제1장 자료구조를 배우기 위한 준비

자료구조, ADT, 성능분석

1-1 자료구조

- **자료구조(Data Structure)**
  - 일련의 동일한 타입의 데이터를 정돈하여 저장한 구성체
- **데이터를 정돈하는 목적**
  - 프로그램에서 저장하는 데이터에 대해
    탐색, 삽입, 삭제 등의 **연산**을 효율적으로 수행하기 위해
- **자료구조를 설계할 때에는**
  데이터와 데이터에 관련된 연산들을 함께 고려해야

시스템 생명 주기(System Life Cycle)

- **1. 요구사항(requirement)**
  - 프로젝트 목적을 정의한 명세 집합인 사용자 요구 사항 추출 단계. 입력과 출력 명시.
- **2. 분석(analysis)**
  - 문제들을 다룰 수 있는 작은 단위들로 나눔
  - 추출된 요구의 정확성, 완전성 등을 확인하여 사용자의 요구를 확정
- **3. 설계(design)**
  - 사용자의 요구를 소프트웨어 시스템의 구조로 변환하는 단계
  - 소프트웨어 구조/사용자 인터페이스/자료구조(ADT)/알고리즘을 설계
- **4. 정제와 코딩(refinement and coding)**
  - 데이타 객체에 대한 표현 선택 / 수행되는 연산에 대한 알고리즘 작성
  - 설계 문서를 프로그래밍 언어를 사용하여 source code로 변환
  - 구현 단계: 문제 이해 → 알고리즘 설계 → 코딩(coding) → 오류 제거(debugging)
- **5. 검증(verification)**
  - 테스트(testing):프로그램의 정확한 수행 검증 및 성능 검사
    - ex. 모든 if 문과 switch 문의 case를 확인하고 있는가?
  - 오류 제거(error removal) :독립적 단위로 테스트 후 전체 시스템으로 통합

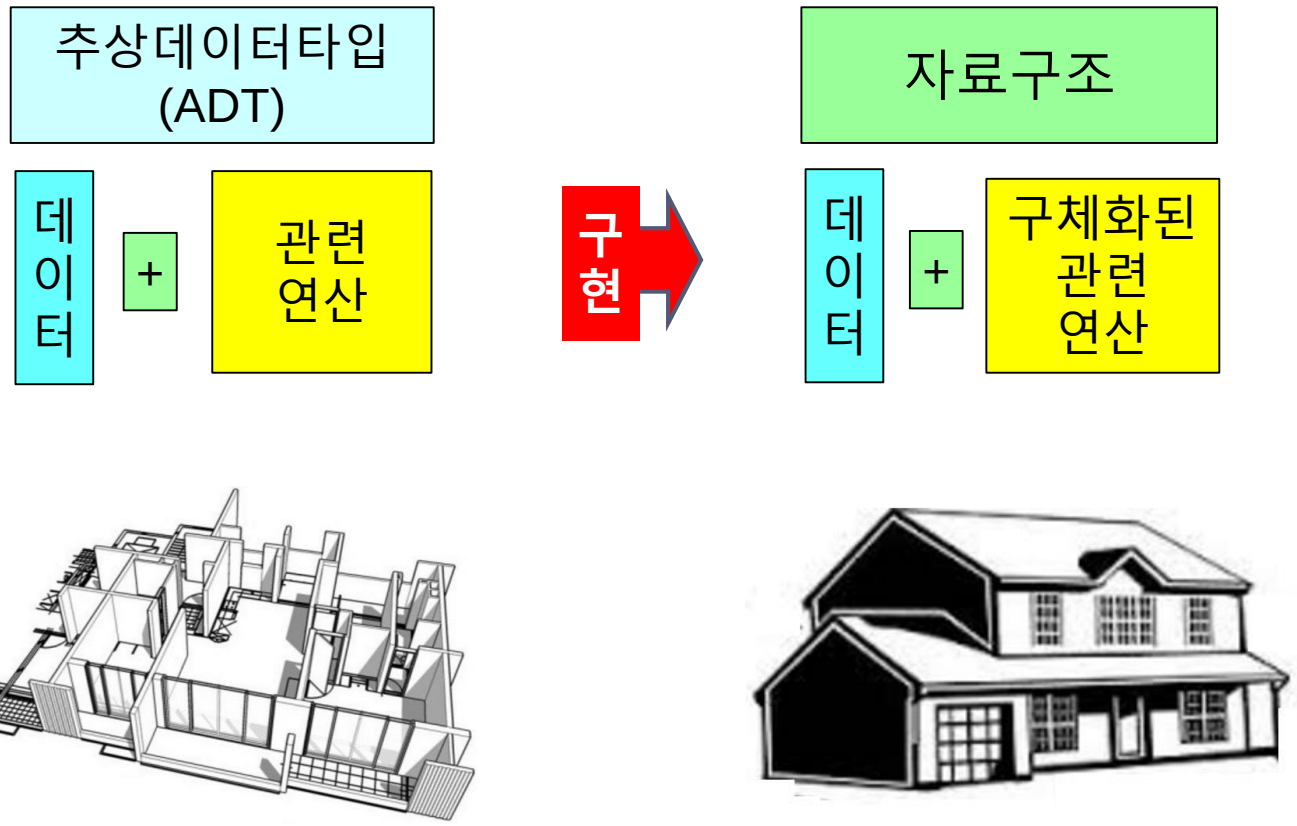
알고리즘 명세

- **알고리즘 (algorithm)**
  - 특정 작업을 수행하는 명령어들의 유한 집합
- **알고리즘의 요건**
  - 입력(input) : 외부에서 제공되는 데이타가 0개 이상
  - 출력(output) : 적어도 한 개 이상의 결과 생성
  - 명확성(definiteness) : 각 명령은 명확하고 모호하지 않아야 함
  - 유한성(finiteness) : 알고리즘을 수행하면 어떤 경우에도 반드시 종료
  - 유효성(effectiveness) : 반드시 실행 가능해야 함
- **알고리즘 기술 방법**
  - 자연어
  - 흐름도(flowchart)
  - 프로그래밍 언어

추상데이터타입

- **추상데이터타입(Abstract Data Type)**
  - 데이터와 그 데이터에 대한 추상적인 연산들로 구성
- **'추상'의 의미**
  - 연산을 구체적으로 어떻게 구현하여야 한다는 세부 명세를 포함하고 있지 않다는 의미
- **자료구조는 추상데이터타입을 구체적(실제 프로그램)으로 구현한 것**

추상데이터타입과 자료구조 관계



1-2 성능 분석과 측정 (1)

- **좋은 프로그램이란?**
- **정성적 평가 기준 (Qualitative Criteria)**
  - 프로그램이 사용자의 요구에 부합하는가?
  - 프로그램이 정확하게 작동하는가?
  - 프로그램 내부에 문서화가 잘 되어 있는가?
  - 프로그램의 논리적인 단위를 생성하기 위해서 함수를 효과적으로 사용하고 있는가?
  - 프로그램의 코드가 읽기 쉬운가?
- **정량적 평가 기준 (Quantitative Criteria)**
  - 프로그램이 주기억장치와 보조기억장치를 효율적으로 사용하는가?
  - 프로그램의 실행시간은 적절한가?

성능 분석과 측정 (2)

- **성능 평가(performance evaluation)**
  - 성능 분석 (performance analysis)
    - 사전 예측. 복잡도 이론에 기초
    - **공간복잡도(space complexity)**
      - 프로그램을 실행시켜 완료하는데 필요한 메모리 공간의 크기
    - **시간복잡도(time complexity)**
      - 프로그램을 실행시켜 완료하는데 필요한 CPU 시간
  - 성능 측정 (performance measurement)
    - 이후 검사. 실제 실행 시간을 측정
    - 프로그래머의 숙련도, 구현에 사용된 프로그래밍 언어의 종류 그리고 알고리즘을 실행한 컴퓨터의 성능에 따라서 수행시간은 얼마든지 달라질 수 있기 때문에 선호되지 않음

공간 복잡도 (1)

- **공간 복잡도(space complexity)**
  - 프로그램을 실행시켜 완료하는 데 필요한 메모리 양
  - 프로그램 P의 공간 요구 S(P) = c + SP
    - c : 상수
    - SP : 인스턴스 특성. 문제에 의존. 인스턴스로 인해 필요로 하는 공간
- **고정 부분 : 프로그램의 입출력의 크기와 관계없는 메모리 요구량**
  - 프로그램 코드를 저장하기 위한 공간 (명령어 공간)
  - 단순 변수 또는 고정 크기의 집합체 변수를 위한 공간
  - 상수들을 저장하기 위한 공간
- **가변 부분: 프로그램의 입출력에 따라 변동하는 메모리 요구량**
  - 변수, 참조된 변수가 필요로 하는 공간, 순환 스택 공간
  - n개의 원소를 갖는 배열을 입력으로 사용하는 프로그램
    - n명의 학생에 대하여 어떠한 업무를 처리하는 문제
  - 입출력의 크기가 n인 프로그램
    - 1부터 n까지의 합을 계산하는 문제
    - n!의 값을 계산하는 문제
- **메모리 가격의 하락으로 인하여 공간복잡도의 중요성은 감소**

공간 복잡도 (2)

```
float Sum(float a[ ], const int n)
{
   float s = 0;
   for(int i = 0 ; i < n ; i++)
      s+= a[i];
   return s;
}
```

- 고정크기 공간만 필요하므로 S(Sum) = 0

```
1  float Rsum(float a[ ], const int n)
2  {
3     if(n<=0) return 0;
4     else return(Rsum(a, n-1) + a[n-1]);
5  }
```

- RSum을 호출할 때마다 적어도 네 개의 워드
  (n과 a 값, 반환 값, 반환 주소에 필요한 공간 포함)
- 순환 깊이가 n+1이므로 순환 스택 공간에 4(n+1)

시간 복잡도 (1)

- **정의: 프로그램을 실행시켜 완료하는데 소요되는 총 시간의 추산량**
  - 컴파일 시간 + 실행 시간
  - 컴파일 시간: 고정 시간 요구
  - 실행 시간: 입출력의 크기에 따라 변동되므로,

    기본적인 연산 횟수를 입력 크기의 함수로 표현
    - 기본 연산(Elementary Operation)이란 탐색, 삽입이나 삭제와 같은 연산이 아닌, 데이터 간 크기 비교, 데이터 읽기 및 갱신, 숫자 계산 등과 같은 단순한 연산

  - T(P) = 컴파일 시간 +실행시간(t_P(인스턴스 특성))

  - $t_P(n) = c_aADD(n)+c_sSUB(n)+c_mMUL(n)+c_dDIV(n) + \cdots$
    - n : 인스턴스 특성
    - C_a, C_s, C_m, C_d : +, -, x, / 연산을 위한 상수 시간
    - ADD, SUB, MUL, DIV : 특성 n인 인스턴스에서 각 연산의 실행 횟수

시간 복잡도 (2)

- **프로그램 단계 수(number of steps)**
  - 실행시간이 인스턴스 특성에 상관없이 구문적으로나 의미적으로 뜻을 갖는 프로그램 세그멘트
  - 프로그램 세그멘트마다 소요되는 실행시간은 서로 다름
    - 1) a = 2;
    - 2) a = 2*b+3*c/d-e+f/g/a/b/c;
  - 주석 : 0
  - 선언문 : 0
    - 변수, 상수 정의(int, long, short, char,...)
    - 사용자 데이타 타입 정의(class, struct, union, template)
    - 접근 결정(private, public, protected, friend)
    - 함수 타입 결정(void, virtual)
  - 산술식 및 지정문 : 1
    - 예외 : 함수 호출을 포함하는 산술식

시간 복잡도 (3)

- **프로그램 단계 수(number of steps)**
  - 반복문 : 제어 부분에 대해서만 단계수 고려
  - 스위치 명령문
    - switch(<expr>)의 비용 = <expr>의 비용
    - 각 조건의 비용 = 자기의 비용 + 앞서 나온 모든 조건의 비용
  - if-else 문
    - <expr>, <statement1>, <statement2>에 따라 각각 단계수가 할당
  - 함수 호출
    - 값에 의한 전달 인자 포함하지 않을 경우 : 1
    - 값에 인한 전달 인자 포함할 경우 : 값 인자 크기의 합
    - 순환적인 경우 : 호출되는 함수의 지역 변수도 고려
  - 메모리 관리 명령문 : 1
  - 함수 명령문 : 0
    - 비용이 이미 호출문에 할당
  - 분기 명령문 : 1

단계 수 테이블

- **명령문의 실행당 단계 수(s/e : step per execution) 결정**
  - s/e : 그 명령문의 실행 결과로 count가 변화하는 양
- **명령문이 실행되는 총 횟수 계산**

```
line float Sum(float a[ ], const int n)
1   {
2       float s = 0;
3       for(int i = 0 ; i < n ; i++)
4           s+= a[i];
5       return s;
6   }
```

행 번호	s/e	빈도	단계 수
1	0	0	0
2	1	1	1
3	1	$n+1$	$n+1$
4	1	n	n
5	1	1	1
6	0	1	0
	총 단계 수		$2n+3$

네 종류의 성능 분석

- **최악의 경우 분석(Worst-case Analysis)**
  - '어떤 입력이 주어지더라도 알고리즘의 수행시간이 얼마 이상은 넘지 않는다' 라는 상한(Upper Bound)의 의미
  - 일반적으로 알고리즘의 수행시간은 최악경우 분석으로 표현
- **평균 경우 분석(Average-case Analysis)**
  - 입력의 확률 분포를 가정하여 분석하는데, 일반적으로 균등분포(Uniform Distribution)를 가정
- **최선의 경우 분석(Best-case Analysis)**
- **상각분석(Amortized Analysis)**
  - Amortize: "분할 상환하다". 상각(償却): "보상하여 갚아주다"
  - 일련의 연산을 수행하여 총 연산 횟수를 합하고 이를 연산 횟수로 나누어 수행시간을 분석

등교 시간 분석의 예

- 상황
  - 집을 나와 버스 정류장까지 5분, 버스 배차 간격은 20분,
    버스 타고 안 막히면 학교까지 30분, 강의실까지는 걸어서 10분

Best Case	Worst Case	**Average Case**
집을 나와서 운이 좋게 바로 버스를 타고 막히지 않은 도로상태에서 등교하는 경우. 최선의 경우 시간은 5 + 30 + 10 = 45분	정류장에 도착하는 순간 버스가 떠나고 길이 최고로 막혀 평소보다 30분이 더 걸린 경우. 최악의 경우 시간은 5 + 20 + 30 + 30 + 10 = 95분	버스를 딱 10분 기다리고, 길도 중간정도 막혀서 15분 정도 더 걸린 경우. 평균 경우의 시간은 5 + 10 + 30 + 15 + 10 = 70분

상각 분석이란?

- **단순히 한 번의 등교시간을 분석하는 것이 아니라, 예를 들어, 한 학기 동안의 등교시간을 분석해본다는 점에서 그 의미를 가짐**
  - 한 학기 동안에 100번 등교 할 때, 항상 버스만 타서 걸리는 최악의 경우 시간 9,500분은 실제 등교 시간보다 지나치게 긴 시간
  - 실제로 어떤 날은 택시나 카풀로 학교에 갈 수 있으며, 자취하는 친구 집에서 잘 수도 있으니까.
  - 상각분석은 한 학기 동안 학교에 가는데 소요된 시간을 모두 합해서 학교에 간 횟수인100으로 나눈 값을 1회 등교 시간으로 분석
- **상각분석은 입력의 확률분포에 대한 가정이 필요 없으며, 최악경우 분석보다 매우 정확한 분석이 가능.**
- **상각분석이 가능한 알고리즘들은 시간이 아주 오래 걸리는 연산과 짧은 시간이 걸리는 연산들이 뒤섞여서 수행되는 공통점.**