    /* 큐가 empty 인경우
      front = temp;
      rear = temp;
   }
   else {
      rear->next = temp;
      rear = temp;
   }
}
```





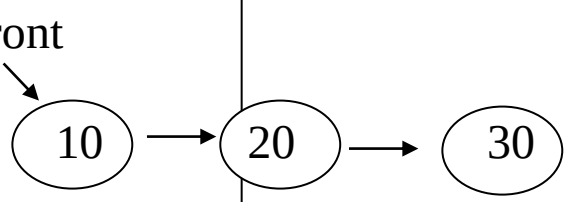
- dequeue 함수

```
int linkedQueue::dequeue()
{
   Node *p;      int    num;

   num = front->data;
   p = front;

   if (front == rear)    { front = 0;      rear = 0; }
   else              front = front->next;

   delete p;
   return num;
}
```



- Display-Queue 함수

```
void linkedQueue::displayQueue()
{
   Node *p;

   if (!isEmpty()) {
      p = front;
      while (p) {
           cout << setw(8) << p->data;
           p = p->next;
      }
      cout << endl;
   }
   else
        cout << "Queue is empty!\n";
}
```



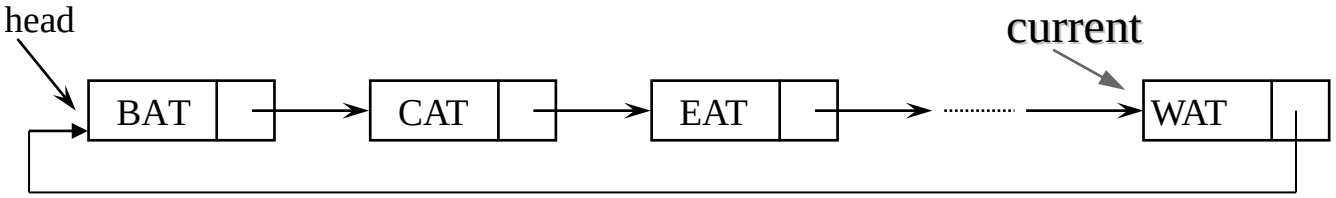
// Linked Queue Sample Code

```
//*****************************/
#include <iostream>
#include <iomanip>
using namespace std;

class Node {
   private: int data; Node *next;
          Node(int value)
             {data = value; next = 0;}
   friend class linkedQueue;
};

class linkedQueue {
   private: Node *front; Node *rear;
   public:
    linkedQueue ()
         {front = 0; rear = 0;}
    ~linkedQueue();
    void createQueue();
    void enqueue(int); int    dequeue();
    int isEmpty();
    void displayQueue();
};
linkedQueue::~linkedQueue(){
       Node *p;
       while (front != 0) {
        p = front;
        front = front->next;
        delete p;
      } }

  void linkedQueue::createQueue(){
       front = 0; rear = 0; }

void linkedQueue::enqueue(int data){
       Node *temp = new Node(data);
       if (front == 0) {
        front = temp; rear = temp;
       }
       else {
        rear->next = temp;
        rear = temp;
      } }
  int linkedQueue::dequeue(){
     Node *p;      int num;
     num = front->data; p = front;
     if (front == rear) {
        front = 0;    rear = 0;    }
     else
        front = front->next; delete p;
     return num;
   }

  int linkedQueue::isEmpty(){
   if (front == 0)      return 1;
   else      return 0; }

void linkedQueue::displayQueue() {
  Node *p;
  if (!isEmpty()) {
     p = front;
     while (p) {
        cout<< setw(8) << p->data;
        p = p->next;    }
     cout << endl;
  }
  else cout << "Queue is empty!\n";
}

void main() {
   linkedQueue s1; char input[10]; int num;
   int stopflag = 1;

  while (stopflag) {
       cout << "enqueue, dequeue, display, quit => ";
        cin >> input;

     if(strcmp(input,"enqueue")==0){
        cout << "Input a number => ";
        cin >> num; s1.enqueue(num);
       }
     else if(strcmp(input,"dequeue")==0){
          if (!s1.isEmpty()) {
             num = s1.dequeue();
             cout <<num << " has been deleted" << endl;
           }
           else    cout << "Queue is empty!\n";
         }
     else if(strcmp(input,"display")== 0)
           s1.displayQueue();
     else if(strcmp(input,"quit")==0)
           stopflag = 0;
     else
         cout << "Bad command" << endl;
      }
}
```

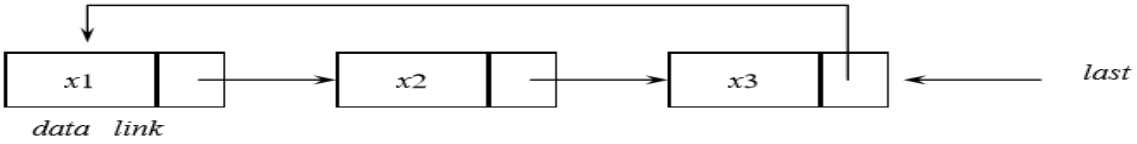
2. Circularly Linked List (CLL)

◆ **원형 리스트(circular list)**

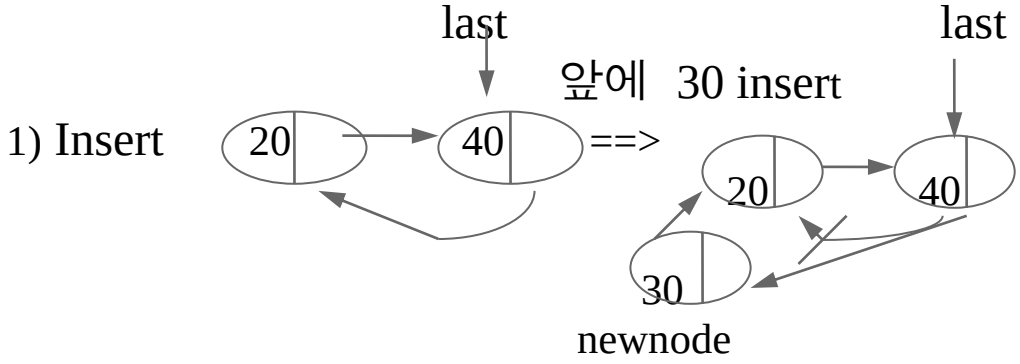
- 연결 리스트(체인)에서 마지막 노드의 link 필드가 첫 번째 노드를 가리킴 ex) current->link == head
- Head Node 필요, . No NIL . I/O 시 Buffer 에 이용



* 원형 리스트 접근 포인터가 마지막 노드를 가리키면 편리

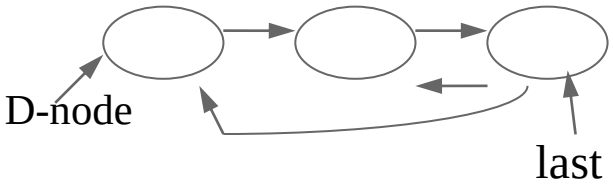


- 원형 연결 리스트 연산




```
template <class Type>
 void CircList::InsertFront(ListNode <Type> *newnode)
 // newnode 가 가리키는 노드를 원형 리스트의 앞에
    삽입한다.  last 는 리스트의 마지막 노드를 가리킨다.
 {
     if(!last) {
        last = newnode;      newnode->link = newnode; //empty node
     }
     else {
        newnode->link = last->link;      last->link = newnode;
     }}
   last=newnode;
 }
```

2) Delete




```
if (last ==NULL)
        {list_empty()}
else{
        D-node = last->link;
        last->link = D-node->link
        delete D-node;
}
```

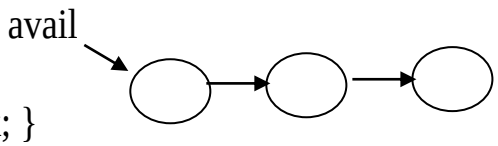
3) length

```
int length()   {   /* 원형 리스트 ptr의 길이를 계산한다. */
    int count = 0;
     if (last) {
          temp = last;
           do {
                  count++;
                  temp = temp->link;
            } while (temp != last);
       }
       return count;
    }
```

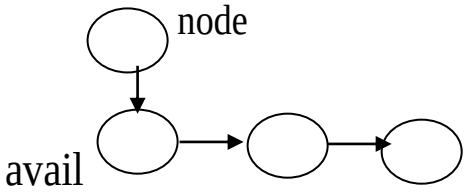
< Available space list(빈 공간리스트)의 관리>

- 필요가 없어진 리스트를 반환하는 경우 노드를 하나씩 반환하면 시간이 낭비됨. 쓰지 않는 노드의 리스트를 별도로 관리하면 효율성을 높일 수 있음.

```
template <class Type>
ListNode<Type>* CircList:: get_node( ) {    //* 사용할 노드를 제공 */
    ListNode<Type>* node;

     if (avail) {    node = avail;    avail = avail->link; }
     else {
              node = new ListNode<Type>;   //노드 없을 때 새로 생성
     }
     return node;
}
```




```
template <class Type>       // 가용 리스트에 노드 반환 */
void CircList<Type>::return_node(ListNode<Type>* node)
{
   node->link = avail;
    avail= node ;
 }
```




```
template <class Type>
void CircList<Type> :: cerase( ) {    /* 원형리스트 전체 제거 */
   if (ptr)    {
        temp = ptr->link;      (1)          ptr->link = avail;      (2)
        avail = temp;          (3)           ptr    = NULL;}
   }
```