자료구조 Data Structure | 조행래

자료구조의 기본 개념

자료구조와 알고리즘의 정의 및 표현

학습 목표

- 본 강좌의 필요성을 이해할 수 있다.
- 자료구조와 알고리즘의 정의를 이해할 수 있다.
- 자료구조와 알고리즘을 표현할 수 있다.

1. 강의 소개

- **자료구조와 알고리즘은 컴퓨터 프로그램을 구성하는 가장 중요한 두 가지 요소**

- **강의의 주요 내용**
  - 보편적으로 사용되고 있는 자료 구조들(Array, Stack, Queue, List, Tree, Graph) 소개
  - 자료구조를 이용한 알고리즘들(Sorting, Searching)의 기본 개념 소개
  - 알고리즘의 복잡성을 공간 복잡성 및 시간 복잡성의 관점에서 분석할 수 있는 방법을 설명

2. 자료구조란?

- **자료구조의 정의**
  - 문제 해결을 위해 데이터를 조직하여 표현하는 방법
- **자료구조의 예**
  - 전화번호부: (이름, 전화번호)의 나열 → **배열, 연결 리스트, 트리**
    - 제한 사항: 이름을 빨리 검색, 또는 새로운 (이름, 전화번호) 쌍을 쉽게 추가
  - 수강신청, 연말정산의 대기 줄 → **큐**
  - 지하철 노선도: 역 이름, 역 간의 선후 관계 → **그래프**
- **자료구조의 중요성**
  - 주어진 문제의 특성에 맞는 자료구조를 선택 → 프로그램의 개발이 쉽고, 성능이 향상

추상 데이터 타입

- **추상 데이터 타입(Abstract Data Type)**
  - 자료구조를 기술할 때 사용하는 방법
  - 데이터 객체 및 연산의 명세와 데이터 객체의 내부 표현양식/연산의 구현 내용을 분리
    - 예: Ada package, C++ class
- **ADT에서 연산의 명세**
  - 구성 요소: 함수 이름, 인자들의 타입, 결과들의 타입
  - 함수의 호출 방법 및 결과물이 무엇인지를 설명
  - 함수의 내부 동작과정 및 구현 방법은 은폐
  - **Information Hiding**

추상 데이터 타입의 예 - 자연수

ADT NatNum

객체: 0부터 시작하여 컴퓨터로 표현할 수 있는 최대 정수(INT_MAX)까지의 범위에 속하는 정수들의 집합

연산:

for all x, y ∈ NatNum; TRUE, FALSE ∈ Boolean
and where +, –, <, and == are the usual integer operations

```
NatNum Zero( )          ::=  0
Boolean Is_Zero(x)      ::=  if (x) return FALSE
                             else return TRUE
NatNum Add(x, y)        ::=  if ((x + y) <= INT_MAX) return x + y
                             else return INT_MAX
Boolean Equal(x, y)     ::=  if ( x == y ) return TRUE
                             else return FALSE
NatNum Successor(x)     ::=  if ( x == INT_MAX ) return x
                             else return x + 1
NatNum Subtract(x, y)   ::=  if (x < y) return 0, else return x – y
```

end NatNum

3. 알고리즘 이란?

- **알고리즘의 정의**
  - 문제 해결을 위해 특정한 일을 수행하는 명령어들의 집합

- **알고리즘이 만족해야 할 조건**
  - 입력(Input): 0 혹은 그 이상의 입력이 존재
  - 출력(Output): 적어도 하나 이상의 결과물이 출력
  - 명확성(Definiteness): 알고리즘을 구성하는 명령어들의 의미는 명확하여야 하며, 애매모호해서는 안 된다.
  - 유한성(Finiteness): 알고리즘은 한정된 수의 명령어들을 실행한 후 종료하여야 한다.
  - 유효성/실행가능성(Effectiveness): 모든 명령어들은 실행 가능하여야 한다.

알고리즘의 예

- **코끼리를 냉장고에 넣는 방법**

  (입력: 냉장고와 코끼리, 출력: 코끼리가 들어간 냉장고)

  1. 냉장고 문을 연다.
  2. 코끼리를 냉장고에 넣는다.
  3. 냉장고 문을 닫는다.

- **라면을 끓이는 법**

  (입력: 라면 재료, 출력: 맛있게 끓인 라면)

  1. 냄비에 물을 500ml 넣고 거품이 날 때까지 끓인다.
  2. 라면과 수프를 함께 넣는다.
  3. 거품이 나면 불을 끈다.

4. 알고리즘의 표현

- **알고리즘을 표현하는 다양한 방법이 존재**
- **예: 이진 검색**
  - 오름차순으로 정렬된 정수배열 list[]에서 key가 주어질 때,
    list[i] = key인 i를 발견하는 문제
- **알고리즘의 표현 방법 1: 자연어**
  - left = 0, right = n-1. middle = (left+right)/2로 두자.
  - list[middle]과 key를 비교하여, 아래 경우들을 처리
    - list[middle] < key: key가 list[middle+1]부터 list[right]에 있으므로, 다시 조사
    - list[middle] = key: middle을 반환
    - list[middle] > key: key가 list[left]부터 list[middle-1]에 있으므로, 다시 조사

4. 알고리즘의 표현(계속)

- **알고리즘의 표현 방법 2: 유사 코드(pseudo code)**

```
while ( there are more integers to check ) {
     middle = ( left + right ) / 2;
     if ( key < list [ middle ] )
            right = middle - 1;
     else if ( key == list [middle ] )
          return middle;
     else
          left = middle + 1;
}
```

4. 알고리즘의 표현(계속)

- **알고리즘의 표현 방법 3: 프로그래밍 언어**

```
int binsearch(int list[], int key, int left, int right)
{   // search list[0] <= list[1] <= ... <= list[n-1] for key.

    int middle;                         // left = 0, right = n-1 로 전달
    while (left <= right) {
        middle = (left + right)/2;
        switch (compare(list[middle], key)) {
            case -1: left = middle + 1;       // key가 크다
                break;
            case 0: return middle;            // 같다
            case 1: right = middle – 1;       // key가 작다
        }
     }
    return -1;
}
```

```
#define compare(x, y) (((x)<(y)) ? -1 : ((x)==(y)) ? 0 : 1)
```

5. 순환 알고리즘

- **순환 알고리즘 (Recursive Algorithm)의 정의**
  - 자기 자신을 다시 호출하는 알고리즘
  - 예: 팩토리얼 계산

```
int factorial(int n) {
    if (n <= 1)   return 1;
    else          return n * factorial(n-1);
}
```

- **재귀 알고리즘을 작성하는 방법**
  - 재귀 호출을 종료하는 경계 조건을 설정
  - 각 단계마다 경계 조건에 접근하도록 알고리즘의 재귀 호출

순환 알고리즘의 예 – 이진 검색

```
int binsearch(int list[], int key, int left, int right)
{
    int middle;
    if (left <= right) {      // 경계 조건
        middle = (left + right) / 2;
        switch (compare(list[middle], key)) {
         case -1:                 // 경계 조건에 접근
             return binsearch(list, key, middle+1, right);
         case 0: return  middle;
         case 1:                  // 경계 조건에 접근
             return binsearch(list, key, left, middle-1);
        }
    }
    return -1;
}
```

요약 정리

- 자료구조의 정의를 이해
- 알고리즘의 정의와 제한 조건을 이해
- 알고리즘의 표현방식을 이해
- 순환 알고리즘을 이해