

hw1 오리낙시

2019110514 김병재

1. 개요

Linked List를 이용하여 오리 낙시 프로그램을 만드는 과제이다. 구조체를 활용하여 노드를 생성했으며 데이터 입력, 리스트 전체 출력, 특정 노드 출력 및 삭제의 기능을 구현하였다. 기존에 lms에는 오리잡기라고 공지되지 않았으나, 교수님께서 수업시간에 오리 잡기라는 주제로 과제를 설명해주셨기에, 각 노드를 한 마리의 오리로 보았고, 노드에 오리의 이름을 노드에 저장할 데이터로 결정하였다. (추후 재공지 되었으나 많이 진행하였던 터라 변경하지 않았습니다.)

2. 프로그램의 구조

1) 파일 소개

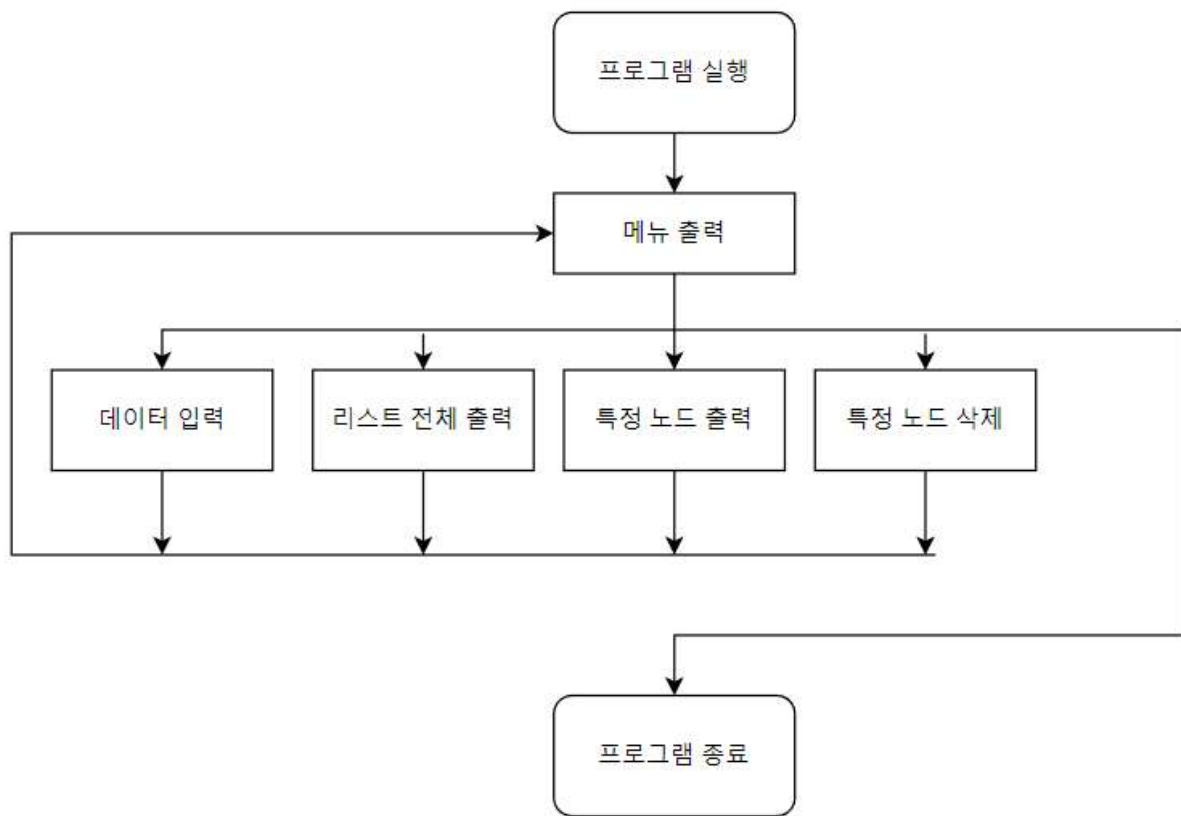
우선, 프로그램을 크게 3개의 파일로 나누어 분할 컴파일 시켰다.

첫 번째는 헤더파일인 Duck.h이다. 헤더파일에는 include할 다른 헤더들과, 나머지 파일들에서 사용되는 구조체 및 함수의 이름을 선언하였다.

두 번째는 Duck.c 파일이다. 이곳에는 main함수에서 사용될 사용자 정의 함수를 정의하였다. 마지막 hw1.c 파일에는 main 함수가 들어있는데, 프로그램의 뼈대를 구성하며 값을 입력받는 등 사용자와 직접적으로 상호작용하는 파일이다. 이외에도 makefile이 있는데 이는 프로그램이 실행되도록 .out 파일을 만들어 준다.

2) 프로그램의 동작 과정

프로그램의 전체적인 틀을 생각해보자. 메뉴를 보여준 후, 사용자로부터 값을 입력받으면 특정 기능을 수행해야 한다. 요구된 기능은 1. 데이터 입력 2. 리스트 전체 출력 3. 특정 노드 출력 4. 특정 노드 삭제가 있으며, 따로 언급되지는 않았지만 '프로그램 종료'는 필수적으로 구현되어야 한다. 또한, 사용자가 프로그램을 종료시키기 전까지는 기능을 수행 한 후, 그 다음 작업을 하기 위한 값을 입력받아야 한다. 순서대로 나타내면 다음과 같다.



[그림 1] 프로그램의 순서도

3) 인자 및 변수

주요 함수를 설명하기 이전에, 함수에서 사용되는 인자 및 변수에 어떤 것이 있는지 확인해보자. 프로그래밍의 편의를 위해 비슷한 의미를 가진 변수와 인자의 이름을 모두 동일하게 설정하였다.

```

// 구조체 정의
typedef struct node {
    char name[30]; // 데이터를 저장할 멤버 변수
    struct node* next; // 다음 노드를 가리키는 포인터
} Duck;
  
```

[그림 2] 구조체 변수

위는 헤더파일에서 정의한 구조체이며, 구조체의 이름은 'Duck'이며 'name'과 'next'라는 멤버 변수를 가지고 있다. 'name'에는 오리의 이름을 저장하며, 구조체 포인터인 'next'는 구조체의 한 노드에서 다음 노드로 이동할 때 사용된다. main 함수에서 사용자로 부터 오리의 이름을 입력받을 때는 문자열 변수 역시 name으로 설정하였다. 나중에 또 언급하겠지만 데이터를 입력받는 함수에서도 'name'을 사용하였다. 나머지 변수들은 다음과 같다.

```

Duck* head = NULL; // 리스트의 첫 번째 노드를 가리키는 포인터 변수
int num;           // 메뉴 선택시 입력받은 값을 저장할 변수
char name[30];     // 오리의 이름을 저장할 변수
int index;         // 특정 노드를 출력하거나 삭제할 때 사용되는 변수
  
```

4) 사용자 정의 함수

(함수를 쉽게 이해하기 위해 설명을 덧붙인다. head는 리스트의 첫 번째 노드를 가리키며, 노드->next가 NULL이면 그 노드는 마지막 노드를 의미한다. 그리고 오리의 이름을 지칭하는 변수는 name이다.)

catch는 데이터를 입력받고 새로운 노드를 만드는(오리잡기) 함수이다.

```
// 데이터 입력하기(오리잡기)
Duck* catch (Duck* head, char name[30]) {
    Duck* new_node = (Duck*)malloc(sizeof(Duck)); // 새로운 노드 생성
    strcpy(new_node->name, name); // 새 노드에 데이터 복사
    new_node->next = NULL; // 새 노드의 다음 노드를 NULL로 설정
    if (head == NULL) { // head가 NULL인 경우, 리스트가 비어있음
        head = new_node; // head에 새 노드를 할당
    }
    else {
        Duck* current = head;
        while (current->next != NULL) { // 마지막 노드를 찾기
            current = current->next;
        }
        current->next = new_node; // 마지막 노드의 다음 노드에 새 노드 할당
    }
    return head; // 새로운 head 포인터 반환
}
```

[그림 3] catch 함수

이 함수는 리스트의 첫 번째 노드를 가리키는 head 포인터와 오리의 이름을 가리키는 name 문자열을 입력받는다. 주석에 설명이 다 되어있긴 하지만 함수가 어떻게 동작하는지 자세히 살펴보자.

- (1) new_node라는 새로운 노드를 생성 후, 구조체의 멤버 변수인 name에 입력받은 name을 복사한다.
- (2) 새 노드의 다음 노드를 비워두고(당연히 새로운 노드 다음에는 데이터를 입력받지 않았기 때문에 NULL이다.) if문을 돌린다.
- (3) head가 NULL인 경우, 즉 리스트의 첫 번째 노드가 비어있는 경우에는 입력받은 head에 new_node를 할당한다.
- (4) 그렇지 않은 경우, current에 현재 노드를 할당한 후, current를 다음 노드로 이동시키면서, 그 다음 노드가 NULL이 되기 전까지 반복문을 돌린다. 반복문에서 빠져나온 후, current의 다음 노드는 비어있을 것이므로 그곳에 새 노드를 할당한다.
- (5) 3번에서 새로 할당된 head를 return해준다.

call_all 함수는 리스트 전체를 출력한다.

```
// 전체 리스트 출력하기
void call_all(Duck* head) {
    int index = 1;
    if (head == NULL)
        printf("잡힌 오리들 없습니다.\n");
    else
    {
        printf("#잡힌 오리들 명단입니다.\n");
        printf("-----\n");
        while (head != NULL) { // head가 NULL이 아닐 때까지 루프 실행
            printf("%d번째 오리는 %s입니다.\n", index++, head->name); // 현재 노드의 데이터 출력
            head = head->next; // 다음 노드로 이동
        }
        printf("-----\n");
    }
}
```

[그림 4] call_all 함수

마찬가지로 리스트의 첫 번째 노드가 NULL인 경우 잡힌 오리가 없음을 의미하고, 그렇지 않은 경우 head노드를 다음 노드로 옮기면서 각 노드에 저장되어있는 name을 출력한다. 여기서 index는 그 노드가 몇 번째 노드인지 사용자에게 알려주기 위하여 사용하였다. index가 출력된 후 1씩 증가하며, n번째 오리를 출력할 경우 while문이 n번 반복되기 때문에 n번째 오리는 다음과 같은 형식으로 출력된다. “n번째 오리는 ooo입니다.”

다음은 특정 노드를 출력하는 call_one 함수이다.

```
void call_one(Duck* head, int index) {
    int count = 0;
    index -= 1;
    while (head != NULL) { // head가 NULL이 아닐 때까지 루프 실행
        if (count == index) { // 인덱스가 일치하는 노드인 경우
            printf("%d번째 오리는 %s입니다.\n", index+1, head->name); // 사용자의 편의를 위해 index와 n번째가 동일하게 맞춤
            return;
        }
        head = head->next; // 다음 노드로 이동
        count++; // 노드의 개수 증가
    }
    printf("%d번째 오리가 없습니다.\n", index+1); // 해당 인덱스의 노드가 없는 경우
}
```

[그림 5] call_one 함수

call_all 함수와 다르게 이 함수는 특정 노드를 출력해야 하므로 매개변수인 index를 추가로 입력받았다. count는 0부터 시작하는데 반복문이 실행되면 head가 그 다음 노드로 이동하면서 count도 1씩 증가한다. 이때 count와 index가 같으면 그 노드에 저장된 데이터인 name을 출력하고 함수가 종료되도록 설계하였다. 이때 while문은 head가 빈 노드일 경우 종료하는데, 그때까지 index와 count가 같은 값이 되지 않으면, 즉 입력받은 index가 총 노드의 개수보다 많은 경우에는 사용자에게 잘못된 값을 입력받았음을 알려준다.

여기서 index -=1; 을 한 이유는 나중에 고찰에서 설명하겠다.

```

// 특정 노드 삭제하기
Duck* kill(Duck* head, int index) {
    int count = 0;
    index -= 1;
    Duck* current = head; // 현재 노드를 가리키는 포인터
    Duck* prev = NULL; // 이전 노드를 가리킬 포인터
    while (current != NULL) { // current가 NULL이 아닐 때까지 루프 실행
        if (count == index) { // 주어진 인덱스에 해당하는 노드를 찾으면
            if (index == 0) { // 삭제할 노드가 헤드 노드일 경우
                printf("%d번째 오리 %s 를 꺼냈습니다\n", index+1, current);
                head = current->next; // 헤드 노드를 변경
            }
            else {
                printf("%d번째 오리 %s 를 꺼냈습니다\n", index + 1, current);
                prev->next = current->next; // 그렇지 않으면 이전 노드의 다음 노드를 현재 노드의 다음 노드로 변경
            }
            free(current);
            return head;
        }
        else {
            prev = current;
            current = current->next;
            count++;
        }
    }
    printf("%d번째 오리가 없습니다.\n", index+1); // 주어진 인덱스에 해당하는 노드가 없는 경우
    printf("%d보다 작은 수를 입력하세요", count);
    return head;
}

```

[그림 6] kill 함수

kill 함수는 특정 노드를 삭제해주는 함수로, 리스트의 첫 부분인 head와 삭제하고 싶은 index를 입력받는다. 삭제할 노드를 찾는 과정은 call_one 함수와 동일하지만, 삭제된 노드의 앞부분과 뒷부분을 서로 연결 해야하며 동적 할당된 메모리를 초기화시켜야 한다는 차이가 있다.

- (1) 현재 노드와 이전 노드를 가리키는 포인터 생성 및 count를 0으로 초기화.
- (2) 현재 노드가 비어있을 때 까지 루프 실행 (3.1 ~ 3.2)
 - (3.1) 입력받은 index와 count가 같을 경우 현재
 - (3.1.1) index가 0인 경우 head 노드를 그 다음 노드로 변경 후 동적 메모리 해제 및 함수 종료
 - (3.1.2) 그 외, 이전 노드의 다음 노드를 현재 노드의 다음 노드로 변경 후 동적 메모리 해제 및 함수 종료
 - (3.2) index와 count가 서로 다를 경우 이전 노드에 현재 노드를 할당하고, 현재 노드는 다음 노드로 이동 및 count 1 증가
- (4) index가 리스트의 노드 수 보다 더 클 경우 사용자에게 알려준다.

이전의 call_one 함수에서는 head 포인터를 이동시켰으나 current를 이용하여 다음 노드로 이동한 이유는 동적 할당을 안전하게 해제하기 위해서이다. 만약 free(head)를 할 경우, head가 가리키는 메모리 주소에 접근할 수 없기 때문에 다음 노드로 접근이 불가능하다.(head = head->next 불가) current를 이용할 경우, 삭제될 노드의 앞뒤 노드를 이어 붙인 다음 메모리해제를 해주기 때문에 오류가 발생하지 않는다. 쉽게 말하자면 a와 b의 값을 바꿀 때 temp라는 변수를 추가로 사용하는 것과 비슷한 맥락이라고 생각한다.

그 다음은 메뉴를 출력하는 print_menu() 함수인데 main 함수를 깔끔하게 만들기 위해 작성하였다. 이 함수는 매우 직관적이므로 추가적인 설명은 생략한다.

```
// 메뉴출력 함수
void print_menu() {
    printf("오리잡기 프로그램입니다.\n");
    printf("-----menu-----\n");
    printf("1. 오리잡기\n"); // 데이터 입력하기(문자열을 리스트에 넣기)
    printf("2. 잡힌오리 모두 확인하기\n"); // 전체 리스트 출력하기
    printf("3. n번째 오리 이름 확인하기\n"); // 특정 노드 출력하기
    printf("4. 오리 꺼내기\n"); // 특정 노드 삭제하기
    printf("5. 프로그램 종료\n");
    printf("-----\n");
    printf("메뉴를 고르세요: ");
}
```

[그림 7] 메뉴 출력 함수

```
void end(Duck* head) {
    int count = 0;
    printf("\n");
    while (head != NULL) { // head가 NULL이 아닐 때까지 루프 실행
        Duck* temp = head; //동적할당을 해제하기 위한 임시 포인터
        printf("%s : 살려주셔서 감사합니다!!\n\n", temp->name);
        head = head->next;
        free(temp);
        count++;
    }
    printf("%d마리의 오리를 살려주었습니다. 곧 프로그램이 종료됩니다...\n\n", count);
}
```

[그림 8] end 함수

end 함수 자체로는 프로그램을 종료시키지 않지만, 이는 프로그램을 종료하기 전에 리스트를 구성하는 모든 노드를 메모리에서 해제하도록 만들었다. 매개변수로는 리스트의 첫 번째 노드를 가리키는 포인터 head가 전달되었으며, kill 함수와 같은 이유로 temp라는 포인터를 추가로 사용하였다. 함수는 다음과 같이 동작한다.

- (1) 현재 노드(head)가 NULL이 아닐 때까지 루프 실행 (2번 ~ 4번)
- (2) temp에 동적 할당된 메모리 주소를 저장
- (3) head를 다음 노드로 이동
- (4) temp가 가리키는 메모리의 동적 할당을 해제

count변수는 동적 할당 해제시 아무 쓸모가 없지만 디버깅을 위해 추가하였으며, 프로그램 종료 시 사용자가 잡았던 오리가 총 몇 마리 인지 확인하기 위해 함수 마지막 부분에 print문을 추가하였다.


```

int main() {
    Duck* head = NULL; // 리스트의 첫 번째 노드를 가리키는 포인터 변수
    int num; // 메뉴 선택시 입력받은 값을 저장할 변수
    char name[30]; // 오리 이름 저장할 변수
    int index; // 특정 노드를 출력하거나 삭제할 때 사용되는 변수

    print_menu(); //메뉴출력
    printf("메뉴를 선택하세요 : ");
    scanf("%d", &num); // 메뉴 선택

    while (1) { // 프로그램 종료까지 실행되는 무한루프

        switch (num) {
            case 1: // 데이터 입력하기
                printf("잡을 오리의 이름을 입력하세요 : ");
                scanf("%s", name);
                head = catch(head, name); // 입력받은 데이터를 리스트에 추가
                printf("%s를 잡았습니다\n", name);
                printf("\n");
                break;

            case 2: // 전체 리스트 출력하기
                call_all(head); // 전체 리스트 출력
                printf("\n");
                break;

            case 3: // 특정 노드 출력하기
                printf("몇 번째 오리를 확인할까요? : ");
                scanf("%d", &index); // 출력할 노드의 인덱스를 입력
                call_one(head, index); // 해당 노드를 출력
                printf("\n");
                break;
        }
    }
}

```

```

case 4: // 특정 노드 삭제하기
    printf("몇 번째 오리를 꺼낼까요? : ");
    scanf("%d", &index); // 삭제할 노드의 인덱스를 입력
    head = kill(head, index); // 해당 노드를 삭제
    printf("\n");
    break;

case 5: // 프로그램 종료
    end(head);
    break;

case 6: // 메뉴 출력
    print_menu();
    break;

default: // 잘못된 입력을 받을 경우
    printf("1에서 5까지 정수를 입력하세요\n");
    break;
}

if (num == 5) // 5번을 고른경우 프로그램 종료
    break;

else { //그렇지 않은 경우 옵션을 입력받을
    printf("메뉴를 선택하세요(메뉴를 확인하시려면 6번을 입력하세요) : ");
    scanf("%d", &num);
}

return 0;

```

[그림 9] main함수

마지막으로 main 함수이다. 프로그램을 실행하면 메뉴를 출력 후, num에 숫자를 입력받은 다음 무한루프를 실행한다. 이때 switch ~ case 문을 이용하여 num에 따라 특정 함수가 실행되도록 만들었다. 프로그램이 잘 동작하는지 확인하는 도중, 굳이 매번 메뉴를 출력할 필요성을 느끼지 못하였기에 6을 입력받으면 메뉴가 출력되도록 프로그램을 짰다. 이 과정에서 print_menu 함수를 추가로 만들었다. 다른 함수에 비해 main 함수를 만들 때 어려움은 없었으나 프로그램이 잘 돌아가는지 확인하다가 큰 문제를 발견하였다. (이 역시 고찰에서 언급하겠다.)

```

byoengjae@byoengjae-VirtualBox: ~/Desktop/dnscp/hw1
hw1.out : duck.o hw1.o
gcc -o hw1.out duck.o hw1.o

duck.o : duck.c
gcc -c -o duck.o duck.c

hw1.o : hw1.c
gcc -c -o hw1.o hw1.c

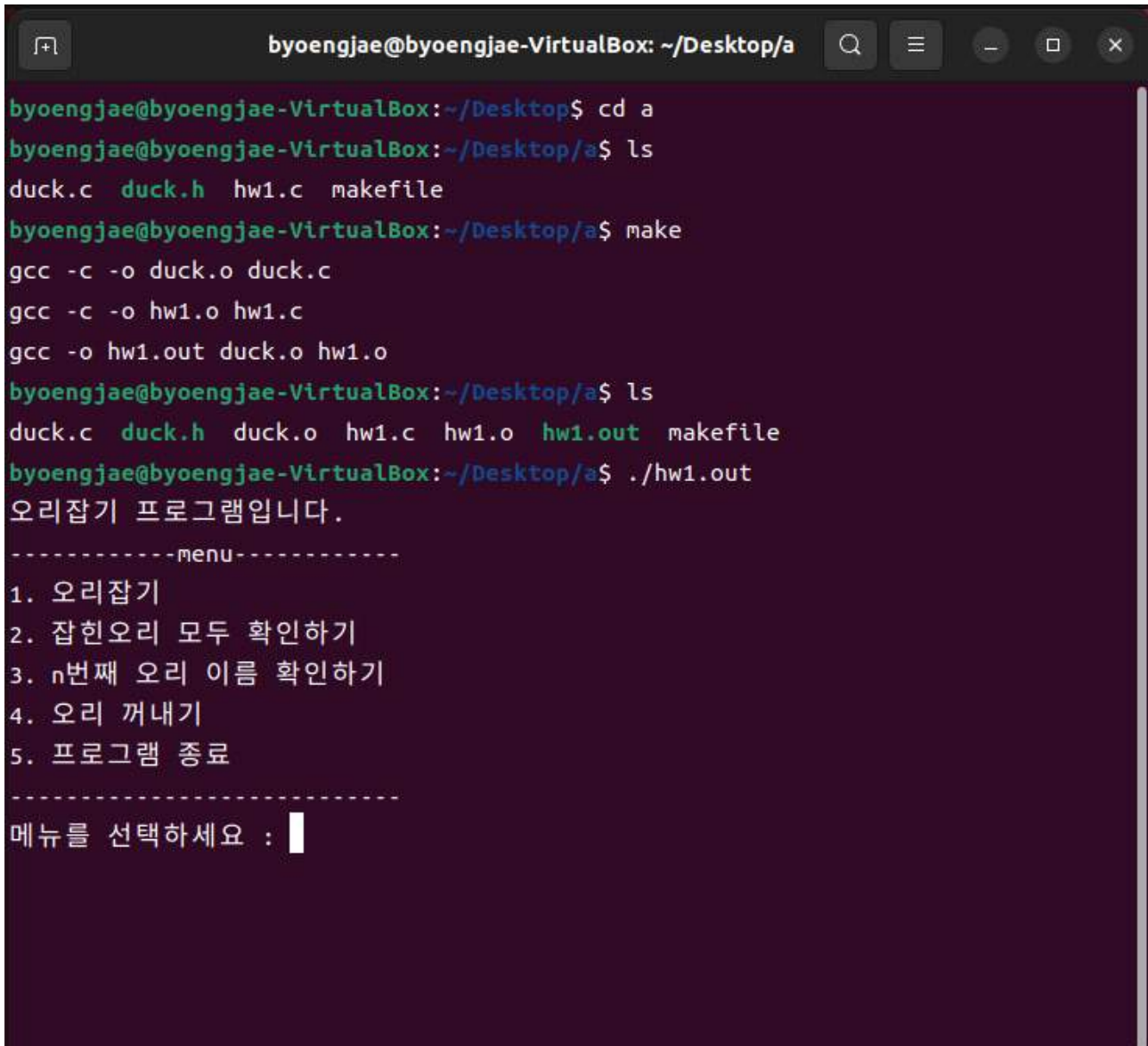
```

[그림 10] Makefile

Makefile은 내가 작성한 duck.c hw1.c를 컴파일 및 실행파일인 hw1.out을 생성하기 위해 작성하였다. 처음에는 오브젝트 파일을 직접 만든 후, 실행파일을 만드는 방식으로 컴파일 하였는데 프로그램을 수정 할때마다 오브젝트파일과 실행파일을 새로 생성해야 한다는 불편함을 해결하기 위해 Makefile을 만들었다. duck.c와 hw1.c파일의 objectfile을 각각 duck.o hw1.o로, 실행 파일의 이름은 hw1.out이다.

[프로그램 실행 매뉴얼]

1. 파일이 저장된 디렉토리로 이동 후, make를 입력하여 실행파일인 hw1.out을 반드시 생성해주세요
2. 메뉴에는 반드시 정수를 입력하세요.
3. 오리 이름은 가급적 5글자 이내로 입력하세요.

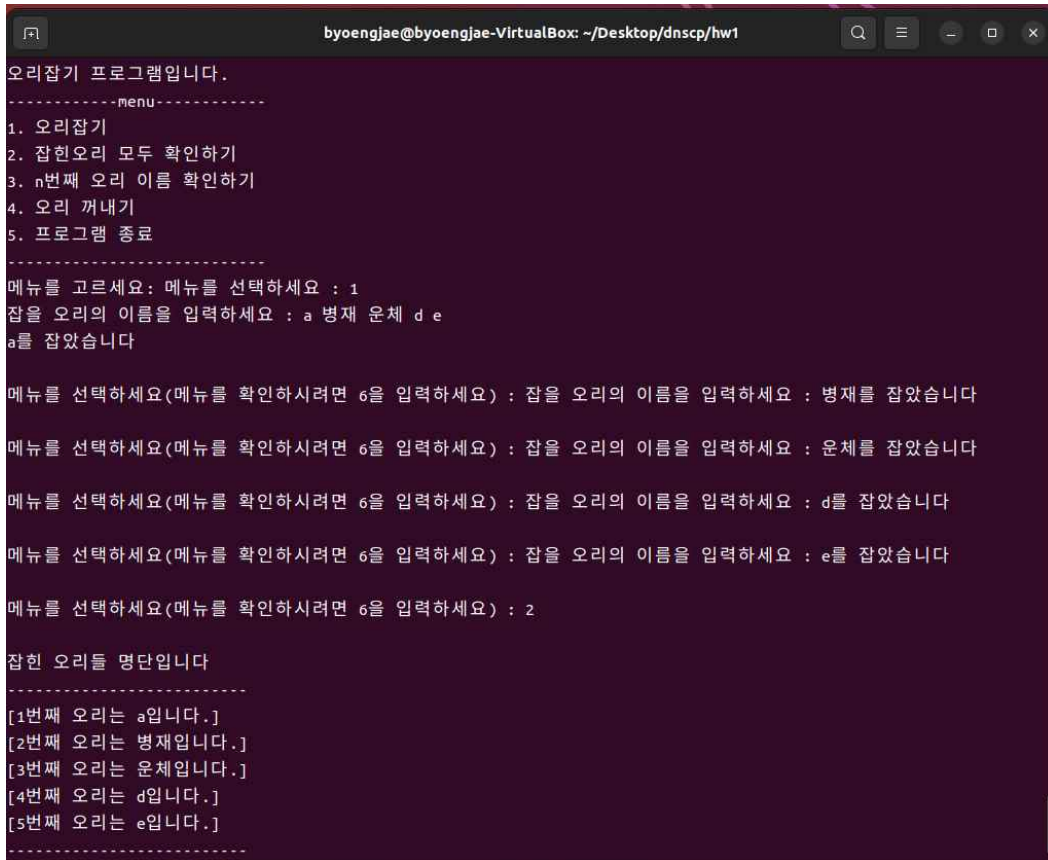


```
byoengjae@byoengjae-VirtualBox: ~/Desktop/a
byoengjae@byoengjae-VirtualBox:~/Desktop$ cd a
byoengjae@byoengjae-VirtualBox:~/Desktop/a$ ls
duck.c  duck.h  hw1.c  makefile
byoengjae@byoengjae-VirtualBox:~/Desktop/a$ make
gcc -c -o duck.o duck.c
gcc -c -o hw1.o hw1.c
gcc -o hw1.out duck.o hw1.o
byoengjae@byoengjae-VirtualBox:~/Desktop/a$ ls
duck.c  duck.h  duck.o  hw1.c  hw1.o  hw1.out  makefile
byoengjae@byoengjae-VirtualBox:~/Desktop/a$ ./hw1.out
오리잡기 프로그램입니다.
-----menu-----
1. 오리잡기
2. 잡힌오리 모두 확인하기
3. n번째 오리 이름 확인하기
4. 오리 꺼내기
5. 프로그램 종료
-----
메뉴를 선택하세요 : █
```

[그림 11] 프로그램 실행 방법

3. 실행결과

프로그램 실행결과는 다음과 같다.



```
byoengjae@byoengjae-VirtualBox: ~/Desktop/dnscp/hw1
오리잡기 프로그램입니다.
-----menu-----
1. 오리잡기
2. 잡힌오리 모두 확인하기
3. n번째 오리 이름 확인하기
4. 오리 꺼내기
5. 프로그램 종료
-----
메뉴를 고르세요: 메뉴를 선택하세요 : 1
잡을 오리의 이름을 입력하세요 : a 병재 운체 d e
a를 잡았습니다

메뉴를 선택하세요(메뉴를 확인하시려면 6을 입력하세요) : 잡을 오리의 이름을 입력하세요 : 병재를 잡았습니다

메뉴를 선택하세요(메뉴를 확인하시려면 6을 입력하세요) : 잡을 오리의 이름을 입력하세요 : 운체를 잡았습니다

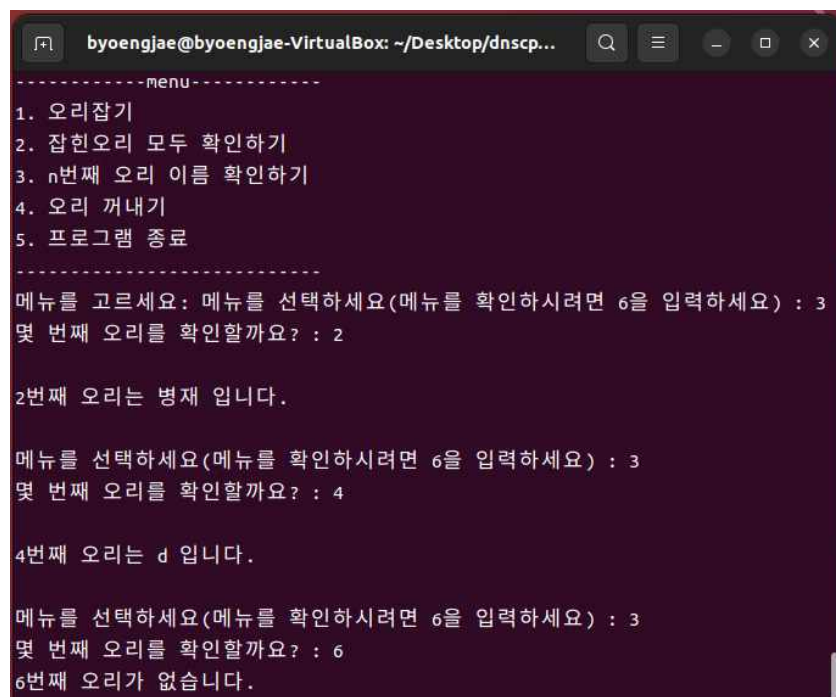
메뉴를 선택하세요(메뉴를 확인하시려면 6을 입력하세요) : 잡을 오리의 이름을 입력하세요 : d를 잡았습니다

메뉴를 선택하세요(메뉴를 확인하시려면 6을 입력하세요) : 잡을 오리의 이름을 입력하세요 : e를 잡았습니다

메뉴를 선택하세요(메뉴를 확인하시려면 6을 입력하세요) : 2

잡힌 오리들 명단입니다
-----
[1번째 오리는 a입니다.]
[2번째 오리는 병재입니다.]
[3번째 오리는 운체입니다.]
[4번째 오리는 d입니다.]
[5번째 오리는 e입니다.]
-----
```

[그림 12] 메뉴 1번(데이터 입력) 및 2번(전체 리스트 출력)



```
byoengjae@byoengjae-VirtualBox: ~/Desktop/dnscp...
-----menu-----
1. 오리잡기
2. 잡힌오리 모두 확인하기
3. n번째 오리 이름 확인하기
4. 오리 꺼내기
5. 프로그램 종료
-----
메뉴를 고르세요: 메뉴를 선택하세요(메뉴를 확인하시려면 6을 입력하세요) : 3
몇 번째 오리를 확인할까요? : 2

2번째 오리는 병재 입니다.

메뉴를 선택하세요(메뉴를 확인하시려면 6을 입력하세요) : 3
몇 번째 오리를 확인할까요? : 4

4번째 오리는 d 입니다.

메뉴를 선택하세요(메뉴를 확인하시려면 6을 입력하세요) : 3
몇 번째 오리를 확인할까요? : 6
6번째 오리가 없습니다.
```

[그림 13] 메뉴 3번(특정 노드 출력)

원하는 노드를 출력하고, 존재하지 않는 노드를 입력받은 경우에도 잘 동작하는 것을 볼 수 있다.

```
byoengjae@byoengjae-VirtualBox: ~/Desktop/dnsc...
메뉴를 선택하세요(메뉴를 확인하시려면 6을 입력하세요) : 6
오리잡기 프로그램입니다.
-----menu-----
1. 오리잡기
2. 잡힌오리 모두 확인하기
3. n번째 오리 이름 확인하기
4. 오리 꺼내기
5. 프로그램 종료
-----
메뉴를 고르세요: 메뉴를 선택하세요(메뉴를 확인하시려면 6을 입력하세요) :
4
몇 번째 오리를 꺼낼까요? : 2

2번째 오리 병재 를 꺼냈습니다

남아있는 오리입니다
-----
[1번째 오리는 a입니다.]
[2번째 오리는 운체입니다.]
[3번째 오리는 d입니다.]
[4번째 오리는 e입니다.]
-----
```

```
byoengjae@byoengjae-VirtualBo...
메뉴를 선택하세요(메뉴를 확인하시려면 6을 입력하세요) : 4
몇 번째 오리를 꺼낼까요? : 3

3번째 오리 d 를 꺼냈습니다

남아있는 오리입니다
-----
[1번째 오리는 a입니다.]
[2번째 오리는 운체입니다.]
[3번째 오리는 e입니다.]
-----
메뉴를 선택하세요(메뉴를 확인하시려면 6을 입력하세요) : 4
몇 번째 오리를 꺼낼까요? : 4

4번째 오리가 없습니다.
현재 잡힌 오리는 총 3마리 입니다.

남아있는 오리입니다
-----
[1번째 오리는 a입니다.]
[2번째 오리는 운체입니다.]
[3번째 오리는 e입니다.]
-----
```

[그림 14] 메뉴 4번(특정 노드 삭제)

특정 노드를 삭제하며, 존재하지 않는 노드를 삭제하라고 입력받은 경우에도 잘 동작하는 것을 확인할 수 있다.

```
byoengjae@byoengjae-VirtualBox: ~/Desktop/dnscp...
메뉴를 선택하세요(메뉴를 확인하시려면 6을 입력하세요) : 6
오리잡기 프로그램입니다.
-----menu-----
1. 오리잡기
2. 잡힌오리 모두 확인하기
3. n번째 오리 이름 확인하기
4. 오리 꺼내기
5. 프로그램 종료
-----
메뉴를 고르세요: 메뉴를 선택하세요(메뉴를 확인하시려면 6을 입력하세요) : 5

a : 살려주셔서 감사합니다!!

운체 : 살려주셔서 감사합니다!!

e : 살려주셔서 감사합니다!!

3마리의 오리를 살려주었습니다. 곧 프로그램이 종료됩니다...

byoengjae@byoengjae-VirtualBox:~/Desktop/dnscp/hw1$
```

[그림 15] 메뉴 5번 및 6번

6번을 입력하면 메뉴를 출력하며, 5번을 선택 시 남아있는 모든 오리를 살려주고(모든 메모리의 동적 할당을 해제 하고) 프로그램이 종료된다.

※실행에 실패한 과정은 모두 고찰에서 언급하겠다.

4. 고찰 및 느낀점

이번 과제는 Linked List를 이용한 오리잡기 프로그램을 만드는 것이다. 따라서 표면적으로는 운영체제보다 자료 구조에 더 가깝다고 생각했는데, 실제로는 그렇지 않았다. 가상머신을 다운받아 Ubuntu를 설치하고, 터미널에서 vim을 이용하여 소스 코드를 수정하며, gcc를 이용하여 컴파일을 하는 등 새로운 환경에서 프로그래밍을 하는 것이 과제의 핵심이라고 생각한다. 처음에는 진짜 막막했다. 교수님께서 수업시간에 설명해주시긴 했는데 그 당시에 하나도 이해가 되지 않았을뿐더러, ppt를 보아도 무슨 내용인지 알 수 없었다. 일단 우분투를 설치하긴 했는데 그 다음에 뭘 어떻게 해야하는지, 소스 코드는 어디다 작성해야 하고 파일은 어떻게 생성하는지 몰라서 며칠 동안은 Ubuntu의 사용법을 공부했다. 그 과정에서 vim 에디터라는 것을 알게 되었고 ls, cd 등의 명령어를 몸에 익혔다.

Ubuntu에는 적응했으나, 문제는 내가 자료구조 수업을 수강하지 않았다는 것이다. Linked list가 어떠한 원리로 동작하는지는 이해했지만 공부해 본 적이 없어서 프로그램을 만드는데 시간이 꽤 걸렸다. 책을 빌리고, 구글링을 하면서 코드를 얼추 작성하였으나 꽤 많이 실패를 겪었다. 보고서 작성을 생각하기 전에는 실패한 경우를 모두 기록하지 않았기 때문에 아래에서 언급하는 에러 이외에도 다양한 실패과정이 있었음을 알아주셨으면 한다(합니다.)

처음에는 변수 char을 문자열이 아닌 문자로 지정해두었기에 오리 이름을 입력받을 때 제한이 있었다. 따라서 char name[30]으로 변경하였다. (한글이나 여러 글자로 입력할 경우 오류 발생)

또한, catch함수에서 new_node의 멤버 변수 name에 입력받은 name을 저장하기 위해 `new_node->name = name;` 라는 코드를 작성하였더니, `식이 수정할 수 있는 lvalue여야 합니다.` 라는 경고메시지가 발생하였다.

이 오류는 C 언어에서 포인터를 가리키는 변수나 배열 요소에 값을 할당하려 할 때 발생하는데 이 코드에서 `new_node->name`은 문자열을 가리키는 포인터 변수이고, `name`은 문자열이므로 `name`의 주소값이 포인터 변수에 할당된다. 이후, `name`이 변경된다면 `new_node`가 가리키는 메모리 영역에 저장된 값도 변경 될 수 있는데, `name`은 구조체에서 정의된 문자열 이므로 수정이 불가능하다. 따라서 따라서 strcpy 함수를 사용하여 값을 복사하는 방식으로 이 문제를 해결하였다.

지금 막 보고서를 작성하는 도중 생각난 방법인데, name이 문자열이기 때문에 즉, 수정할 수 없는 lvalue라서 발생하는 오류면 구조체를 정의할 때 포인터로 설정하면 어떨까? 라는 생각을 하였다. 따라서 구조체를 다음과 같이 정의한 후, 프로그램을 실행해보았다.

```
typedef struct node {
    char* name; // 데이터를 저장할 멤버 변수
    struct node* next; // 다음 노드를 가리키는 포인터
} Duck;
```

```
Duck* catch (Duck* head, char name[30]) {
    Duck* new_node = (Duck*)malloc(sizeof(Duck)); // 새로운 노드 생성
    new_node->name = name;
    //strcpy(new_node->name, name); // 새 노드에 데이터 복사
    new_node->next = NULL; // 새 노드의 다음 노드를 NULL로 설정
    if (head == NULL) { // 리스트가 비어있는 경우
        head = new_node; // head에 새 노드를 할당
    }
    else {
        Duck* current = head; // 현재 노드를 head로 설정
        while (current->next != NULL) { // 마지막 노드를 찾기위한 while문
            current = current->next; //현재 노드를 다음 노드로 이동
        }
        current->next = new_node; // 마지막 노드의 다음 노드에 새 노드 할당
    }
    return head; // 새로운 head 포인터 반환
}
```

[그림 18] 멤버 변수 name을 문자열에서 포인터로 바꾸기

```

잡힌 오리들 명단입니다
-----
[1번째 오리는 c입니다.]
[2번째 오리는 c입니다.]
[3번째 오리는 c입니다.]
-----

메뉴를 선택하세요(메뉴를 확인하시려면 6을 입력하세요) : 4
몇 번째 오리를 꺼낼까요? : 1

1번째 오리 (효쫄 를 꺼냈습니다

```

[그림 19] 오류 발생

프로그램은 실행되지만, 이상한 값이 출력되는 것을 확인 할 수 있다. name[30]은 catch함수의 지역변수인데, new_node->name에 name[30]의 주소를 복사한 경우, 함수가 종료되면 name[30]이 사라지기 때문에 name이 가리키는 메모리 영역이 정상적인 값을 가지지 않게 되어 위와 같은 오류가 발생한 것이다. 따라서 그냥 strcpy를 사용하여 문자열을 복사하는 방법이 가장 최선이라고 판단하였다. (여기서 빨간색 name은 구조체 포인터를 의미함)

교수님께서 수업시간에 설명해주신 오리잡기는 FIFO 방식을 적용해야 하므로, 기존의 catch함수는 지금과는 달랐다. 아래와 같이 새로운 데이터가 가장 앞쪽 노드에, 기존 데이터가 뒤로 가도록 설계했었다. 그러나 lms상에 공지된 과제에는 그런 조건이 없었을뿐더러, 특정 노드 삭제시 index를 사용하기 때문에 먼저 잡힌 오리가 1번에 위치하도록 설정하는 것이 유저에게 편리하다고 생각하였다. 따라서 새로 입력받은 데이터를 기존 데이터의 다음 노드에 저장하도록 수정하였다.

```

/*(나중에 잡힌 오리가 리스트의 앞쪽에 위치하는 방식)
Duck* catch (Duck* head, char name[30]) {
    Duck* new_node = (Duck*)malloc(sizeof(Duck)); // 새로운 노드 생성
    strcpy(new_node->name, name); // 새 노드에 데이터 복사
    new_node->next = NULL; // 새 노드의 다음 노드를 NULL로 설정
    if (head == NULL) { // head가 NULL인 경우, 즉 첫번째 노드인 경우
        head = new_node; // head에 새 노드를 할당
    }
    else {
        new_node->next = head; //그렇지 않을 경우 새 노드의 다음노드를 head로 설정
        head = new_node; //head에 새 노드를 할당
    }
    return head; // 새로운 head 포인터 반환
}
*/

```

[그림 20] 수정전 catch함수

수정전에는 head(기존 리스트의 가장 앞쪽)를 new_node의 다음 노드로 설정한 다음, head에 new_node를 할당하는 방식으로 코드를 짰다. 이 경우 굳이 마지막 노드를 찾을 필요가 없기 때문에, 지금과 비교하면 매우 간단해 보인다. 이번 과제에서 사용되지 않는 코드지만 작성해둔게 아까워서 그냥 첨부해두었다.


```

// 특정 노드 출력하기
void call_one(Duck* head, int index) {
    int count = 0;
    while (head != NULL) { // head가 NULL이 아닐 때까지 루프 실행
        if (count == index) { // 인덱스가 일치하는 노드인 경우
            printf("\n%d번째 오리는 %s 입니다.\n", index, head->name);
            return;
        }
        head = head->next; // 다음 노드로 이동
        count++; // 노드의 개수 증가
    }
    printf("%d번째 오리가 없습니다.\n", index); // 해당 인덱스의 노드가 없는 경우
}

```

[그림 21] 기존의 call 함수

```

1. 오리잡기
2. 잡힌 오리 모두 확인하기
3. 특정 오리 확인하기
4. 오리 꺼내기
5. 프로그램 종료
선택하세요: 1
어떤 오리를 잡을까요: a b c d e

잡힌 오리는 다음과 같습니다: a
메뉴를 선택하세요: 어떤 오리를 잡을까요:
잡힌 오리는 다음과 같습니다: a b
메뉴를 선택하세요: 어떤 오리를 잡을까요:
잡힌 오리는 다음과 같습니다: a b c
메뉴를 선택하세요: 어떤 오리를 잡을까요:
잡힌 오리는 다음과 같습니다: a b c d
메뉴를 선택하세요: 어떤 오리를 잡을까요:
잡힌 오리는 다음과 같습니다: a b c d e
메뉴를 선택하세요: 3
몇번째 오리를 확인할까요: 1
1번째 오리는 b 입니다.
메뉴를 선택하세요:

```

[그림 22] 실행 결과

이번에는 프로그래밍적(c언어 관련) 오류 말고 사용자가 프로그램을 사용할 때 발생했던 오류를 살펴보자.

[그림 22]를 보면 잡힌 오리는 a b c d e인데 1번째 오리를 확인하려고 하니 2번째 오리 b가 출력되었다. count는 0부터 시작하여 while문이 반복될 때마다 1씩 증가하는데 index가 n인 경우, 루프가 n+1번 반복되어야 count = index = n이 되므로 위와 같은 결과가 나왔다. 따라서 내가 생각해낸 방법은 index -= 1;을 취한 후, 값을 찾은 다음 출력할 때는 1을 더하는 방식으로 코드를 짰다. 그런데 보고서를 작성하면서 생각해보니까 count가 0이 아닌 1부터 시작하면 더 간단하게 해결 할 수 있다는 사실을 깨달았다. 불필요한 계산과정을 줄일 수 있기 때문에 더욱 효율적인 코드라고 생각한다.

```

void call_one(Duck* head, int index) {
    int count = 1;
    while (head != NULL) { // head가 NULL이 아닐 때까지 루프 실행
        if (count == index) { // 인덱스가 일치하는 노드인 경우
            printf("\n%d번째 오리는 %s 입니다.\n", index, head->name); // 사용자의 편의를 위해 index와 n번째가 동일하게 맞춤
            return;
        }
        head = head->next; // 다음 노드로 이동
        count++; // 노드의 개수 증가
    }
    printf("%d번째 오리가 없습니다.\n", index); // 해당 인덱스의 노드가 없는 경우
}

```

[그림 23] count가 1부터 시작

```

1. 오리잡기
2. 잡힌 오리 모두 확인하기
3. 특정 오리 확인하기
4. 오리 꺼내기
5. 프로그램 종료
선택하세요: 1
어떤 오리를 잡을까요: a b c d e

잡힌 오리는 다음과 같습니다: a
메뉴를 선택하세요: 어떤 오리를 잡을까요:
잡힌 오리는 다음과 같습니다: a b
메뉴를 선택하세요: 어떤 오리를 잡을까요:
잡힌 오리는 다음과 같습니다: a b c
메뉴를 선택하세요: 어떤 오리를 잡을까요:
잡힌 오리는 다음과 같습니다: a b c d
메뉴를 선택하세요: 어떤 오리를 잡을까요:
잡힌 오리는 다음과 같습니다: a b c d e
메뉴를 선택하세요: 3
몇번째 오리를 확인할까요: 3
3번째 오리는 c 입니다.
메뉴를 선택하세요:

```

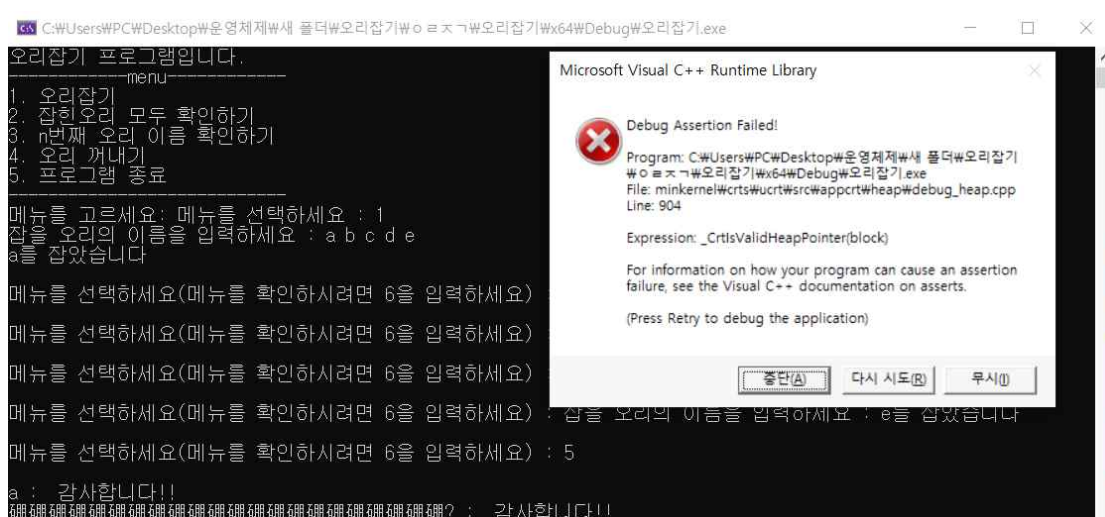
[그림 24] 인덱스 수정결과

프로그램 종료 전, 모든 노드의 동적 메모리를 해제하는 과정에서도 많은 시행착오를 겪었다. 제일 처음 만든 end 함수에서는 temp라는 임시 포인터를 이용하여 동적 할당을 해제한 후, head를 다음 노드로 옮기려고 하였다.

```

void end(Duck* head) {
    Duck* temp = head; //메모리를 초기화 시키기 위한 임시 포인터
    int count = 0;
    while (temp != NULL) { // temp가 NULL이 아닐 때까지 루프 실행
        printf("%s : 감사합니다!!\n", temp->name);
        free(temp);
        head = head->next;
        count++;
    }
    printf("%d마리의 오리를 살려주었습니다. 곧 프로그램이 종료됩니다...", count);
}

```



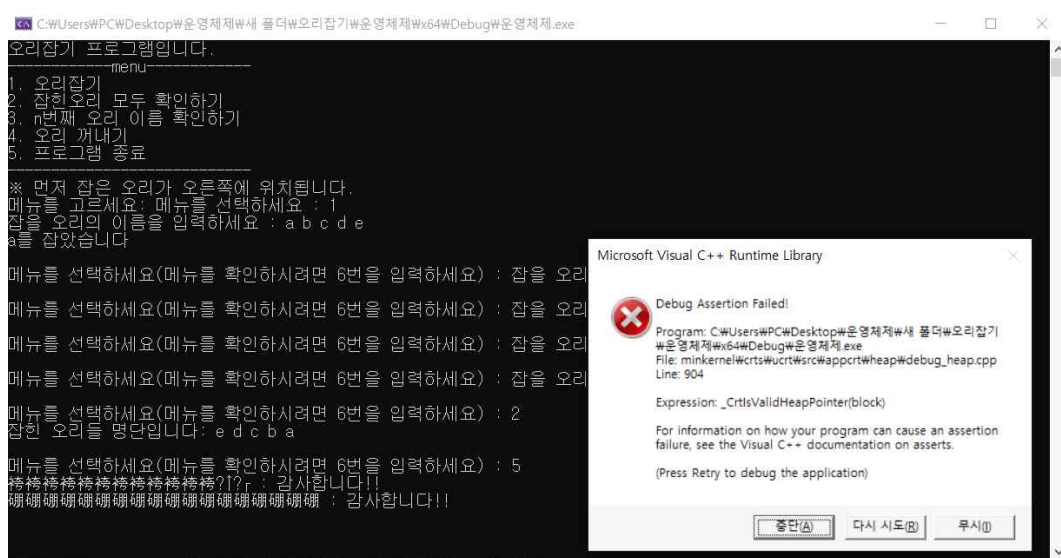
[그림 25] 기존의 end함수 와 실행결과

처음에는 temp에 동적 할당을 하지 않았기 때문에 그랬더니 위와 같은 오류가 발생한 줄 알고 아래와 같이 코드를 수정하였다.

```
void end(Duck* head) {
    Duck* temp = (Duck*)malloc(sizeof(Duck)); //동적할당을 해제하기 위한 임시 포인터
    int count = 0;
    printf("\n");
    while (head != NULL) { // head가 NULL이 아닐 때까지 루프 실행

        printf("%s : 감사합니다!!\n", temp->name);
        free(temp);
        head = head->next;

        count++;
    }
    printf("%d마리의 오리를 살려주었습니다. 곧 프로그램이 종료됩니다...\n\n", count);
}
```



[그림 26] end함수 1차 수정본 및 실행결과

수정 전에는 첫 번째 오리라도 출력되었는데 이번에는 첫 번째 오리도 이상하게 출력되는 것을 확인할 수 있다. 따라서 이는 동적 할당의 문제가 아니라 temp에 head를 대입하지 않았기 때문임을 알게 되었고, 루프가 반복될 때 free(temp) 이후에 temp->name에서 오류가 발생했을 거라고 확신하였다.

```
void end(Duck* head) {
    int count = 0;
    while (head != NULL) { // head가 NULL이 아닐 때까지 루프 실행
        Duck* temp = head; //동적할당을 해제하기 위한 임시 포인터
        printf("%s : 감사합니다!!\n", temp->name);
        free(temp);
        head = head->next;
        count++;
    }
    printf("%d마리의 오리를 살려주었습니다. 곧 프로그램이 종료됩니다...", count);
}
```

[그림 27] end함수 2차 수정

이번에는 while문 내부에서 temp 포인터를 생성하기 때문에 메모리 동적 할당을 해제하더라도, 다음 루프에서 재 생성된 후 head를 대입하므로 오류가 발생하지 않을것이라 생각하였으나

```

Microsoft Visual Studio 디버그 콘솔
오리잡기 프로그램입니다.
menu
1. 오리잡기
2. 잡힌 오리 모두 확인하기
3. n번째 오리 이름 확인하기
4. 오리 꺼내기
5. 프로그램 종료

※ 먼저 잡은 오리가 오른쪽에 위치됩니다.
메뉴를 고르세요: 메뉴를 선택하세요 : 1
잡을 오리의 이름을 입력하세요 : a b c d e
a를 잡았습니다

메뉴를 선택하세요(메뉴를 확인하시려면 6번을 입력하세요) : 잡을 오리의 이름을 입력하세요 : b를 잡았습니다
메뉴를 선택하세요(메뉴를 확인하시려면 6번을 입력하세요) : 잡을 오리의 이름을 입력하세요 : c를 잡았습니다
메뉴를 선택하세요(메뉴를 확인하시려면 6번을 입력하세요) : 잡을 오리의 이름을 입력하세요 : d를 잡았습니다
메뉴를 선택하세요(메뉴를 확인하시려면 6번을 입력하세요) : 잡을 오리의 이름을 입력하세요 : e를 잡았습니다
메뉴를 선택하세요(메뉴를 확인하시려면 6번을 입력하세요) : 5
e : 감사합니다!!

C:\Users\PC\Desktop\운영체제\새 폴더\오리잡기\운영체제\x64\Debug\운영체제.exe(프로세스 20436개)이(가) 종료되었습니다(코드: -1073741819개).
이 창을 닫으려면 아무 키나 누르세요...
  
```

[그림 28] 2차 수정 후 실행 결과

첫 번째 동적 할당 해제는 잘 되지만 그 이후에는 실패하는 것을 확인 할 수 있다. 코드를 한 줄씩 읽으면서 생각해보자. While문 내부에서 temp 포인터에는 head 포인터가 가리키는 노드의 주소가 저장된다. 오리의 이름을 출력 한 후, free(temp)로 동적 메모리를 해제한 다음 head를 다음 노드로 이동시키는 것이 2차 수정본 코드이다. 이때 해제되는 메모리는 head가 가리키는 메모리이므로, head = head->next를 하는 과정에서 이미 해제된 메모리를 호출했기 때문에 오류가 발생한거 같다.

```

void end(Duck* head) {
    int count = 0;
    while (head != NULL) { // head가 NULL이 아닐 때까지 루프 실행
        Duck* temp = head; //동적할당을 해제하기 위한 임시 포인터
        printf("%s : 감사합니다!!\n", temp->name);
        head = head->next;
        free(temp);
        count++;
    }
    printf("%d마리의 오리를 살려주었습니다. 곧 프로그램이 종료됩니다...\n", count);
}
  
```

[그림 29] end함수 최종

따라서 위와 같이 현재 head포인터가 다음 노드를 가리키도록 head = head->next를 한 후, free(temp)를 사용하여 기존의 head 포인터가 가리키는 메모리를 해제하도록 코드를 수정하니 문제가 해결되었다.

사용자 정의 함수에서 발생하는 오류는 모두 수정하였으나 매우 근본적인 문제가 발생하였다. 메모리를 해제하여도 프로그램이 종료되지 않았다!!!

```
메뉴를 선택하세요(메뉴를 확인하시려면 6을 입력하세요) : 5
a : 살려주셔서 감사합니다!!
b : 살려주셔서 감사합니다!!
c : 살려주셔서 감사합니다!!
d : 살려주셔서 감사합니다!!
e : 살려주셔서 감사합니다!!
5마리의 오리를 살려주었습니다. 곧 프로그램이 종료됩니다...
메뉴를 선택하세요(메뉴를 확인하시려면 6을 입력하세요) : _
```

[그림 30] 프로그램 종료 안됨

제일 처음 end 함수를 설명할 때, 함수 자체로는 프로그램을 종료시키지 않는다고 하였는데 깜박하고 있었다. 입력 받은 수가 5번일 프로그램을 종료시키기 위해서 while문을 탈출하도록 main함수를 수정해야 했다. while(1)을 while(num!=5)로 바꾸어도 되지만 그렇게 될 경우, 모든 case문마다 printf와 scanf를 넣어야 하기 때문에 코드가 지저분해 보인다. 따라서 switch~case문 종료 후, num이 5면 break; 그렇지 않은 경우 새로운 num을 입력받도록 코드를 수정하였다.

```
default: // 잘못된 입력을 받을 경우
    printf("1에서 5까지 정수를 입력하세요\n\n");
    break;
}
printf("메뉴를 선택하세요(메뉴를 확인하시려면 6을 입력하세요) : ");
scanf("%d", &num);
return 0;
```

```
if (num == 5) // 5번을 고른 경우 프로그램 종료
    break;

else { // 그렇지 않은 경우 옵션을 입력받음
    printf("메뉴를 선택하세요(메뉴를 확인하시려면 6을 입력하세요) : ");
    scanf("%d", &num);
}
return 0;
```

[그림 31] (좌) 수정 전

(우) 수정 후

이외에도 kill 함수에서 return head;를 빼먹거나, makefile의 이름을 마음대로 설정하는 등 다양한 실수를 하였다.

```
남아있는 오리입니다
-----
[1번째 오리는 a입니다.]
[2번째 오리는 c입니다.]
[3번째 오리는 e입니다.]
-----

메뉴를 선택하세요(메뉴를 확인하시려면 6을 입력하세요) : 4
몇 번째 오리를 꺼낼까요? : 5

5번째 오리가 없습니다.
현재 잡힌 오리는 총 3마리 입니다.

남아있는 오리입니다
-----
세그멘테이션 오류 (코어 덤프됨)
byoengjae@byoengjae-VirtualBox: ~/Desktop/dnscp/hw1$
```

[그림 32] head를 return하지 않아서 오류 발생

이번 과제를 수행하면서 매우 많은 것을 배우고 느꼈다. 처음에 프로그램을 어떻게 만들어야 할지 막막했는데, 프로그램의 동작 과정을 생각하면서 **순서도를 그리는 과정에서 큰 틀을 구상**할 수 있었다. 큰 그림을 그리고 나니, 해야 할 것이 눈에 보였다. 과제에서 요구하는 기능을 수행하는 함수를 만들고, main 함수에서 필요할 때마다 호출하면 끝이었다. 하지만 자료구조 수업을 듣지 않았기에 linked list를 만드는 과정은 목록치 않았다. 보고서에 언급된 오류 이외에도 기억도 나지 않을 만큼 많은 수정과정을 거쳐서 프로그램을 완성하였다. 특히 포인터와 포인터, 노드와 노드가 얹혀있는 관계를 정리하는 과정이 힘들었는데, 머릿속으로 생각하는 것 보다 **그림을 그리는 것이** 매우 도움되었다.

앞에서 프로그래밍을 편하게 하려고 오리 이름을 나타내는 변수(또는 인자)는 모두 name으로 설정했다고 하였는데 **인자나 변수의 이름이 매우 중요**하는 것을 깨달았다. 디버깅을 할 때 Visual Studio를 사용하였기 때문에 name에 마우스를 갖다대면 멤버 변수 name인지, 함수의 인자 name인지 구별 하기 쉬웠는데 만약 shell에서 바로 코드를 짰다던가 남에게 코드를 설명해줘야 할 경우(보고서를 작성할 때 많이 느꼈다.) name과 name을 구별하기 어렵다는 문제가 발생한다. 따라서 다음부터 코드를 작성할 때는 이름을 적절히 설정하여 **누가 봐도 알아보기 쉽도록** 구별해야겠다고 다짐하였다. 마지막으로, 과제 수행을 위해 책을 읽고 많은 사이트를 돌아다니면서 구글링을 하였는데, 이 과정에서 **프로그래밍을 공부하는 방법**을 배울 수 있었다. 솔직히 힘들었지만(virtualbox의 화면이 확장되지 않거나, 한글이 입력되지 않는 등 잡다한 문제가 매우 많았다.) 이번 과제를 통해 한층 성장하였음을 느꼈다.

5. 참고자료

<https://modoocode.com/14> - vim 사용법

<https://reakwon.tistory.com/25> - 링크드 리스트

<https://app.diagrams.net/> -순서도 그리기

<https://guidey.tistory.com/50> - 동적 할당

<https://bowbowbow.tistory.com/12> - make파일 만들기

<https://klkl0.tistory.com/40> - 우분투 터미널 명령어

<https://coder-in-war.tistory.com/entry/C-17-%EC%97%B0%EA%B2%B0%EB%A6%AC%EC%8A%A4%ED%8A%B8-%EA%B5%AC%EC%A1%B0%EC%B2%B4-Linked-List> - 링크드 리스트

이것이 자료구조+알고리즘이다 with C 언어 (한빛미디어, 박상현)

