<자료구조 4차 프로그래밍 보고서 – list>

201720736 이균

1. Data Structure

이 과제에서 사용된 자료구조는 list이다.
Linked list란 list의 끝 정보가 다음 링크를 가리키는 것을 의미하는데 다음과 같이 구조체 형식으로 표현된 리스트에서 구조체 포인터 link를 이용해 다음 리스트를 가리키면 linked list를 만들 수 있다.

```
typedef struct listNode* listPointer;
typedef struct listNode{
    int data;
    listPointer link;
};
```

struct의 data에는 입력되는 숫자가 저장될 수 있도록 int data;와 다음 리스트를 가리키도록 link를 선언하였다.

다음은 사용하지 않는 노드 관리를 위한 코드이다.

```
listPointer avail = NULL; /* a global variable that points to the first
node of the free nodes list */

listPointer getNode(void)
{
    listPointer node;
    if(avail){
        node = avail;
        avail = avail->link;
    }
    else
        MALLOC(node, sizeof(*node));
    return node;
}

void retNode(listPointer* node, listPointer prev)
{
    listPointer temp;
    temp = (*node)->link;
    if(avail){
        (*node)->link = avail;
        avail = *node;
    }
    else{
        avail = *node;
        avail->link = NULL;
    }

    if(prev)
        prev->link = temp;
    else
    {
        *node = temp;
    }
}
```

전역변수로 avail을 만들어 주었고 이는 사용하지 않는 노드의 첫번째를 나타낸다.
getNode를 통해 avail이 있으면 avail의 맨 앞을 get하고, 없으면 동적할당을 통해 노드를 만들어준다.
retNode는 데이터 delete시 그 리스트를 avail이 있을 경우 avail의 맨 앞으로 추가하고, avail이 없다면 그 리스트의 주소를 avail로 한다. 또한 삭제하는 데이터의 앞뒤로 링크를 연결해주기 위해서 prev가 있는 경우와 없는 경우를 나누어 연결시켜주고 있다.

과제의 linked list는 data가 추가될 때마다 리스트에 데이터의 크기 순으로 연결해줘야 한다. 따라서 data를 추가하는 함수를 다음과 같이 구현하였다.
`addData(listPointer* first, int* nump);`

이때 parameter로 리스트 첫 노드의 포인터의 포인터 first와 사용자로부터 입력 받은 *nump가 들어온다.
이때 크기비교 시 바뀌는 *first를 위해서 *first를 head로 바꿔주었으며, 크기 순으로 데이터를 집어넣어야 하기 때문에 데이터가 들어가야 할 위치인 `listPointer insertPtr`과 이전 위치인 `prev` 를 선언하였다.

크기 비교를 위해서는 head의 링크를 증가시켜가면서 어디에 데이터를 집어넣을지 확인하여 insrtptr을 결정한다. 다음은 크기 비교를 위한 코드이다.

```
while(head)    /* Check data and add according to the size */
    {
        if(head->data == *nump)
        {
            printf("중복된 데이터입니다.\n");
            printf("-------\n");
            return;
        }
        if(head->link)
        {
            if( head->data < *nump && head->link->data > *nump)
            {
                insertPtr = head;
                break;
            }
            else if(head->data > *nump)
            {
                insertPtr = prev;
                break;
            }
        }
        else
        {
            if( head->data < *nump)
            {
                insertPtr = head;
            }
            else{
                insertPtr = prev;
            }
            break;
        }
        prev = head;
        head = head->link;
    }
```

위의 코드에서 크기 비교를 해서 insrtPtr을 결정하고 있다.
우선 맨 처음 노드가 있을 경우 비교 대상이 있는 것 이기 때문에 while문을 통해서 반복하며 head노드와 prev를 하나씩 증가시킨다.
만약 데이터가 리스트에 이미 있는 데이터라면 중복된 데이터일 것이고, 만약 리스트에 단 하나만의 데이터만 있다면 그 데이터와 크기비교 후에 들어오는 데이터가 크면 뒤에 넣고, 작으면 앞에 넣어야 할 것이다.
반면 리스트가 두개 이상일 때 (if(head->link)) 헤드노드의 데이터와 들어오는 데이터를 비교해서 앞에 넣을지, 뒤에 넣을지 결정한다.

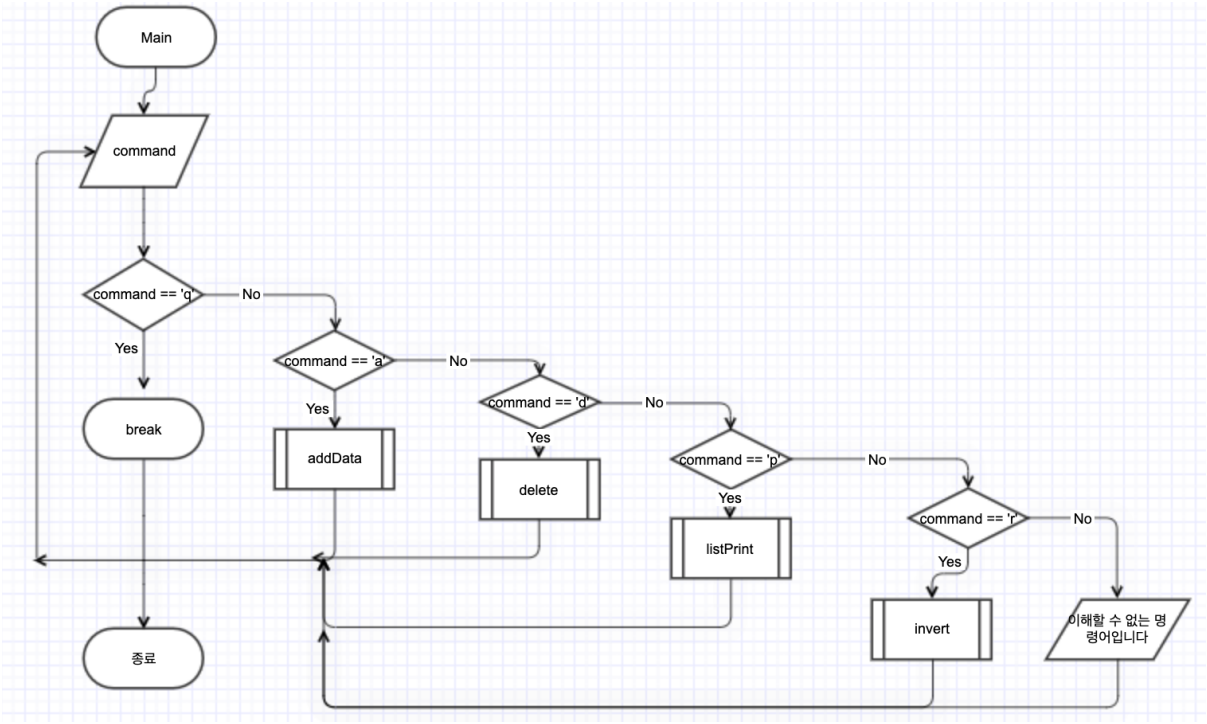
이때 데이터를 삭제하면 사용하지 않는 데이터 리스트를 관리하다가 데이터를 추가할 시에 그 관리 데이터를 가져와 다시 사용하는 코드를 위해서 getNode와 retNode 함수를 만들었기 때문에 data를 추가할 때 사용하지 않는 리스트의 첫 수소인 avail이 있는지 없는지에 따라서 구분될 것이다.
만약 avail이 있다면 getNode의 return값을 ptr로 대입해서 쓰지않는 노드를 가져온것인 ptr에 데이터를 넣으면 될것이다. 이때 크기 비교를 통해 insertPtr을 구했으므로 insertptr이 존재하면 그 뒤에 데이터를 넣으면 되고, 존재하지 않는다면 *first의 유무 즉 첫번째 노드의 유무에 따라 흐름을 분기해준다.

데이터 삭제 시에는 temp라는 포인터를 이용하여 temp를 반복시키면서 리스트에 찾는 값이 있다면 temp에 data를 넣고, 없다면 해당 데이터가 없다고 출력한다.
또한 retNode를 통해서 avail이 가리키는 리스트에 반환한다. 또한 리턴하면 그 전의 노드와 리턴하는 노드의 다음 노드를 이어줘야 하기 때문에 그 작업을 위해서 prev를 선언하였다.

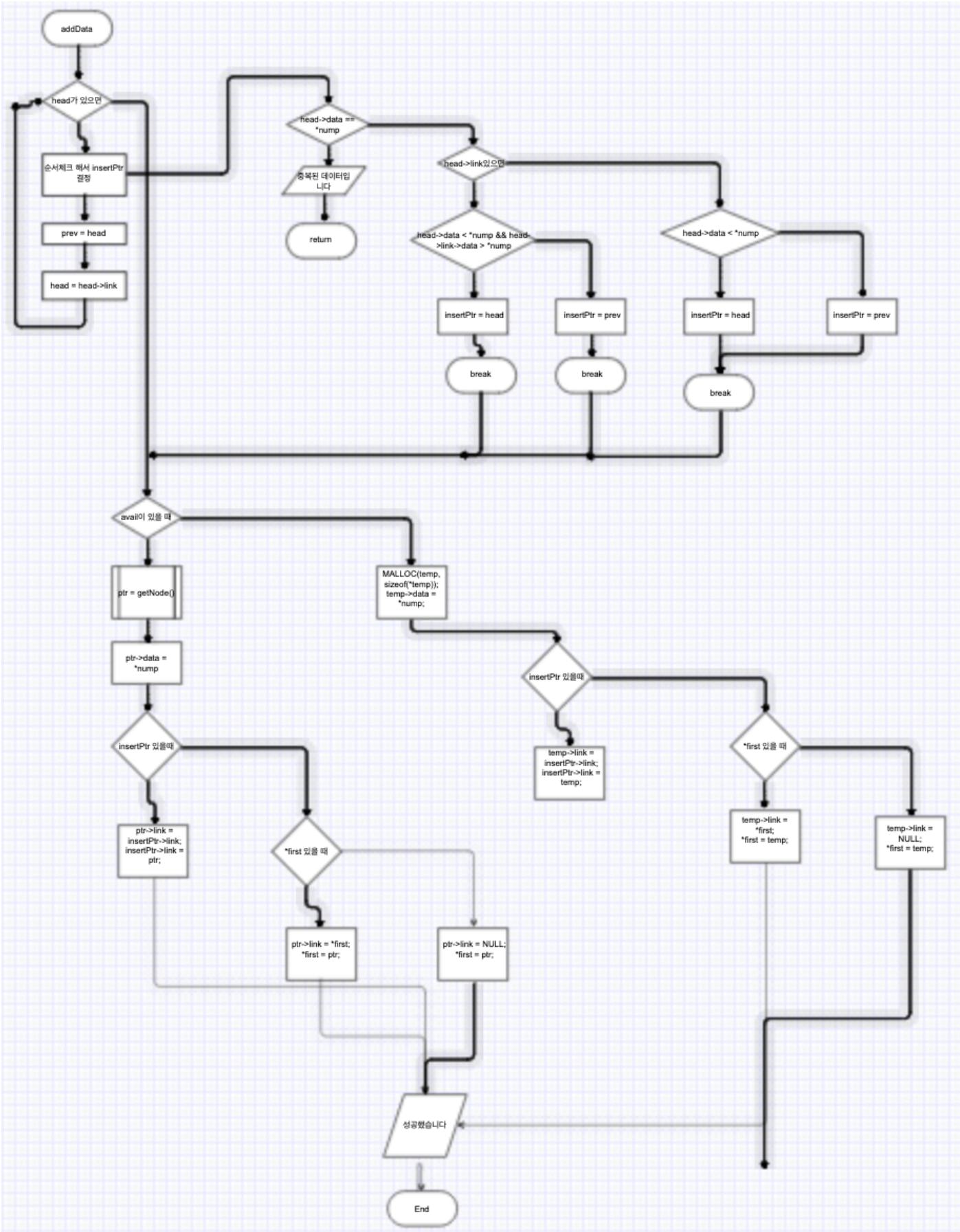
리스트를 출력 할 때에는 리스트가 없을 경우 리스트가 비어있다고 출력하면 되고, 만약 있다면 리스트의 데이터를 출력 한 뒤에 링크를 하나 증가시킨 후 다시 출력하게 반복하다가 링크라 NULL이 되면 루프를 빠져나오면 된다.

리스트를 반대로 출력 할 때에는 스택을 활용한다.
스택을 이용해서 리스트의 data를 차례대로 push하고, 스택에 있는 값들을 다시 차례대로 pop하면서 출력하면 반대로 출력될 것이다.

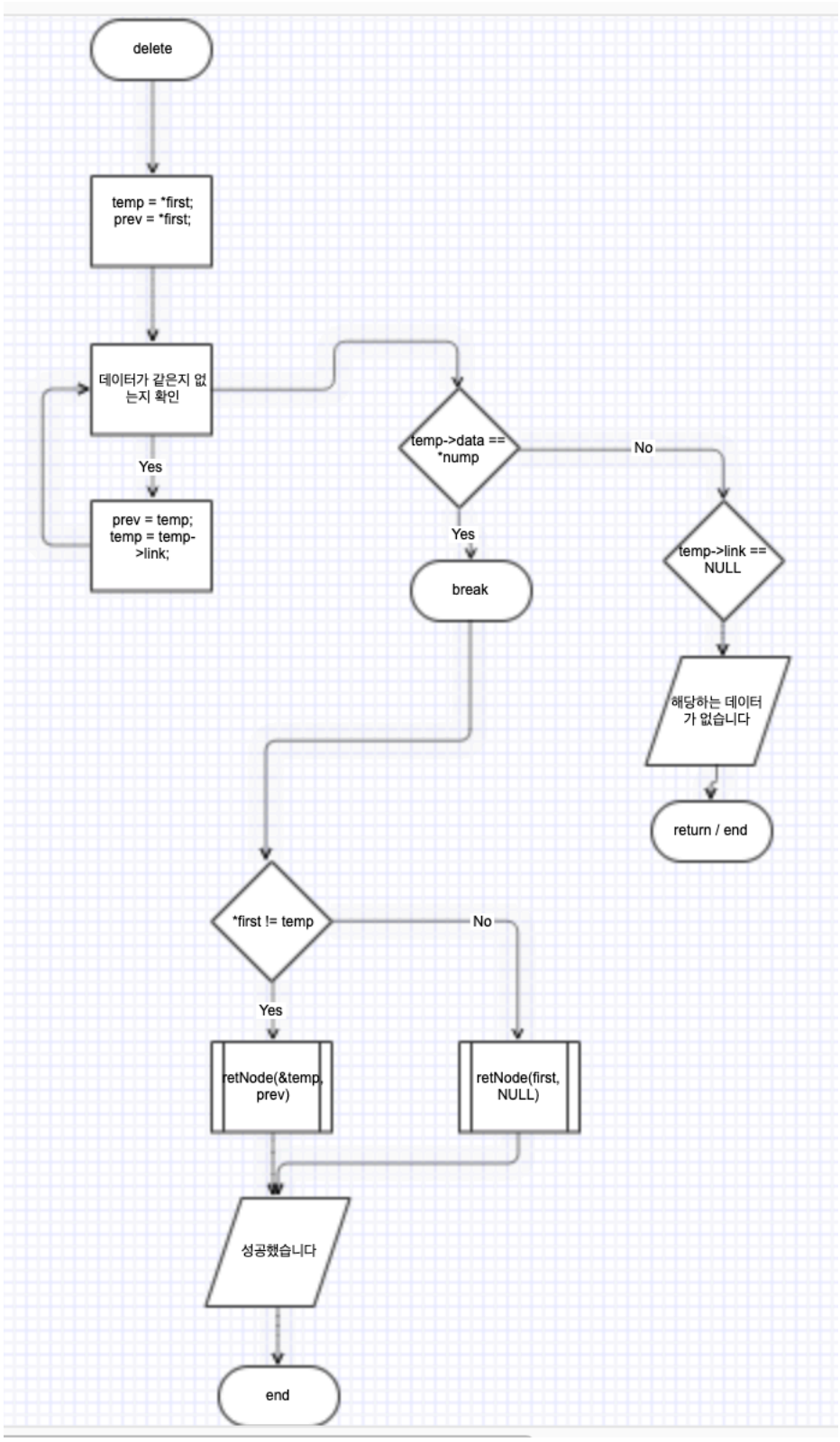
2. Flow Chart


Main
command
command == 'q'
No
Yes
break
종료
command == 'a'
No
Yes
addData
command == 'd'
No
Yes
delete
command == 'p'
No
Yes
listPrint
command == 'r'
No
Yes
invert
이해할 수 없는 명령어입니다

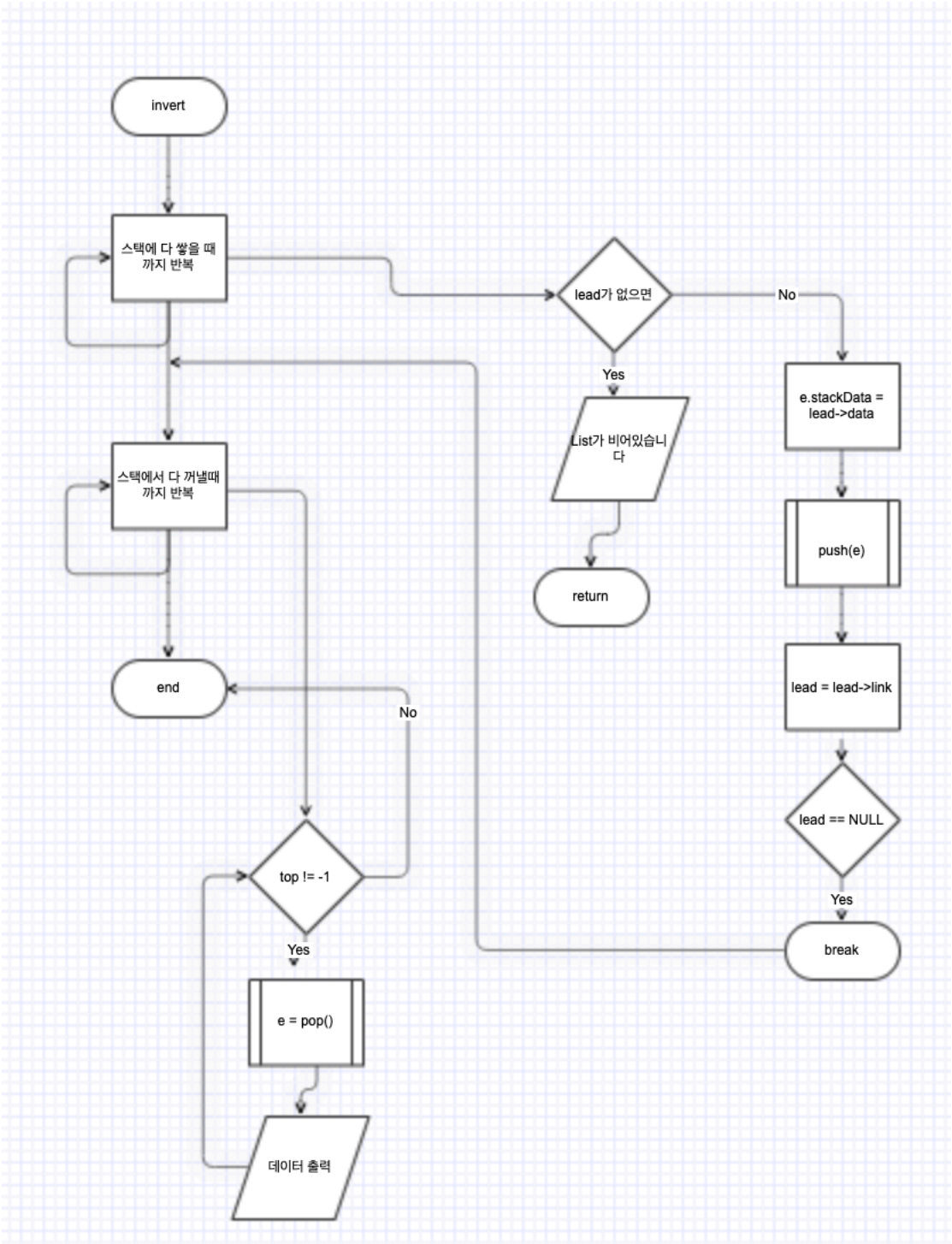
main 함수


addData
head가 있으면
순서체크 해서 insertPtr 결정
prev = head
head = head->link
head->data == *nump
중복된 데이터입니다
return
head->link있으면
head->data < *nump && head->link->data > *nump
head->data < *nump
insertPtr = head
insertPtr = prev
insertPtr = head
insertPtr = prev
break
break
break
avail이 있을 때
ptr = getNode()
MALLOC(temp, sizeof(*temp)); temp->data = *nump;
ptr->data = *nump
insertPtr 있을때
insertPtr 있을때
*first 있을 때
ptr->link = insertPtr->link; insertPtr->link = ptr;
*first 있을 때
temp->link = insertPtr->link; insertPtr->link = temp;
temp->link = *first; *first = temp;
temp->link = NULL; *first = temp;
ptr->link = *first; *first = ptr;
ptr->link = NULL; *first = ptr;
성공했습니다
End

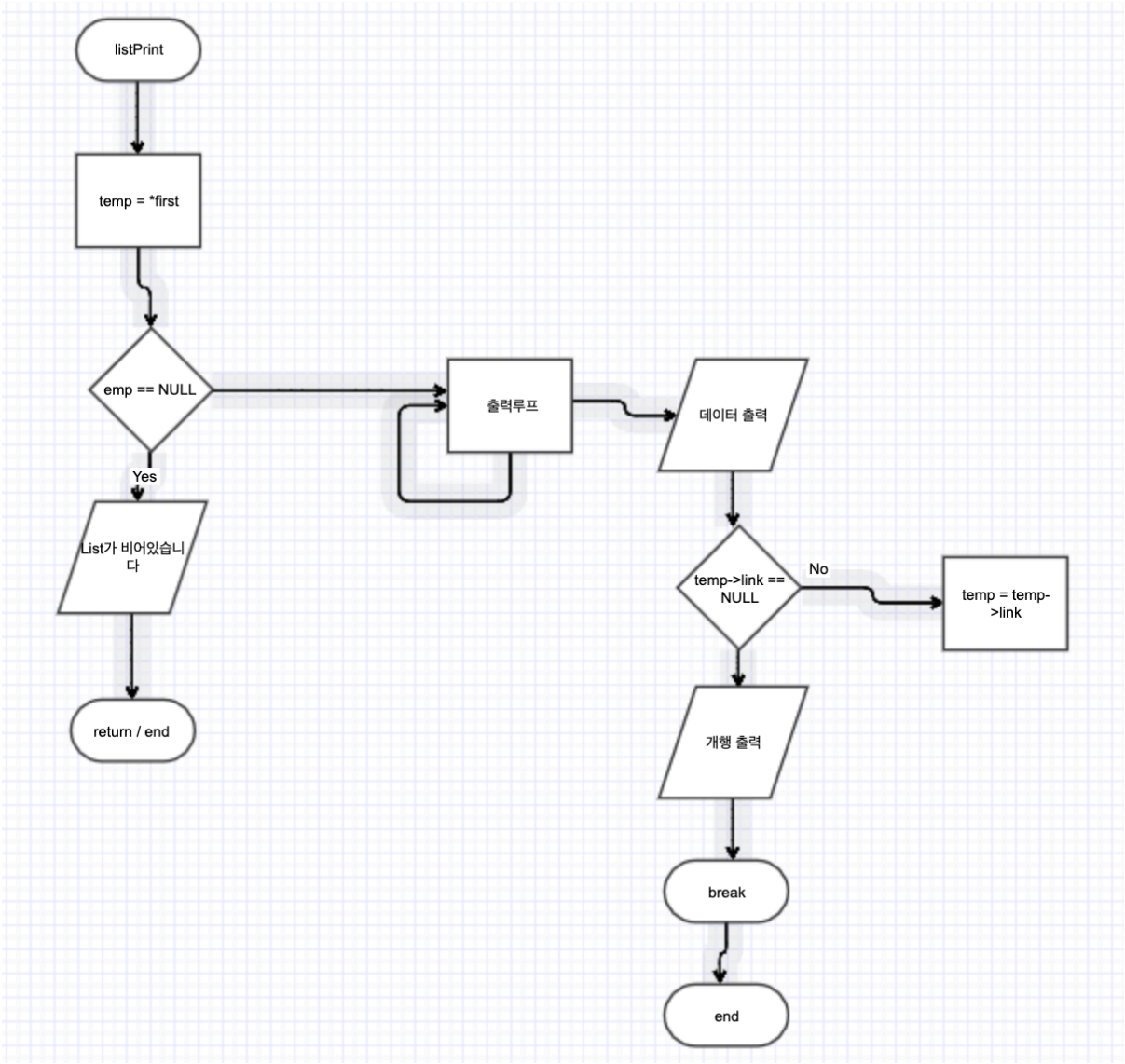
addData 함수


delete
temp = *first;
prev = *first;
데이터가 같은지 없는지 확인
Yes
prev = temp;
temp = temp->link;
temp->data == *nump
No
Yes
break
temp->link == NULL
해당하는 데이터가 없습니다
return / end
*first != temp
No
Yes
retNode(&temp, prev)
retNode(first, NULL)
성공했습니다
end

delete함수


invert
스택에 다 쌓을 때 까지 반복
lead가 없으면
No
Yes
List가 비어있습니다
return
e.stackData = lead->data
push(e)
lead = lead->link
lead == NULL
Yes
break
스택에서 다 꺼낼때 까지 반복
end
No
top != -1
Yes
e = pop()
데이터 출력

invert함수



listPrint 함수