고려대학교 ALPS 2021

C언어 스터디

Week 0x06

이중 포인터, 구조체

Contents

0x01 Double Pointer

이중 포인터

이중 포인터의 개념에 대해서 알아봅시다.
지금은 굳이 포인터를 두번이나 쓸 일이 있을까? 라는 생각이 드실 수도 있겠지만,
구조체를 사용해서 자료구조 수업 때 자료구조 구현을 하거나 할 때 쓰실 일이 있으실 거고,
그러한 상황에서 개념을 확실히 알아두시는 것이 큰 도움이 되실 것이라고 생각합니다!

이중 포인터는 포인터를 가리키는 포인터입니다.
포인터를 사용하는 이유는
1. call-by-reference로 함수를 호출하기 위해,
2. 동적 할당을 사용하기 위해.
두가지가 있었습니다.

이중 포인터 역시 같습니다.
call by reference로 함수를 호출하는 이유는,
함수에서 값을 바꾸기 위함이었습니다.

그렇다면 포인터 값, 즉 변수의 주솟값을 바꾸려면 어떻게 해야할까요?

0x01 Double Pointer

이중 포인터

포인터 값을 인자로서 함수를 호출하는 call-by-reference 방식도
원론적으로는 값을 복사하는 방식이기에,
변수의 주소값을 복사하여 함수 내에서 사용합니다.

그렇다면, 주소값을 저장하는 포인터 값 자체를 바꾸려면 포인터 값을 주면 안되고,
해당 포인터의 메모리 주소를 인자로 불러줘야 함수에서 포인터 값을 바꿀 수 있겠죠.

이중 포인터는 단순하게 포인터의 포인터
이기 때문에 *을 하나 더 붙여서 선언하면
됩니다.

(int) *(pointer);
(int *) *(doublePointer);
의 느낌으로 생각하시면 됩니다.

역참조 역시 비슷한 방식으로
*을 두개 붙이면 되겠죠.

```
int num = 1905;
int *pointer = &num;
changeValue(num); // 값 변경 불가!
changeValue(pointer); // 값 변경 가능!

int **doublePointer = &pointer; // 'pointer'의 주소 저장
changePointer(pointer); // pointer 값 변경 불가
changePointer(doublePointer); // 값 변경 가능!
```

0x01 Double Pointer

이중 포인터

동적 할당 시 포인터를 이용하여 이뤄지는 경우가 많으므로

동적 할당된 자료들을 다루거나, 2차원 배열을 만들거나 할 때에도 이중 포인터가 사용됩니다.

개념만 정확히 알아두시면 수울 듯 합니다.

0x02 Structure

구조체

이번에는 구조체에 대해 알아봅시다.
구조체를 통해 프로그래머가 원하는 자료형의 집합을 만들 수 있습니다.

예를 들어서 대학생들을 다루는 프로그램을 만든다고 합시다.
그렇다면, 이름, 학번, 나이 등등의 정보를 담아야 할 것입니다.

물론, 학생별로 번호를 매겨서, 이름이 저장된 문자열 배열,
학번이 저장된 숫자 배열, 나이가 저장된 숫자 배열… 등등 여러 가지 배열을 선언하면,
따로따로 배열에 저장되어 있기에 접근하기가 몹시 불편하겠죠.

그런데, 구조체로서 이 정보들을 묶어서 표현할 수 있다면,
구조체 배열만 선언하면 되기 때문에 편리하게 데이터들을 관리할 수 있을 것이고,
해당 구조체를 대상으로 하는 함수도 선언할 수 있기에 더욱 편리하고 체계적으로
프로그램을 짤 수 있습니다.

이러한 것들이 바로 자료구조와 구조체를 이용하는 이유라고 할 수 있겠습니다.

0x02 Structure

구조체

그렇다면 구조체를 어떻게 선언하는지 알아보도록 합시다.
오른쪽과 같이 struct라는 키워드를 통해 구조체를 선언할 수 있습니다.
이렇게 선언을 하면, 여러분들이 int, char 등의 자료형처럼,

```
struct student{
    char name[30];
    int studentID;
    int age;
}
```

char[30], int, int 자료형을 묶은 'struct student'라는
새로운 자료형을 만든 것이라고 할 수 있습니다.
자료형을 만들었으면 사용을 해야겠죠?

사용 방법도 여타 다른 자료형들과 같습니다.
오른쪽과 같이 선언을 할 수 있고,
초기화를 하지 않은 변수처럼 지금 값이 들어있지 않은 상태입니다.

```
struct student jeonghoon;
```

구조체 내부의 값들에는 '.' 즉 Dot 연산자를 이용해 접근가능합니다.

```
jeonghoon.studentID = 2020320094;
jeonghoon.age = 20;
strcpy(jeonghoon.name,"jeonghoon");
```

0x02 Structure

구조체

그런데 구조체 변수 선언 시 struct 를 앞에 붙이는게 상당히 더럽습니다.
이러한 단점을 해결하는 것이 'typedef' 키워드입니다.

typedef는 type define, 즉 타입 정의의 약자로,
프로그램에서 쓰이는 자료형 이름에 다른 이름을 지어주는 키워드입니다.

오른쪽과 같이 struct student라는 타입에
Student 라는 별칭을 지어주면, struct student 대신
Student 사용 가능합니다. (원래 이름과 달라야 함)

```
typedef struct student Student;
struct student jeonghoon;
Student seungmin;
```

구조체 선언 시 사용

```
typedef struct Student{
  char name[30];
  int studentID;
  int age;
} student;
```

구조체 이름 생략

```
typedef struct{
  char name[30];
  int studentID;
  int age;
} student;
```

일반 자료형에도 사용 가능

```
typedef long long ll;
typedef int i;
```

0x02 Structure

구조체

그런데 구조체 변수 선언 시 struct 를 앞에 붙이는게 상당히 더럽습니다.
이러한 단점을 해결하는 것이 'typedef' 키워드입니다.

typedef는 type define, 즉 타입 정의의 약자로,
프로그램에서 쓰이는 자료형 이름에 다른 이름을 지어주는 키워드입니다.

오른쪽과 같이 struct student라는 타입에
Student 라는 별칭을 지어주면, struct student 대신
Student 사용 가능합니다. (원래 이름과 달라야 함)

```
typedef struct student Student;
struct student jeonghoon;
Student seungmin;
```

구조체 선언 시 사용

```
typedef struct Student{
  char name[30];
  int studentID;
  int age;
} student;
```

구조체 이름 생략

```
typedef struct{
  char name[30];
  int studentID;
  int age;
} student;
```

일반 자료형에도 사용 가능

```
typedef long long ll;
typedef int i;
```

0x02 Structure

구조체

포인터를 이용해서 구조체를 가리키는 경우가 있습니다.
이럴 때에는 . 연산자 대신 ->, 화살표 연산자를 사용합니다.

```
typedef student* studentP;
studentP studentPtr = &jeonghoon;
(studentPtr->age)++;
(*studentPtr).age++;
```

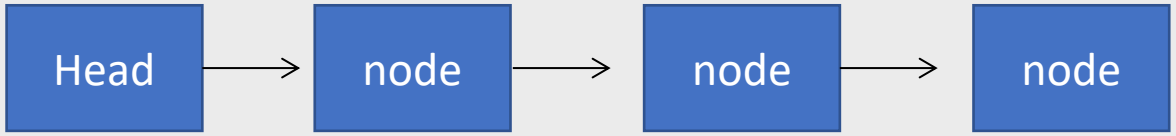
이러한 방식으로 구조체에 다양한 방법으로 접근이 가능합니다.
지금까지 배웠던 구조체, 포인터(특히 이중 포인터) 를 이용해
Linked List, Tree, Stack, Queue 등
다양한 자료 구조를 구현할 수 있습니다.

이러한 자료구조들은 자기참조 구조체 Self-referential structure의 형식이며
오늘 실습에서는 간단하게 링크드 리스트를 구현해보도록 하겠습니다.

0x03 Linked List Practice

링크드 리스트 실습

자기 참조 구조체는 구조체를 구성하는 멤버 중
그 해당 구조체 타입을 가리키는 포인터가 존재하는 구조체를 말합니다.
이러한 방식을 통해 같은 타입의 구조체들을 연결할 수 있습니다.



트리나 그래프 등에서 데이터 단위 하나를 보통 노드라고 부릅니다.

링크드 리스트는 배열과 비슷한 구조입니다.
배열은 일자로 연결된 메모리 주소를 가지며 인덱스를 통해 값에 접근하는 것은 쉬우나
배열 값 사이 사이에 값을 추가하거나 삭제하는 것이 어렵다는 단점이 있습니다.

링크드 리스트는 메모리 주소가 독립적이기에 인덱스가 없어, 값에 접근하는 데에는
포인터를 통해 접근해야하기 때문에 시간이 배열보다는 걸린다는 단점이 있으나,
리스트 사이사이에 값을 추가하거나 삭제하는 것이 쉽습니다.

0x03 Linked List Practice

링크드 리스트 실습

그렇다면 해당 자료구조의 구조체를 정의해봅시다.
먼저 링크드 리스트의 데이터 단위가 될 Node 라는 구조체를 만들것이고,
그 Node들을 감싸는 List라는 구조체를 만들겠습니다.

Node 구조체는 int형 데이터와 다음 Node의 메모리 주소를 저장하는 것으로 하고,
List 구조체는 가장 첫 Node와 노드 개수를 저장하는 것으로 합시다.

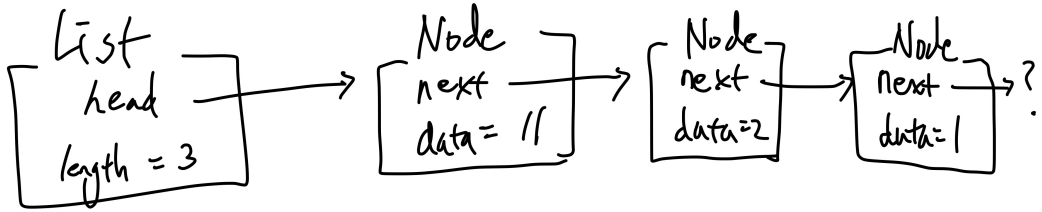
그렇다면 다음과 같이 구조체를 만들어 볼 수 있겠습니다.

```c
typedef struct node {
    int data;
    struct node* next;
} Node;
```

```c
typedef struct {
    Node* head;
    int length;
} List;
```

구조체 내부에서 Node 구조체를 참조하는 포인터 멤버가 있으므로

구조체 이름을 생략하지 않아야 합니다.



0x03 Linked List Practice

링크드 리스트 실습

이제 링크드 리스트를 다룰 수 있는 함수들을 선언해 봅시다.

```
List* makeList(){
  List* newList = (List*)malloc(sizeof(List));
  if(newList){
    newList->length = 0;
    newList->head = NULL;
  }
  return newList;
}
```

```
int insertNode(List* list, int data){
  Node* currentHead = list->head;
  Node* newNode = (Node*)malloc(sizeof(Node));

  if(!newNode) return 0;
  newNode->data=data;
  newNode->next=currentHead;

  list->head=newNode;
  list->length++;
  return 1;
}
```

```
Node* searchNode(List* list, int value){
  Node* current = list->head;
  while(current){
    if(current->data == value) break;
    current = current->next;
  }
  return current;
}
```

```
void destroyList(List* list){
  Node* current = list->head, next;
  while(current){
    next=current->next;
    free(current);
    current=next;
  }
  free(list);
}
```

ASCII 코드는 0, 유니코드는 U+0000이 널 문자다.[2]

C 프로그래밍에서는 '존재하지 않는 메모리 주소'를 NULL로 나타낸다. 특정 포인터 변수를 초기화할 때는 실제로 메모리를 할당하기 전에 NULL로 초기화하는 버릇을 들이자. 이런 식으로

```
char *ptr = NULL;
ptr = (char*)malloc(...);
```

포인터 변수와 관련된 연산에서 값이 0인 정수 상수식이 나오면 이를 곧 NULL로 해석한다(사실 NULL은 0, 혹은 (void*)0과 같도록 정의되어야 한다). 따라서 위의 코드는 아래와 같이 써도 된다.

```
char *ptr = 0;
ptr = (char*)malloc(...);
```

NULL == 0 == false

부족한 수업
들어 주시느라 수고 많으셨습니다!

들어주셔서 감사하고, 질문은 언제든 환영이에요

다음 주에 봅시다~