

[AI.SW개론-F]

C언어 연결리스트 구현 문법

Chat GPT를 활용한 프로그래밍 기초 문법 습득

202578218 정지은 AISW계열

목차

연결 리스트 (Linked List) 구현	1
연결 리스트 개념	4
append	4
delete	4
printList	5
Chat GPT와의 대화	6
“이 코드의 전체 흐름을 c언어 초보자에게 설명해봐”	6
◆ 전체 흐름 개요	6
◆ 사용되는 구조체	6
◆ append 함수: 리스트에 데이터 추가	7
◆ delete 함수: 특정 데이터 삭제	7
◆ printList 함수: 리스트 출력	8
◆ 전체 흐름 예시	8
◆ 한 줄 요약	9
“이 코드의 결과를 시각화해서 보여줘”	9

연결 리스트 (Linked List) 구현

- append, delete, printList

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

```
#include <stdlib.h>
```

```
typedef struct Node {
```

```
    int data;
```

```
    struct Node* next;
```

```
} Node;
```

```
Node* head = NULL;
```

```
void append(int data);
```

```
void delete(int data);
```

```
void printList();
```

```
int main() {
```

```
    append(10);
```

```
    append(20);
```

```
    append(30);
```

```
    printList();
```

```
    delete(20);
```

```
    printList();
```

```
}
```

```
void append(int data) {
```

```
    Node* newNode = (Node*)malloc(sizeof(Node));
```

```
    newNode->data = data;
```

```
    newNode->next = NULL;
```

```
    if (head == NULL) {
```

```
        head = newNode;
```

```
        return;
```

```
    }
```

```
Node* temp = head;
```

```
while (temp->next != NULL) {
```

```
    temp = temp->next;
```

```
}
```

```
temp->next = newNode;
```

```
}
```

```

void delete(int data) {
    Node* temp = head;
    Node* prev = NULL;
    while (temp != NULL && temp->data != data) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        return;
    }
    if (prev == NULL) {
        head = temp->next;
    }
    else {
        prev->next = temp->next;
    }
    free(temp);
}

```

```

void printList() {
    Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

```

```

int main() {
    append(10);
    append(20);
    append(30);
    printList();
}

```

```

delete(20);
printList();
while (head != NULL) {
    Node* temp = head;
    head = head->next;
    free(temp);
}
return 0;
}

```

연결 리스트 개념

Linked List

데이터를 저장하는 노드들이 포인터를 통해 서로 연결되어 있는 동적 자료구조

- 동적 메모리 관리
- 삽입 및 삭제 효율성

append

: 연결 리스트의 끝에 새로운 노드를 동적으로 생성하여 추가하는 함수

매개변수 **data** : 새로 추가할 노드에 저장할 실제 값

1) **Node* newNode = (Node*)malloc(sizeof(Node));**

: 새로운 노드를 생성하기 위해 힙 메모리에 공간을 동적으로 할당

sizeof(Node) : 노드 하나의 크기만큼 메모리 요청

(Node*) : 반환된 **void*** 포인터를 **Node*** 타입으로 형변환

2) **newNode->data = data;**

: 생성된 노드에 전달받은 값 저장

3) **newNode->next = NULL;**

: 새 노드는 리스트의 마지막에 붙을 것이므로 아직은 다음 없다. 때문에 **NULL**을 설정

4) **if (head == NULL) {**

head = newNode;

return;

}

: 리스트가 비어있는 경우, 방금 만든 노드가 첫노드가 된다.

delete

: 연결 리스트에서 특정 데이터를 가진 첫번째 노드 하나를 찾아 삭제하는 함수

해당 노드 제거 후, 리스트의 연결이 유지되도록 이전 노드와 다음 노드를 다시 연결해야 한다.

- 1) `Node* temp = head;`
`temp` : 현재 탐색 중인 노드를 가리키는 포인터
- 2) `Node* prev = NULL;`
`prev` : `temp`보다 한 노드 앞에 있는 노드를 기억하기 위한 포인터
처음에는 아무 노드도 지나지 않았으므로 `NULL`로 초기화
- 3) `while (temp != NULL && temp->data != data) {`
 `prev = temp;`
 `temp = temp->next;`
 }
: 리스트를 따라가며 `temp`가 가리키는 노드의 `data`가 삭제하려는 값과 다를 때, 반복
`temp`가 삭제 대상 노드를 가리키게 만들고, `prev`가 그 앞 노드를 가리키도록 한다
- 4) `if (temp == NULL) {`
 `return;`
 }
: 삭제할 값을 찾지 못했을 경우, 함수 종료
- 5) `if (prev == NULL) {`
 `head = temp->next;`
 }
: 삭제할 노드가 리스트의 첫 노드일 경우, `head`를 삭제할 노드의 다음 노드로 바꿔서
리스트의 시작 지점을 한 칸 뒤로 옮겨 첫 노드 삭제
- 6) `else {`
 `prev->next = temp->next;`
 }
: 삭제할 노드가 첫 번째가 아닐 때, 이전 노드의 `next`를 현재 노드의 다음 노드로 연결
- 7) `free(temp);`
 }
메모리 해제
`malloc`으로 동적 할당했던 노드를 삭제함으로써 메모리 누수 방지

printList

: 현재 연결 리스트에 있는 모든 노드를 차례대로 순환하여 출력하는 함수

- 현재 리스트의 상태 확인/디버깅

- 1) `Node* temp = head;`

temp 포인터를 리스트의 시작점인 head로 초기화

이 포인터를 이용해 리스트를 한 칸씩 이동하면서 순회한다

```
2) while (temp != NULL) {  
    printf("%d -> ", temp->data);  
    temp = temp->next;  
}
```

Temp가 NULL이 아닐 때까지 반복

각 노드의 데이터를 printf로 출력하며 ->로 다음 노드와의 연결을 시각적으로 표현

temp = temp->next를 통해 다음 노드로 이동

Chat GPT와의 대화

“이 코드의 전체 흐름을 c언어 초보자에게 설명해봐”

지금 C언어 초보자 입장에서 연결리스트(**Linked List**)의 기본 구조를 이해하고자 하십니다.
아래에서는 지금까지 설명드린 **append**, **delete**, **printList** 함수를 중심으로 전체 흐름을
쉽게 풀어 설명드리겠습니다.

◆ 전체 흐름 개요

C언어에서 연결리스트는 **Node**라는 구조체를 사용해서 데이터와 포인터를 묶은 노드들이
하나씩 이어진 구조입니다. 지금까지 본 함수들은 다음 세 가지 기능을 담당합니다:

1. **append** → 새로운 데이터를 리스트 끝에 추가
 2. **delete** → 특정 데이터를 가진 노드를 찾아서 삭제
 3. **printList** → 현재 연결리스트의 내용을 화면에 출력
-

◆ 사용되는 구조체

c

코드 복사

```
typedef struct Node {  
    int data;           // 실제 저장할 값  
    struct Node* next; // 다음 노드를 가리키는 포인터
```

```
} Node;
```

```
Node* head = NULL;           // 연결리스트의 시작 지점을 가리키는 포인터
```

- **Node**는 하나의 노드를 정의합니다.
- **head**는 연결리스트의 첫 번째 노드를 가리키는 포인터입니다. 아무것도 없을 땐 **NULL**입니다.

◆ **append** 함수: 리스트에 데이터 추가

C

코드 복사

```
void append(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node)); // 새 노드 메모리 할당
    newNode->data = data;                         // 데이터 저장
    newNode->next = NULL;                         // 끝에 붙을 거니까
    다음은 NULL

    if (head == NULL) {
        head = newNode; // 리스트가 비어 있으면 새 노드를 첫 노드로
        return;
    }

    Node* temp = head; // 리스트 순회 시작
    while (temp->next != NULL) {
        temp = temp->next; // 마지막 노드까지 이동
    }
    temp->next = newNode; // 마지막 노드의 next를 새 노드로 연결
}
```

전체 기능: 데이터를 가진 새 노드를 만들어 리스트 끝에 붙입니다.

◆ **delete** 함수: 특정 데이터 삭제

c

코드 복사

```
void delete(int data) {
    Node* temp = head;
    Node* prev = NULL;

    while (temp != NULL && temp->data != data) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) return; // 못 찾으면 아무 일도 안 함

    if (prev == NULL) {
        head = temp->next;    // 삭제할 노드가 첫 번째 노드일 경우
    } else {
        prev->next = temp->next; // 이전 노드가 다음 노드를 건너뛰게 연결
    }

    free(temp); // 삭제한 노드 메모리 해제
}
```

전체 기능: 리스트에서 주어진 값을 가진 노드를 찾아 제거하고, 연결을 다시 잇습니다.

◆ printList 함수: 리스트 출력

c

코드 복사

```
void printList() {
    Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
```


전체 기능: 현재 연결리스트의 모든 값을 차례로 화면에 출력합니다.

예:

rust

코드 복사

```
10 -> 20 -> 30 -> NULL
```

◆ 전체 흐름 예시

c

코드 복사

```
append(10);  
append(20);  
append(30);  
printList();    // 10 -> 20 -> 30 -> NULL
```

```
delete(20);  
printList();    // 10 -> 30 -> NULL
```

```
delete(10);  
printList();    // 30 -> NULL
```

```
delete(30);  
printList();    // NULL
```

◆ 한 줄 요약

이 프로그램은 동적으로 데이터를 추가하고 삭제할 수 있는 연결리스트를 구현한 것으로, 각 노드는 다음 노드를 가리키는 방식으로 리스트를 구성합니다.

“이 코드의 결과를 시각화해서 보여줘”

Initial

