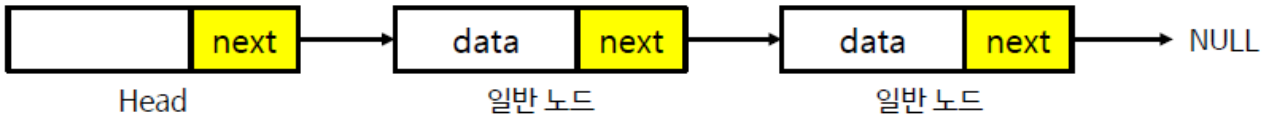
Linked list (단방향, 양방향)

연결 리스트

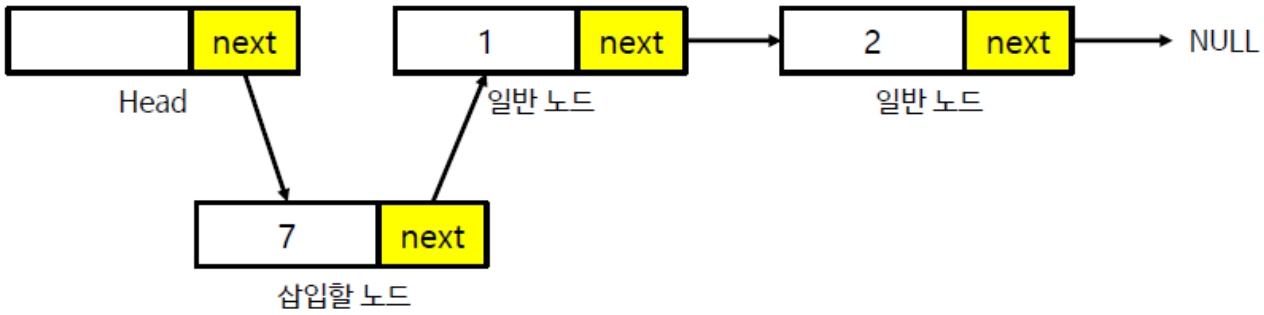
1) 단일 연결 리스트는 다음과 같은 형태로 나타낼 수 있습니다.
2) 포인터를 이용해 단방향적으로 다음 노드를 가리킵니다.
3) 일반적으로 연결 리스트의 시작 노드를 헤드(Head)라고 하며 별도로 관리합니다.
4) 다음 노드가 없는 끝 노드의 다음 위치 값으로는 NULL을 넣습니다.




```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

typedef struct _Node{
        struct _Node* next;
        int data;
} Node;

int main()
{
        Node* pHead = (Node*)malloc(sizeof(Node));

        Node* pNode1 = (Node*)malloc(sizeof(Node));
        pNode1->data = 1;

        Node* pNode2 = (Node*)malloc(sizeof(Node));
        pNode2->data = 2;
```

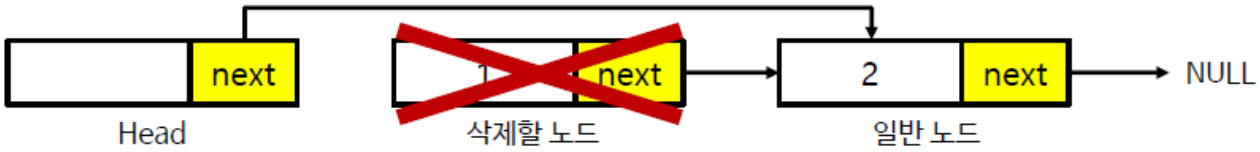
```
        pHead->next = pNode1;
        pNode1->next = pNode2;
        pNode2->next = NULL;

        Node* pCur = pHead->next;
        while (pCur != NULL)
        {
                printf("%d ", pCur->data);
                pCur = pCur->next;
        }

        system("pause");
        return 0;
}
```

연결 리스트 삽입 과정 3




```
void AddFront(Node* root, int nData)
{
        Node* pNode = (Node*)malloc(sizeof(Node));
        pNode->data = nData;
        pNode->next = root->next;
        root->next = pNode;
}
```

연결 리스트 삭제 과정 3




```
void RemoveFront(Node* pRoot)
{
	Node* pFrontNode = pRoot->next;
	pRoot->next = pFrontNode->next;

	free(pFrontNode);
}
```

연결 리스트 전체 삭제

```
void FreeAll(Node* pRoot)
{
	Node* pCur = pRoot->next;
	while (pCur != NULL)
	{
		Node* pNext = pCur->next;
		free(pCur);
		pCur = pNext;
	}
	free(pRoot);
}
```

연결 리스트 전체 출력

```
void ShowAll(Node* pRoot)
{
```

```
        Node* pCur = pRoot->next;
        while (pCur != NULL)
        {
                printf("%d \n", pCur->data);
                pCur = pCur->next;
        }

}
```

완성

```
int main()
{
        Node* pHead = (Node*)malloc(sizeof(Node));
        pHead->next = NULL;

        printf("--------------------\n--------------------
\n");
        AddFront(pHead, 1);
        ShowAll(pHead);
        printf("--------------------\n--------------------
\n");
        AddFront(pHead, 2);
        ShowAll(pHead);
        printf("--------------------\n--------------------
\n");
        AddFront(pHead, 4);
        ShowAll(pHead);
        printf("--------------------\n--------------------
\n");
        RemoveFront(pHead);
        ShowAll(pHead);
```

```
        printf("--------------------\n--------------------
\n");
        AddFront(pHead, 3);
        ShowAll(pHead);
        printf("--------------------\n--------------------
\n");
        AddFront(pHead, 4);
        ShowAll(pHead);
        printf("--------------------\n--------------------
\n");
        AddFront(pHead, 5);
        ShowAll(pHead);
        printf("--------------------\n--------------------
\n");

        FreeAll(pHead);
        system("pause");
        return 0;
}
```

연결 리스트 구현에 있어서 주의할 점

1) 위 소스코드에 덧붙여서 삽입 및 삭제 기능에서의 예외 사항을 처리할 필요가 있습니다.
2) 삭제할 원소가 없는데 삭제하는 경우, 머리(Head) 노드 자체를 잘못 넣은 경우 등을 체크해야 합니다.

연결 리스트의 특징

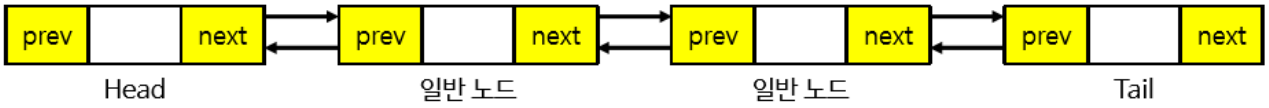
1) 삽입과 삭제가 배열에 비해서 간단하다는 장점이 있습니다.
2) 배열과 다르게 특정 인덱스로 즉시 접근하지 못하며, 원소를 차례대로 검색해야 합니다.
3) 추가적인 포인터 변수가 사용되므로 메모리 공간이 낭비됩니다.

연결 리스트

1) 연결 리스트는 데이터를 선형적으로 저장하고 처리하는 한 방법입니다.
2) 기존에 배열을 이용했을 때보다 삽입과 삭제가 많은 경우에서 효율적입니다.
3) 다만 특정한 인덱스에 바로 참조해야 할 때가 많다면 배열을 이용하는 것이 효율적입니다.

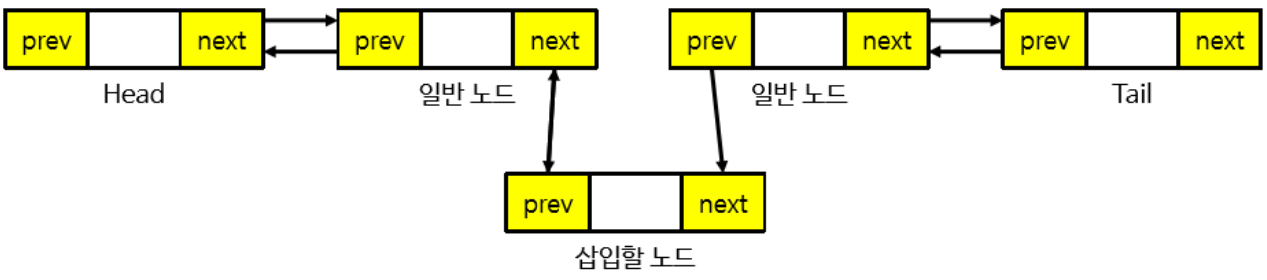
양방향 연결 리스트

1) 양방향 연결 리스트는 머리(Head)와 꼬리(Tail)를 모두 가진다는 특징이 있습니다.
2) 양방향 연결 리스트의 각 노드는 앞 노드와 뒤 노드의 정보를 모두 저장하고 있습니다.
3) 우리는 데이터를 '오름차순'으로 저장하는 양방향 연결 리스트를 구현해 볼 것입니다.




```
typedef struct _Node{
        struct _Node* prew;
        int data;
        struct _Node* next;
} Node;
```

연결 리스트 삽입 과정 4

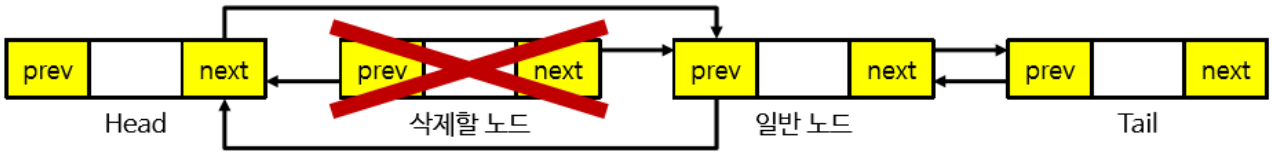



```
void Add_ASC(Node* pHead, int nData, Node* pTail)
{
```

```
        Node* pNode = (Node*)malloc(sizeof(Node));
        pNode->prev = NULL;
        pNode->data = nData;
        pNode->next = NULL;

        Node* pCur = pHead->next;
        while (NULL != pCur && pCur != pTail)
        {
                if (nData < pCur->data)
                        break;

                pCur = pCur->next;
        }
        Node* pPreNode = pCur->prev;
        pPreNode->next = pNode;
        pNode->prev = pPreNode;

        pNode->next = pCur;
        pCur->prev = pNode;
}

void Add_DESC(Node* pHead, int nData, Node* pTail)
{
        Node* pNode = (Node*)malloc(sizeof(Node));
        pNode->prev = NULL;
        pNode->data = nData;
        pNode->next = NULL;

        Node* pCur = pHead->next;
        while (NULL != pCur && pCur != pTail)
```

```
        {
                if (nData > pCur->data)
                        break;

                pCur = pCur->next;
        }
        Node* pPreNode = pCur->prev;
        pPreNode->next = pNode;
        pNode->prev = pPreNode;

        pNode->next = pCur;
        pCur->prev = pNode;
}
```

연결 리스트 삭제 과정 4




```
void RemoveFront(Node* pHead)
{
        Node* pFrontNode = pHead->next;
        Node* pRearNode = pFrontNode->next;

        pHead->next = pRearNode;
        pRearNode->prev = pHead;

        free(pFrontNode);
}
```

```
void RemvoeRear(Node* pTail)
{
        Node* pRearNode  = pTail->prev;
        Node* pFrontNode = pRearNode->prev;

        pFrontNode->next = pTail;
        pTail->prev = pFrontNode;

        free(pRearNode);
}
```

양방향 연결 리스트 구현에 있어서 주의할 점

1) 위 소스코드에 덧붙여서 삽입 및 삭제 기능에서의 예외 사항을 처리할 필요가 있습니다.
2) 더 이상 삭제할 원소가 없는데 삭제하는 경우 등을 체크해야 합니다.

양방향 연결 리스트

1) 양방향 연결 리스트에서는 각 노드가 앞 노드와 뒤 노드의 정보를 저장하고 있습니다.
2) 양방향 연결 리스트를 이용하면 리스트의 앞에서부터 혹은 뒤에서부터 모두 접근할 수 있습니다.

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

typedef struct _Node{
        struct _Node* prev;
        int data;
        struct _Node* next;
} Node;
```

```
void Add_ASC(Node* pHead, int nData, Node* pTail)
{
	Node* pNode = (Node*)malloc(sizeof(Node));
	pNode->prev = NULL;
	pNode->data = nData;
	pNode->next = NULL;

	Node* pCur = pHead->next;
	while (NULL != pCur && pCur != pTail)
	{
		if (nData < pCur->data)
			break;

		pCur = pCur->next;
	}
	Node* pPreNode = pCur->prev;
	pPreNode->next = pNode;
	pNode->prev = pPreNode;

	pNode->next = pCur;
	pCur->prev = pNode;
}

void Add_DESC(Node* pHead, int nData, Node* pTail)
{
	Node* pNode = (Node*)malloc(sizeof(Node));
	pNode->prev = NULL;
	pNode->data = nData;
	pNode->next = NULL;
```

```
        Node* pCur = pHead->next;
        while (NULL != pCur && pCur != pTail)
        {
                if (nData > pCur->data)
                        break;

                pCur = pCur->next;
        }
        Node* pPreNode = pCur->prev;
        pPreNode->next = pNode;
        pNode->prev = pPreNode;

        pNode->next = pCur;
        pCur->prev = pNode;
}

void RemoveFront(Node* pHead)
{
        Node* pFrontNode = pHead->next;
        Node* pRearNode = pFrontNode->next;

        pHead->next = pRearNode;
        pRearNode->prev = pHead;

        free(pFrontNode);
}

void RemvoeRear(Node* pTail)
{
```

```
        Node* pRearNode  = pTail->prev;
        Node* pFrontNode = pRearNode->prev;

        pFrontNode->next = pTail;
        pTail->prev = pFrontNode;

        free(pRearNode);
}

void ShowAll(Node* pHead, Node* pTail)
{
        Node* pCur = pHead->next;
        while (pCur != NULL && pCur != pTail)
        {
                printf("%d \n", pCur->data);
                pCur = pCur->next;
        }

}

void FreeAll(Node* pHead)
{
        Node* pCur = pHead->next;
        while (pCur != NULL)
        {
                Node* pNext = pCur->next;
                free(pCur);
                pCur = pNext;
        }
        free(pHead);
```

```
}

int main()
{
        Node *pHead = NULL, *pTail = NULL;
        pHead = (Node*)malloc(sizeof(Node));
        pHead->prev = NULL;
        pHead->next = NULL;
        pTail = (Node*)malloc(sizeof(Node));
        pTail->prev = NULL;
        pTail->next = NULL;

        pHead->next = pTail;
        pTail->prev = pHead;

        Add_ASC(pHead, 1, pTail);
        ShowAll(pHead, pTail);
        printf("--------------------\n--------------------
\n");
        Add_ASC(pHead, 3, pTail);
        ShowAll(pHead, pTail);
        printf("--------------------\n--------------------
\n");
        Add_ASC(pHead, 2, pTail);
        ShowAll(pHead, pTail);
        printf("--------------------\n--------------------
\n");
        printf("--------------------\n--------------------
\n");
        RemoveFront(pHead);
        ShowAll(pHead, pTail);
```

```
        printf("--------------------\n--------------------
\n");
        RemoveFront(pHead);
        ShowAll(pHead, pTail);
        printf("--------------------\n--------------------
\n");
        RemoveFront(pHead);
        ShowAll(pHead, pTail);
        printf("--------------------\n--------------------
\n");
        FreeAll(pHead);

        printf("--------------------\n--------------------
\n");
        printf("--------------------\n--------------------
\n");
        printf("--------------------\n--------------------
\n");
        printf("--------------------\n--------------------
\n");
        printf("--------------------\n--------------------
\n");

        pHead = (Node*)malloc(sizeof(Node));
        pHead->prev = NULL;
        pHead->next = NULL;
        pTail = (Node*)malloc(sizeof(Node));
        pTail->prev = NULL;
        pTail->next = NULL;

        pHead->next = pTail;
        pTail->prev = pHead;
```

```
        Add_DESC(pHead, 1, pTail);
        ShowAll(pHead, pTail);
        printf("--------------------\n--------------------
\n");
        Add_DESC(pHead, 3, pTail);
        ShowAll(pHead, pTail);
        printf("--------------------\n--------------------
\n");
        Add_DESC(pHead, 2, pTail);
        ShowAll(pHead, pTail);
        printf("--------------------\n--------------------
\n");
        printf("--------------------\n--------------------
\n");
        RemvoeRear(pTail);
        ShowAll(pHead, pTail);
        printf("--------------------\n--------------------
\n");
        RemvoeRear(pTail);
        ShowAll(pHead, pTail);
        printf("--------------------\n--------------------
\n");
        RemvoeRear(pTail);
        ShowAll(pHead, pTail);
        printf("--------------------\n--------------------
\n");
        FreeAll(pHead);

        system("pause");
        return 0;
```

```
}
```