

자료 구조

<https://youtu.be/DIyWTIZO8bI>

송경주

목차

- 자료 구조와 알고리즘
- 순환
- 배열, 구조체, 포인터

자료 구조와 알고리즘

- 자료 구조란?

프로그램에서 자료들을 정리하는 여러가지 구조
(리스트, 스택, 큐, 트리 등등)

- 알고리즘이란?

주어진 문제를 처리하는 절차

추상 데이터 타입

- 새로운 데이터 타입을 **추상적으로** 정의한 것
- 데이터 타입의 구현으로부터 **분리된 데이터 타입**.

→**자료구조**는 이러한 추상 데이터 타입을 프로그래밍 언어로 구현한 것

추상 데이터 타입

인터페이스 제공



- 사용자에게 추상 데이터 타입이 제공하는 인터페이스만을 제공
- 추상 데이터 타입이 수정되더라도 제공하는 인터페이스는 바뀌지 않음.
- 사용자는 추상 데이터 타입의 내부를 볼 수 없음.
- 사용자는 추상 데이터 타입이 제공하는 연산을 어떻게 사용하는지를 알아야 함.

알고리즘의 성능 분석

-상용 프로그램의 규모가 이전에 비해 엄청나게 커지고 있기 때문에 알고리즘의 효율성이 중요. 자료의 양이 많아지면 그 차이는 상당함.

[분석 방법]

- ① 실행 시간 측정 방법
- ② 알고리즘의 복잡도 분석 방법

1. 실행 시간 측정 방법

-알고리즘을 프로그래밍 언어로 작성하여 실제 컴퓨터상에서 실행시킨 다음,
그 실행 시간을 측정하는 것

(clock 함수 사용 → clock함수: 호출 프로세스에 의하여 사용된 CPU시간 계산)

[장점]

- ① 정확하고 확실한 방식임.

[단점]

- ① 복잡한 경우 알고리즘을 구현하고 테스트하기 어려움.
- ② 2개의 알고리즘을 비교하려면 반드시 똑같은 하드웨어를 사용하여 실행시간을 측정해야 함.
- ③ 같은 언어를 사용하여 비교해야 함.

2. 알고리즘의 복잡도 분석 방법

-구현하지 않고 알고리즘의 효율성을 따져보는 기법.

(시간 복잡도: 알고리즘의 실행 시간 분석, 기억 복잡도: 알고리즘이 사용하는 기억 공간 분석)

→알고리즘의 복잡도를 얘기할 때 보통 시간 복잡도를 말한다. (차지하는 공간보다 실행 시간에 더 관심이 있기 때문)

[장점]

- ① 구현하지 않고서도 대략적으로 알고리즘의 효율성을 비교할 수 있다.
- ② 실행 하드웨어나 소프트웨어 환경과는 관계없이 알고리즘의 효율성을 평가할 수 있다.

[단점]

- ① 대략적인 방식임.

시간 복잡도

[시간 복잡도]

연산이 몇 번이나 실행되는지를 숫자로 표시
연산의 실행 횟수를 사용하여 복잡도 분석
(입력 개수 n 에 따라 변하게 됨.)

→ 시간 복잡도 함수 $T(n)$

알고리즘 1

$$3n + 1$$

알고리즘 2

$$5n^2 + 1$$

$T(n)$

입력의 개수 n 과 시간 복잡도 함수 $T(n)$ 의 관계가 상당히 복잡할 수 있다. 이때 사용할 수 있는 여러 표기법이 있다 (빅오, 빅오메가, 빅세타)

빅오 표기법 : 자료의 개수가 많은 경우에는 가장 큰 항이 가장 영향을 크게 미치고 다른 항들은 상대적으로 무시될 수 있다.

Ex)

$$2n^2 + 1 \rightarrow O(n^2)$$

$$3n + 5 \rightarrow O(n)$$

순환

-어떤 알고리즘이나 함수가 자기 자신을 호출하여 문제를 해결하는 프로그래밍 기법.

[내부적인 구현]

1. 복귀주소가 시스템 스택에 저장되고 호출되는 함수를 위한 매개 변수와 지역 변수를 스택으로부터 할당 받음. → 이러한 스택에서의 공간 : 활성레코드
2. 호출된 함수의 코드의 시작 위치로 점프하여 수행을 시작. (호출된 함수가 자기 자신이라면 자기 자신의 시작 위치로 점프)
3. 호출된 함수가 끝나면 시스템 스택에서 복귀 주소를 추출하여 호출한 함수로 되돌아감.

순환

순환 알고리즘은 자기 자신을 순환적으로 호출하는 부분과 순환 호출을 멈추는 부분으로 구성되어 있음.

```
int Factorial
```

```
{
```

```
    if(n <= 1) return 1
```

순환을 멈추는 부분

```
    else return n * factorial (n-1)
```

순환 호출을 하는 부분

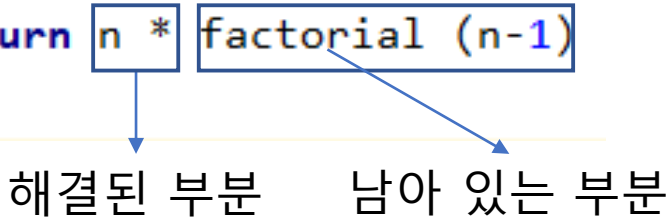
```
}
```

순환

-동일한 문제들로 분해하여 해결하는 방법을 분할 정복이라 하는데, 중요한 것은 순환 호출이 일어날 때마다 문제의 크기가 작아진다는 것.

```
int Factorial
{
    if(n <= 1) return 1

    else return n * factorial (n-1)
}
```



해결된 부분

남아 있는 부분

순환을 사용하는 이유?

-순환으로 구현되는 것은 반복문으로도 구현할 수 있다. 그런데 순환을 사용하는 이유는?
(각 구현할 문제마다 반복문이 더 효율적일때도 있고 순환이 효율적일 때도 있다.)

[순환]

순환적인 문제나 자료 구조를 다루는 문제를 해결하는 데에 적합.

반복에 비해 알고리즘을 훨씬 명확하고 간결하게 나타낼 수 있음.
(가독성이 좋음)

반복에 비해 수행속도는 떨어짐.

[반복]

순환에 비해 수행속도가 빠름.

기억 공간의 사용에 효율적임.

배열

-같은 형의 변수를 여러 개 만드는 경우 사용 (타입이 같은 데이터 묶기)

[함수의 매개 변수로서의 배열]

배열의 이름은 포인터와 같은 역할을 함

→ 배열의 이름을 함수의 매개변수로 전달하여 값을 수정하면 main()에 영향을 미친다.

구조체

-타입이 다른 데이터 묶기

typedef 사용시 구조체를 새로운 타입으로 선언하는 것이 가능

[구조체 사용]

1. 구조체 형식 정의
2. 구조체 선언

```
typedef struct 구조체명 {  
    항목 1;  
    항목 2;  
    ...  
}
```

포인터

-포인터 변수: 다른 변수의 주소를 가지고 있는 변수

[사용 방식]

1. main()함수의 변수값을 바꾸고 싶을 때 포인터를 매개변수로 전달해 값을 바꿀 수 있다.
2. 함수에서 하나 이상의 값을 반환해야 할 때 사용. (C언어에서 return문장은 하나의 값만을 반환할 수 있음.)

포인터

[배열과 포인터]

배열의 이름이 배열의 시작부분을 가리키는 포인터. (메모리 공간 절약)

→ 배열의 이름 전달이 곧 포인터의 전달

[구조체와 포인터]

구조체 자체를 매개변수로 넘기는 경우, 구조체가 큰 경우에는 상당한 부담이 됨.

→ 구조체 포인터를 넘겨 주소만 함수에 전달

포인터에 대한 연산

※ $pi+1$, $pi-1$ 의 값은?

주소 값이 하나 감소되고 증가하는 것이 아니라

$pi-1$: pi 가 가리키는 객체 하나 뒤에 있는 객체를 가르킴. 즉, $A[2]$

$pi+1$: pi 가 가리키는 객체 하나 앞에 있는 객체를 가르킴. 즉, $A[4]$

❖ $(*pi)++$: 포인터가 가리키는 값을 증가시킴

```
int A[6], int *pi ;  
pi = &A[3];
```

동적 메모리 할당

-프로그램 실행 도중에 동적으로 메모리를 할당 받는 것.

메모리 할당 방식에는 정적 할당과 동적 할당 두가지 방식이 있다.

[동적 메모리 할당]

프로그램 실행 도중 메모리 할당.

장점 - 필요한 만큼 할당 받기 때문에 메모리 낭비가
없고 효율적으로 사용 가능.

[정적 메모리 할당]

프로그램 시작 전 결정됨.
프로그램 실행 도중 크기 변경 불가.

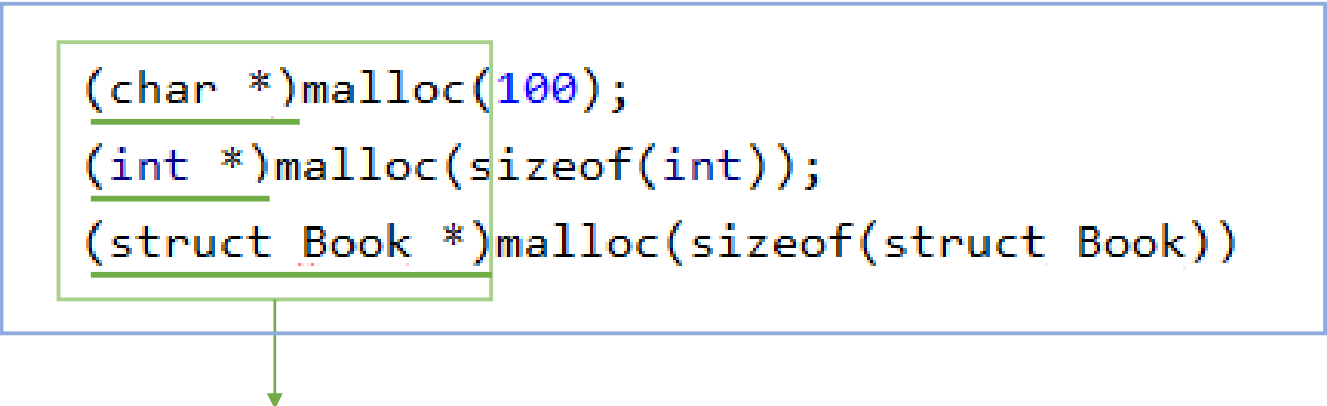
장점 - 간단하게 메모리 할당 가능.
단점 - 메모리가 부족할 경우 고정된 크기 변경 불가,
남는 경우 메모리 낭비

동적 메모리 할당

`void *malloc(int size)`

Size 바이트만큼의 메모리 블록을 할당함. 새로운 메모리 블록의 시작 주소를 반환함. 반환되는 포인터는 `void *`

```
(char *)malloc(100);  
(int *)malloc(sizeof(int));  
(struct Book *)malloc(sizeof(struct Book))
```



`malloc()`은 블록의 시작주소(포인터)를 `void *`로 반환하므로 적절한 타입의 포인터로 타입변환을 해야 한다.