펭권 스터디 내용15

작성자 : 류성수
작성일자 : 24.07.04

url : https://www.youtube.com/watch?v=gHLxOJuElEw&list=PLz--ENLG_8TMdMJIwyqDIpcEOysvNoonf&index=17

● 연결리스트

1. 왜 쓰는가?

일단 연결리스트는 배열의 단점들을 극복하기 위해 만들어진 자료구조이다.
다양한 분야에서 사용되는데,

* 네트워크 : 소켓통신에서 전달된 패킷을 받아서 링큐에 담을 때, 이 링큐를 연결리스트 형식으로 구현한다
* 운영체제 : 연결 리스트는 운영 체제의 메모리 할당과 해제를 관리하는 데 사용된다. 예를 들어, 자유 리스트(free list)는 가용한 메모리 블록을 추적하는 데 연결 리스트를 사용한다.
* 데이터베이스 : 인덱스를 관리하기 위해 연결 리스트가 사용된다. B-트리나 B+트리와 같은 인덱스 구조는 연결 리스트를 사용하여 노드를 관리한다.
* 음악 및 미디어 플레이어 : 음악 및 비디오 플레이어는 재생 목록을 관리하기 위해 연결 리스트를 사용할 수 있다. 각 노드는 재생할 미디어 파일을 나타내며, 연결 리스트를 통해 순차적으로 파일을 재생한다.

등등 다양한 분야에서 연결리스트를 사용한다.

● 간단히 구현

연결리스트의 노드들을 구현하였다.

```
typedef struct Node
{
        int id;
        char name[30];
        struct Node * next;
}Node;

Node* head;
Node* tail;
```

이 노드를 이용해서, 연결리스트의 기능들을 간단히 구현하였다.

```
int insert(int id, char * name)
{
        Node* data = (Node*)malloc(sizeof(Node));
        data->next = NULL;

        data->id = id;
        if(name!=NULL )
                strcpy(data->name,name);

        if(head == NULL) // 처음이냐?
        {
                head = data; // 네
                tail = data;
                return 1;
        }
        else // 아니요
        {
                Node* temp = head;

                while(temp->next)
                        temp = temp->next;

                temp->next = data;
                tail = data;
        }
        return 1;
}
```

위의 코드는 연결리스트에 노드를 하나씩 붙이는 코드이다.
tail 부분에 추가하고, 끝의 노드의 next 구조체 포인터에 생성한 노드를 붙인다.

```
void print()
{
        if(head == NULL)
                return;

        Node* temp = head;
        while(temp->next)
        {
                printf("\nid : %d, name : %s\n",temp->id,temp->name);
                temp = temp->next;
        }

        printf("\nid : %d, name : %s\n",temp->id,temp->name);
        // 마지막 노드는 무조건 next 멤버가 null 이기에
}
```

연결리스트 전체를 머리부터 꼬리까지 전부 출력하는 함수이다.

```
void stack_pop(Node* basket) // 제일 마지막게 pop됨
{
        if(head == NULL)
                return;

        // 가장 끝 노드 이전 노드를 찾기위한 여정
        Node* temp = head;
        Node* before = NULL;

        while(temp->next)
        {
                before = temp;
                temp = temp->next;
        }

        basket->id = tail->id;
        strcpy(basket->name,tail->name); // 매개변수에 값 담기

        free(tail); // 꼬리 제거
        tail = before;

        if (before != NULL) // 연결 리스트가 원래 하나밖에 없었을 때 빼고
                before->next = NULL; // 이제 끝 이전 노드가 끝이다

        if (tail == NULL) // 연결 리스트가 원래 하나밖에 없었을 때
                head = NULL;
}
```

스택은 LIFO, 마지막노드부터 pop한다. 이를 구현한 함수이다.

```
void queue_pop(Node* basket)
{
        if(head == NULL)
                return;

        basket->id = head->id;
        strcpy(basket->name,head->name);

        Node* temp;
        if(head->next != NULL)
                head = head->next;
        else
        {
                head = NULL;
                tail = NULL;
        }
}
```

큐는 FIFO, 첫 노드부터 pop한다. 이를 구현한 함수이다.

```
int find(int id, Node* basket)
{
        if(head == NULL)
                return 0;

        // 머리부터 발끝까지 전부 탐색
        Node* temp = head;
        while(temp)
        {
                if(temp->id == id)
                {
                        basket->id = temp->id;
                        strcpy(basket->name,temp->name);
                        return 1;
                }
                temp = temp->next;
        }

        return 0;
}
```

find 함수는 id를 입력시, 처음부터 꼬리까지 노드를 하나씩 비교해가면서, 찾는대상을 포인터로 복사해준다.

원래는, 이것보다 더 많은 기능을 전공수업 때 구현하였다.
id를 기준으로, 뒤에 추가하는 append 함수, 그 앞의 노드를 삭제하는 delete 함수,
리스트 전부를 삭제하는 collaspe 함수등등.

이 강의는 리눅스로 C를 다루면서, 리눅스와 vim, gdb에 익숙해지기위해 듣는 강의이기에, 그 외의 기능은 따로 구현하지 않았다.