자료구조론

2장 소프트웨어와 자료구조

❑ 이 장에서 다룰 내용

1. 소프트웨어 생명주기
2. 추상 데이터 타입
3. 알고리즘
4. 성능 분석

❑ 소프트웨어 생명주기

❖ 성공적인 소프트웨어 개발이란?

- **원하는 결과물을 얻어야 함**
- **그 밖에도 정확하고 효율적으로 소프트웨어의 개발과 사용 및 관리가 이루어져야 함**
- **정확하고 효율적으로 소프트웨어의 개발을 위해서는**
  - 개발할 소프트웨어에 대한 정확한 이해
  - 사용할 자료와 자료간의 연산관계를 분석하여 최적의 자료구조 정의

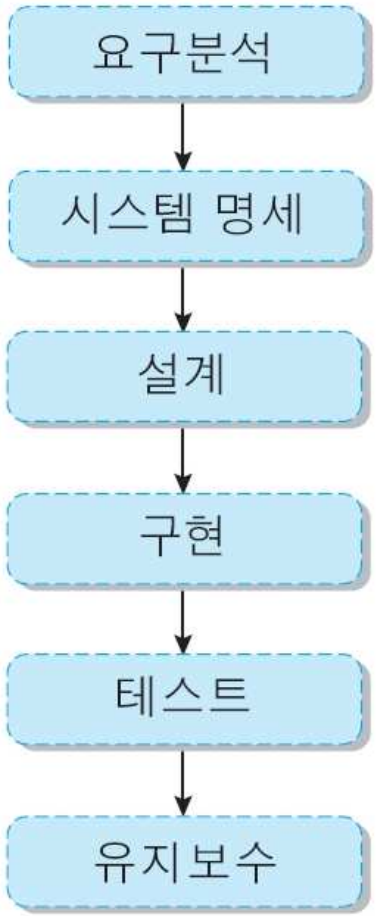
❖ 소프트웨어 생명주기(Software Life Cycle)

- **소프트웨어를 체계적으로 개발하고 관리하기 위해서 개발 과정을 단계별로 나누어 구분한 것**
- **일반적으로 6단계로 구분**

❑ 소프트웨어 생명주기

▪ 일반적인 소프트웨어의 생명주기

- 필요한 단계로 피드백을 반복 수행하면서 소프트웨어의 완성도를 높인다.



❑ 소프트웨어 생명주기

❖ 요구분석 단계

- 문제 분석 단계
- 개발할 소프트웨어의 기능과 제약조건, 목표 등을 소프트웨어 사용자와 함께 명확히 정의하는 단계
- 개발할 소프트웨어의 성격을 정확히 이해하고 개발 방법과 필요한 개발 자원 및 예산 예측
- 요구명세서 작성

❖ 시스템 명세

- 시스템이 무엇을 수행해야 하는가를 정의하는 단계
- 입력 자료, 처리 내용, 생성되는 출력이 무엇인지를 정의
- 시스템 기능 명세서 작성

❑ 소프트웨어 생명주기

❖ 설계 단계

- **시스템 명세 단계에서 정의한 기능을 실제로 수행하기 위한 방법을 논리적으로 결정하는 단계**
- **시스템 구조 설계**
  - 시스템을 구성하는 내부 프로그램이나 모듈 간의 관계와 구조 설계
- **프로그램 설계**
  - 프로그램 내의 각 모듈에서의 처리 절차나 알고리즘을 설계
- **사용자 인터페이스 설계**
  - 사용자가 시스템을 사용하기 위해 보여지는 부분 설계
- **설계방법**
  - 하향식 설계 방법(top-down) : 분할정복(divide and conquer) 방식
  - 상향식 설계 방법(bottom-up)
  - 객체지향 설계 방법(object oriented)

❑ 소프트웨어 생명주기

❖ 구현 단계

- **설계 단계에서 논리적으로 결정한 문제 해결 방법(알고리즘)을 프로그래밍언어를 사용하여 실제 프로그램을 작성하는 단계**
- **프로그래밍 언어 선택시 고려할 사항**
  - 프로젝트에 대한 사용자의 요구
  - 프로그래머의 능력
  - 현재 사용 중인 언어
  - 컴파일러의 가용성과 품질
  - 지원 가능한 개발 도구
  - 언어의 호환성
  - 개발 경험 등
- **표과적인 프로그래밍 스타일**
  - 프로그램을 간결하고 읽기 쉽게 작성

❑ 소프트웨어 생명주기

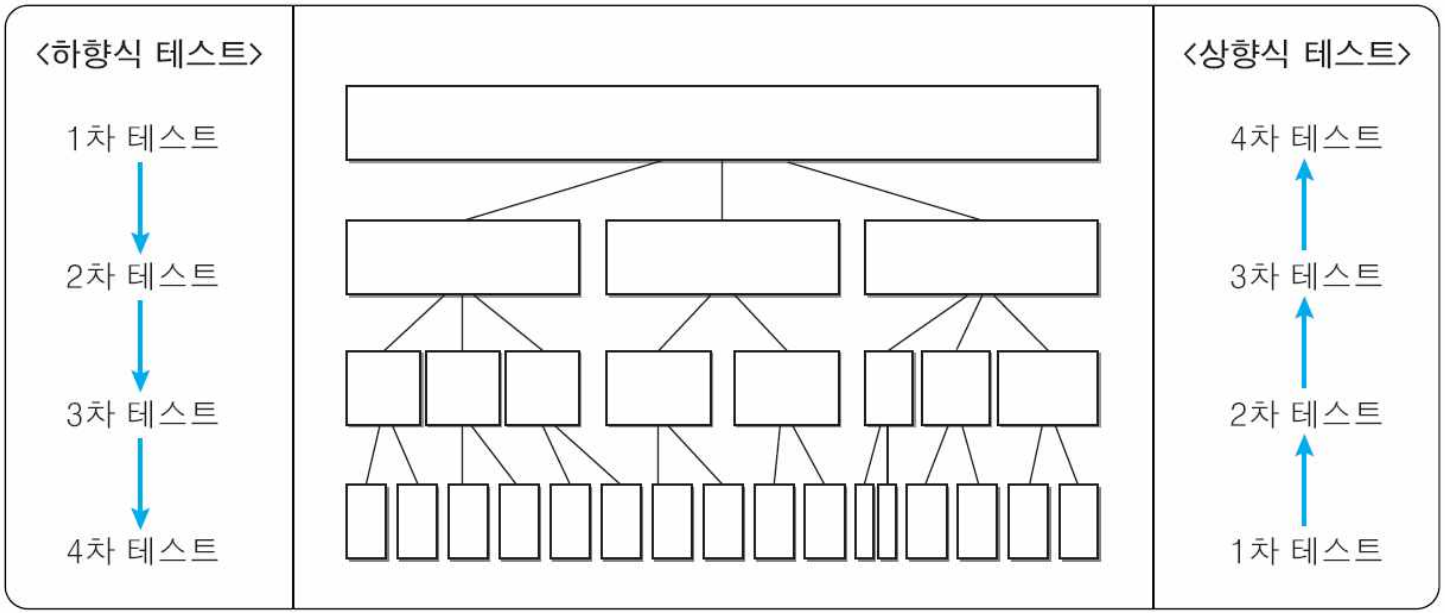
❖ 테스트 단계

- **개발한 시스템이 요구사항을 만족하는지, 실행결과가 예상한 결과와 정확하게 맞는지를 검사하고 평가하는 일련의 과정**
- **숨어있는 오류를 최대한 찾아내어 시스템의 완성도를 높이는 단계**

- **1단계 : 단위 테스트(Unit Test)**
  - 시스템의 최소 구성요소가 되는 모듈에 대해서 개별적으로 시행
- **2단계 : 통합테스트(Integration test)**
  - 단위 테스트를 통과한 모듈을 연결하여 전체 시스템으로 완성하여 통합적으로 시행하는 테스트
  - 구성요소 연결을 점진적으로 확장하면서 테스트 시행
    - ➢ 하향식 테스트
    - ➢ 상향식 테스트

❑ 소프트웨어 생명주기

- 하향식/상향식 점진적 테스트



- **3단계 : 인수 테스트**
  - 완성된 시스템을 인수하기 위해서 실제 자료를 사용한 최종 테스트
  - 알파테스트(내부) / 베타테스트(잠재 고객)

❑ 소프트웨어 생명주기

❖ 유지보수 단계

- **시스템이 인수되고 설치된 후 일어나는 모든 활동**
  - 소프트웨어 생명주기에서 가장 긴 기간
- **유지보수의 유형**
  - 수정형 유지보수
    - ➢ 사용 중에 발견한 프로그램의 오류 수정 작업
  - 적응형 유지보수
    - ➢ 시스템과 관련한 환경적 변화에 적응하기 위한 재조정 작업
  - 완전형 유지보수
    - ➢ 시스템의 성능을 향상시키기 위한 개선 작업
  - 예방형 유지보수
    - ➢ 앞으로 발생할지 모를 변경 사항을 수용하기 위한 대비 작업

❑ 소프트웨어 생명주기

▪ 개발된 소프트웨어의 품질 평가

- 정확성
  - ➢ 요구되는 기능들을 정확하게 수행하는 정도를 평가
- 유지 보수성
  - ➢ 효율적 유지 보수의 정도를 평가
- 무결성
  - ➢ 바이러스 등의 외부 공격에 대한 보안성 평가
- 사용성
  - ➢ 사용자가 쉽고 편리하게 사용할 수 있는가에 대한 평가

❑ 추상 자료형

❖ 컴퓨터를 이용한 문제해결에서의 추상화

- **추상화(abstraction)**
  - 자세하고 복잡한 것 대신, 필수적인 중요한 특징만 골라 단순화 시킴
- **추상화와 구체화**
  - 추상화 – "무엇(what)인가?"를 논리적으로 정의
  - 구체화 – "어떻게(how) 할 것인가?"를 실제적으로 표현
- **컴퓨터에서 문제를 해결할 때도 추상화 작업을 적용하여 복잡한 문제를 단순화 시켜 좀 더 쉽게 해결하는 방법을 찾음**
- **자료 추상화(Data Abstraction) : 처리할 자료, 연산, 자료형에 대한 추상화 표현**
  - 자료(data) : 프로그램의 처리 대상이 되는 모든 것을 의미
  - 연산: 어떤 일을 처리하는 과정. 연산자에 의해 수행
    - 예) 더하기 연산은 +연산자에 의해 수행
  - 자료형(data type) : 처리할 자료의 집합과 자료에 대해 수행할 연산자의 집합
    - 정수 자료형의 자료 : 정수의 집합. $\{\ldots, -1, 0, 1, \ldots\}$
    - 연산자 : 정수에 대한 연산자 집합. $\{+, -, \times, \div, \mathrm{mod}\}$

❑ 추상 자료형

❖ 추상 자료형(ADT, Abstract Data Type)

- 자료와 연산자의 특성을 논리적으로 추상화하여 정의한 자료형
- 스택 추상자료형(7장)

```
ADT Stack
    데이터 : 0개 이상의 원소를 가진 유한 순서 리스트
    연산 :    S ∈ Stack; item ∈ Element;
              createStack() ::= create an empty Stack;
                  // 공백 스택을 생성하는 연산
              isEmpty(S) ::= if (S is empty) then return true
                             else return false;
                  // 스택 S가 공백인지 아닌지를 확인하는 연산
              push(S, item) ::=insert item onto the top of S;
                  // 스택 S의 top에 item(원소)을 삽입하는 연산
              pop(S) ::= if (isEmpty(S)) then return error
                         else { delete and return the top item of S };
                  // 스택 S의 top에 있는 item(원소)을 스택에서 삭제하고 반환하는 연산
              delete(S) ::= if (isEmpty(S)) then return error
                            else delete the top item;a
                  // 스택 S의 top에 있는 item(원소)을 삭제하는 연산
              peek(S) ::= if (isEmpty(S)) then return error
                          else return (the top item of the S);
                  // 스택 S의 top에 있는 item(원소)을 반환하는 연산
End Stack
```

❑ 알고리즘

❖ 알고리즘

- 문제 해결 방법을 추상화하여 각 절차를 논리적으로 기술해 놓은 명세서

▪ 알고리즘의 조건

- 입력(input):
  - ➢ 알고리즘 수행에 필요한 자료가 외부에서 입력으로 제공될 수 있어야 한다.
- 출력(output)
  - ➢ 알고리즘 수행 후 하나 이상의 결과를 출력해야 한다.
- 명확성(definiteness)
  - ➢ 수행할 작업의 내용과 순서를 나타내는 알고리즘의 명령어들은 명확하게 명세되어야 한다.
- 유한성(finiteness)
  - ➢ 알고리즘은 수행 뒤에 반드시 종료되어야 한다.
- 효과성(effectiveness)
  - ➢ 알고리즘의 모든 명령어들은 기본적이며 실행이 가능해야 한다.

❑ 알고리즘

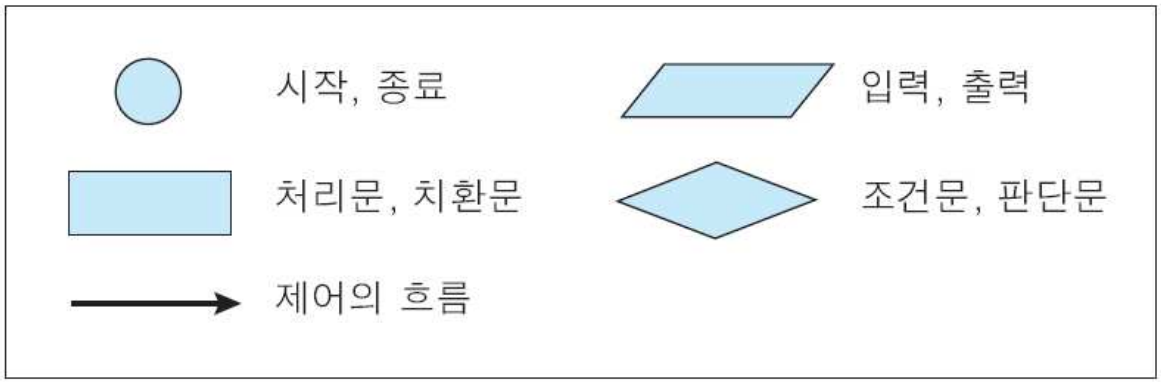
❖ 알고리즘의 표현 방법

- 자연어를 이용한 서술적 표현 방법
- 순서도(Flow chart)를 이용한 도식화 표현 방법
- 프로그래밍 언어를 이용한 구체화 방법
- 가상코드(Pseudo-code)를 이용한 추상화 방법

❑ 알고리즘

❖ 순서도를 이용한 알고리즘의 표현

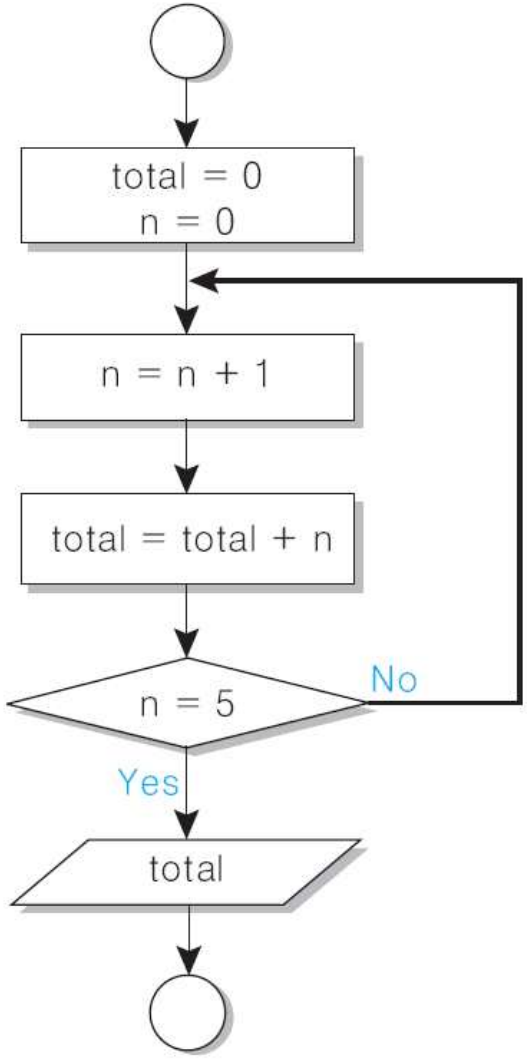
- **순서도에서 사용하는 기호**



- **장점 : 알고리즘의 흐름 파악이 용이함**
- **단점 : 복잡한 알고리즘의 표현이 어려움**

❑ 알고리즘

- 순서도 예) 1부터 5까지의 합을 구하는 알고리즘



❑ 알고리즘

❖ 가상코드를 이용한 알고리즘의 표현

- 가상코드, 즉 알고리즘 기술언어(ADL)를 사용하여 프로그래밍 언어의 일반적인 형태와 유사하게 알고리즘을 표현
- 특정 프로그래밍 언어가 아니므로 직접 실행은 불가능
- 일반적인 프로그래밍 언어의 형태이므로 원하는 특정 프로그래밍 언어로의 변환 용이
- 피보나치 수열 알고리즘

```
   fibonacci(n)
01     if (n<0) then
02                stop ;
03     if (n≤1) then
04                return n ;
05     f1 ← 0 ;
06     f2 ← 1 ;
07     for (i←2 ; i≤n ; i←i+1) do {
08          fn←f1+f2 ;
09          f1←f2 ;
10          f2←fn ;
11     }
12     return  fn ;
13  end
```

❑ 성능분석

❖ 알고리즘 분석

- **하나의 문제에 대해 알고리즘은 여러 개일 수 있으므로 그 중 가장 효율적이고 사용 환경에 최적인 할고리즘을 결정하기 위해서는 알고리즘 분석이 필요하다**
- **분석 기준**
  - 정확성 : 올바른 입력이 들어왔을 대 정해진 시간 내에 올바른 결과를 출력하는가
  - 명확성 : 알고리즘의 표현이 이해하기 쉽게 명확한가
  - 수행량 : 알고리즘의 특성을 나타내는 중요 연산이 얼마나 여러 번 수행되는가
  - 메모리 사용량 : 알고리즘에 의해 사용되는 메모리의 양이 어느정도인가
  - 최적성 : 알고리즘을 적용할 시스템의 사용환경에 적합한가

❑ 성능분석

❖ 알고리즘 성능 분석 방법

▪ 공간 복잡도

- 알고리즘을 프로그램으로 실행하여 완료하기까지 필요한 총 저장 공간량
- 공간 복잡도 = 고정 공간 + 가변 공간

▪ 시간 복잡도

- 알고리즘을 프로그램으로 실행하여 완료하기까지의 총 소요시간
- 시간 복잡도 = 컴파일 시간 + 실행 시간
  - ➢ 컴파일 시간 : 프로그램마다 거의 고정적인 시간 소요
  - ➢ 실행 시간 : 컴퓨터의 성능에 따라 달라질 수 있으므로 실제 실행시간보다 명령문의 실행 빈도수에 따라 계산
- 실행 빈도수의 계산
  - ➢ 지정문, 조건문, 반복문 내의 제어문과 반환문은 실행시간 차이가 거의 없으므로 하나의 단위시간으로 갖는 기본 명령문으로 취급

❑ 성능분석

- 피보나치 수열 알고리즘의 빈도수 구하기

[알고리즘 2-1]

```
   fibonacci(n)
01     if (n<0) then
02                stop ;
03     if (n≤1) then
04                return n ;
05     f1 ← 0 ;
06     f2 ← 1 ;
07     for (i←2 ; i≤n ; i←i+1) do {
08          fn←f1+f2 ;
09          f1←f2 ;
10          f2←fn ;
11     }
12     return fn ;
13 end
```

❑ 성능분석

- n<0, n=0, n=1의 경우에 대한 실행 빈도수

  ➢ for 반복문이 수행되지 않기 때문에 실행 빈도수가 작다.

행 번호	n<0	n=0	n=1
1	1	1	1
2	1	0	0
3	0	1	1
4	0	1	1
5~13	0	0	0

```
   fibonacci(n)
01     if (n<0) then
02              stop ;
03     if (n≤1) then
04              return n ;
05     f1 ← 0 ;
06     f2 ← 1 ;
07     for (i←2 ; i≤n ; i←i+1) do {
08          fn←f1+f2 ;
09          f1←f2 ;
10          f2←fn ;
11     }
12     return  fn ;
13  end
```

❑ 성능분석

- n>1의 일반적인 경우에 대한 실행 빈도수
  - ➢ n에 따라 for 반복문 수행

행 번호	실행 빈도수	행 번호	실행 빈도수
1	1	8	n-1
2	0	9	n-1
3	1	10	n-1
4	0	11	0
5	1	12	1
6	1	13	0
7	n		

  - ➢ 총 실행 빈도수

$$= 1+0+1+0+1+1+n+(n-1)+(n-1)+(n-1)+0+1+0 = \mathbf{4n+2}$$

❑ 성능분석

▪ 시간 복잡도 표기법

- 빅-오(Big-Oh) 표기법 사용
- 빅-오(Big-Oh) 표기법 순서

  ① 실행 빈도수를 구하여 **실행시간 함수** 찾기
  ② 실행시간 함수의 값에 가장 큰 영향을 주는 **n에 대한 항을 선택**하여
  ③ **계수는 생략**하고 O (Big-Oh)의 오른쪽 괄호 안에 표시

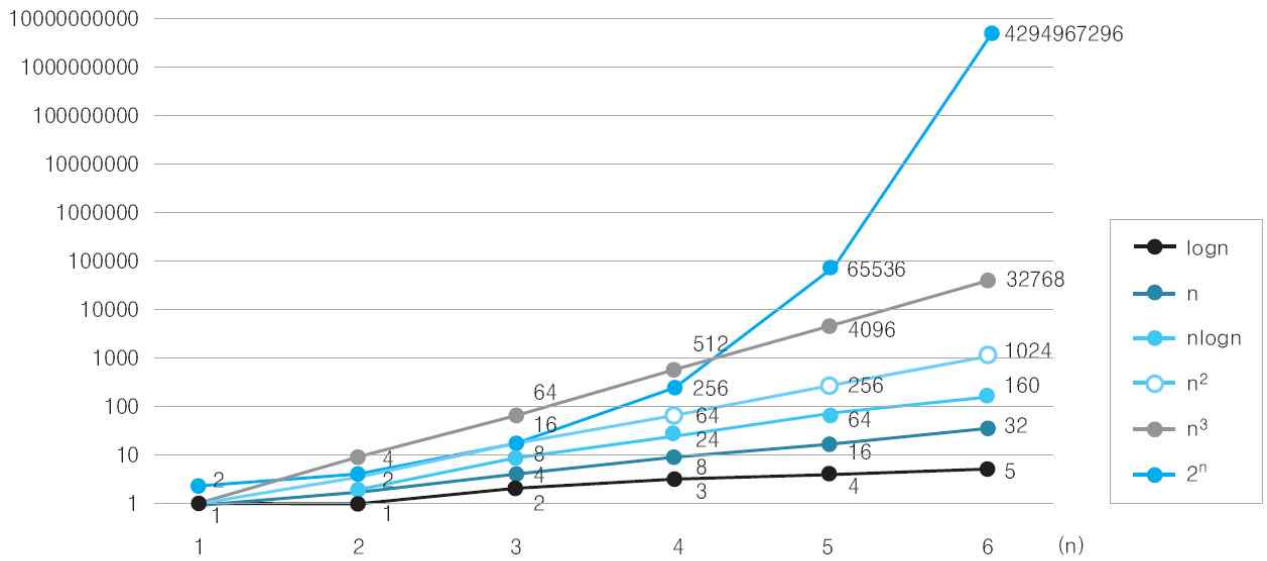
- 피보나치 수열의 시간 복잡도 = O (n)

  ① 실행시간 함수 : $4n+2$
  ② n에 대한 항을 선택 : $4n$
  ③ 계수 4는 생략하고 O (Big-Oh)의 오른쪽 괄호 안에 표시 : O (n)

❑ 성능분석

- 각 실행 시간 함수에서 n값의 변화에 따른 실행 빈도수 비교

logn	<	n	<	nlogn	<	n^2	<	n^3	<	2^n
0		1		0		1		1		2
1		2		2		4		8		4
2		4		8		16		64		16
3		8		24		64		512		256
4		16		64		256		4096		65536
5		32		160		1024		32768		4294967296



[그림 2-17] n값에 대한 실행 시간 함수 그래프