실습과제 3.1

[과제 설명]

구조체를 이용한 연결리스트로는 해당 노드에 저장할 자료를 넣는 data 부분과 다음 노드의 주소가 들어갈 포인터로 이루어진다.
연결리스트의 주요 연산으로 아래의 함수를 생성한다.

▪노드 생성: Node* Create_Node(int newData)
입력 받은 데이터를 저장할 노드를 생성하고, 생성된 노드의 주소를 반환하는 기능을 수행한다.
이 때 일반적인 구조체 변수로 선언하면 함수가 종료된 후에는 지역변수기 때문에 사라진다. 따라서 함수가 종료된 후에도 노드가 유지되게 동적할당을 사용한다. C언어로 구현했기 때문에 malloc을 사용했다. 이때, static을 사용해버리면 그 함수에만 붙어있기 때문에 쓸 수 없다.

▪노드 소멸: void Destroy_Node(Node* node)
입력 받은 노드를 소멸시키는 기능을 수행한다.
노드를 삭제할 때 동적할당을 이용했기 때문에 단순히 연결을 끊어서는 안된다. 단순히 한쪽 link를 NULL로 만들면 그 부분만 메모리상에서 (사용할 수도 없고) 떠버린다. 그렇기 때문에 삭제하려는 부분의 메모리를 해제하고 (삭제시키려던 주소가 담긴) link를 NULL로 만들어야 한다.

▪노드 추가: void Append_Node(Node** head, Node* newNode)
연결리스트의 제일 뒷부분에 새로운 노드를 추가하는 기능을 수행한다.
이 때, 시작주소를 보고 추가할 때 link가 NULL인 곳을 계속 탐색하고 나서 제일 뒷부분이 나왔을 때 새 노드를 추가해야 한다. 그래서 현재 head가 비어있다면 새로운 노드를 추가시키는 예외처리가 들어가는 것이다.

▪노드 출력: Print_Linked_List(Node* head)
입력 받은 주소과 반복자를 이용해 노드의 데이터를 출력하는 기능을 수행한다.
반복자 iter를 따로 지정하는 이유는, 노드의 link를 기준으로 출력하면 마지막의 데이터가 출력되지 않아서 별도로 지정해야 하기 때문이다. 반복자를 따로 변수로 선언해서 사용하면 마지막 데이터까지 출력할 수 있다.

[코드]

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
  int data; //데이터 필드
  struct Node* link; //다음 노드의 주소를 저장
}Node;

Node* Create_Node(int newData) { //노드 생성 함수
  Node* newNode = (Node*)malloc(sizeof(Node));
  //함수가 종료된 후에도 노드가 유지되게 동적할당을 사용
  newNode->data = newData;
  newNode->link = NULL; //쓰레기 값 대신 null로 한다.

  return newNode; //노드의 주소값 반환
}

void Destroy_Node(Node* node) { //입력받은 노드를 소멸시킨다.
  if (node != NULL)
    free(node);
  //해당 노드의 주소가 null이 아니라면 해당 메모리를 해제한다.
}

void Append_Node(Node** head, Node* newNode) {
  //연결리스트의 제일 뒷부분에 새로운 노드를 추가하는 기능

  if ((*head) == NULL) //현재 head가 비어있다면
    *head = newNode; //새로운 노드를 head로 지정
  else { //비어있지 않다면
    Node* tail = (*head);
    while (tail->link != NULL) {
      tail = tail->link;
    } //비어있는 노드까지 탐색
    tail->link = newNode; //제일 끝부분에 새로운 노드 추가
  }
}

void Print_Linked_List(Node* head) {
  Node* iter = head;
  int i = 0;
  while (iter != NULL) {
    printf("node[%d]: %d", i, iter->data);
    iter = iter->link;
    if (iter != NULL) printf(" → ");
    i++;
  } //반복자가 비어있을 때까지 탐색하면서 하나씩 출력한다.
  puts("");
}
```

```
int main(void) {
  Node* head = NULL;
  Node* newNode = NULL;

  newNode = Create_Node(15); // 15노드 생성 및 추가
  Append_Node(&head, newNode);

  newNode = Create_Node(31); //31노드 생성 및 추가
  Append_Node(&head, newNode);

  newNode = Create_Node(3); //3노드 생성 및 추가
  Append_Node(&head, newNode);

  //연결리스트 전체 출력
  Print_Linked_List(head);
  return 0;
}
```

[과제에 대한 고찰]

노드를 제대로 배우기 전에는 배열도 동적할당을 이용해서 메모리를 타이트하게 쓸 수 있는데 왜 굳이 노드를 사용할까 하는 생각을 했다. 하지만 배열을 동적으로 할당하든, 정적으로 할당하든 배열은 메모리에 선형으로 된 공간이 비어있어야 가능한데, 노드는 불규칙적으로 띄엄띄엄 남아있어도 다음 메모리의 주소를 알고 있기 때문에, 메모리가 매우 부족한 상황에서 더 효율적으로 메모리를 사용할 수 있는 장점이 있다는 것을 알게 되었다.

또한 실습을 하면서 노드에 대한 다른 내용도 찾아보았는데, 구조체를 만들 때 자기 참조 구조체에 포인터 변수 대신 자기 자신을 그대로 적으면 왜 오류가 나면서 빌드가 되지 않는 것에 대한 것이었다.
그것은 중괄호가 닫히기 전에는 구조체에 대한 선언이 끝났지 않기 때문에 전체 구조체의 크기를 알 수 없는 것에 비해 포인터 변수는 어떤 변수의 주소를 가리키고 있든 간에 똑같은 크기를 가지고 있기 때문이라는 것을 알게 되었다.