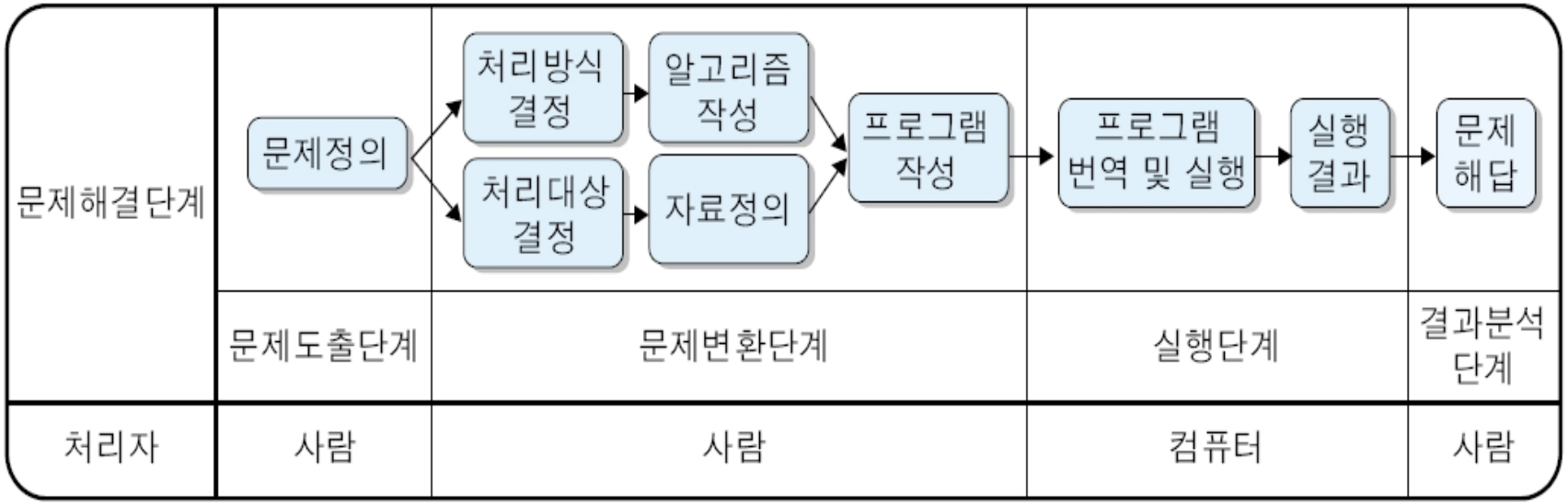
Data Structure

✓ 컴퓨터를 이용하여 프로그램을 하는데 있어서 가장 기본이 되는 자료에 대한 다양한 형태의 구조에 대하여 학습하며, 각 구조의 특징과 주어진 상황에 따른 장단점을 파악하여, 실제적인 프로그램 작업에서 적용할 수 있는 능력을 배양한다.

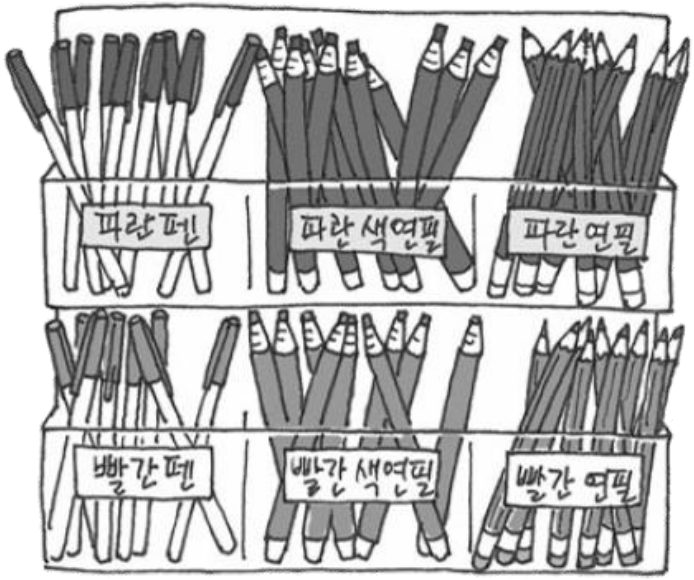
▪ 자료구조의 필요성

- 컴퓨터가 효율적으로 문제를 처리하기 위해서는 문제를 정의하고 분석하여 그에 대한 최적의 프로그램을 작성해야 하기 때문에 자료구조에 대한 개념과 활용 능력을 필요로 한다.



- **자료구조**
  - 프로그램에서 자료들을 정리하는 구조
  - 자료구조란 자료를 효율적으로 사용하기 위해서 자료의 특성에 따라서 분류하여 구성하고 저장 및 처리하는 모든 작업

[나쁜 자료구조]



[좋은 자료구조]

- **자료의 형태에 따른 분류**


자료구조
단순구조
정수
실수
문자
문자열
선형구조
리스트
연결 리스트
단순 연결리스트
이중 연결리스트
원형 연결리스트
스택
큐
덱
비선형구조
트리
일반트리
이진트리
그래프
방향그래프
무방향그래프
파일구조
순차파일
색인파일
직접파일

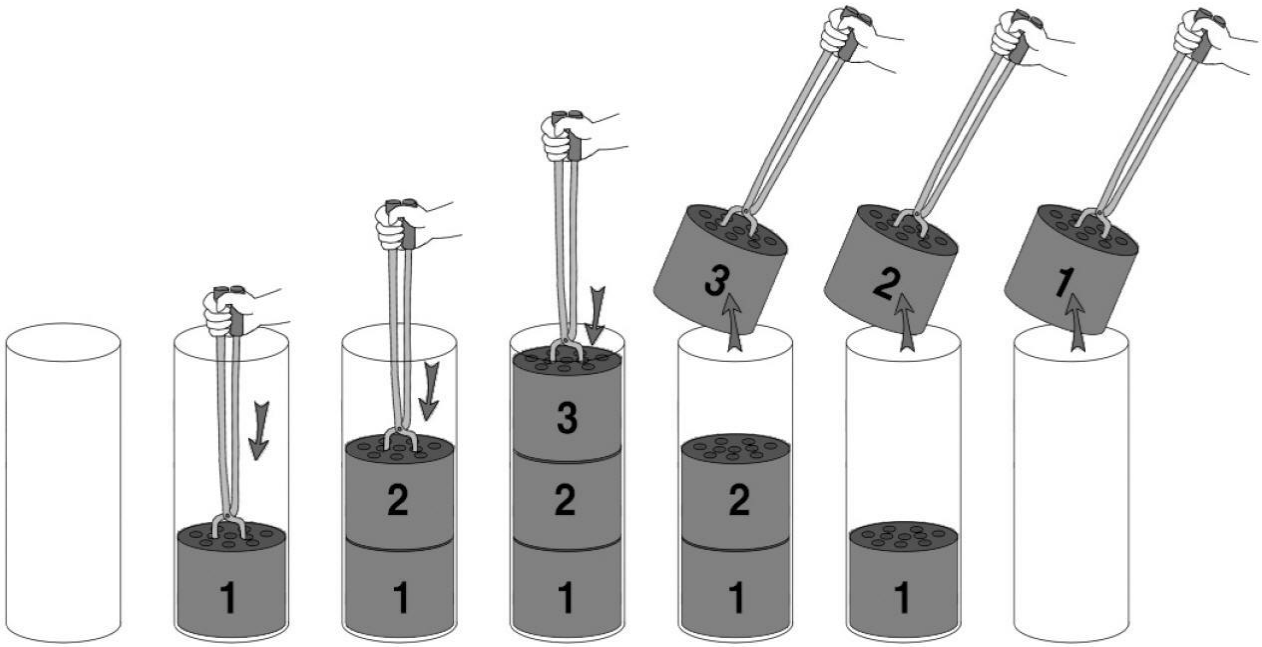
- 기본이 되는 자료구조

  리스트, 스택, 큐, 트리, 정렬, 그래프

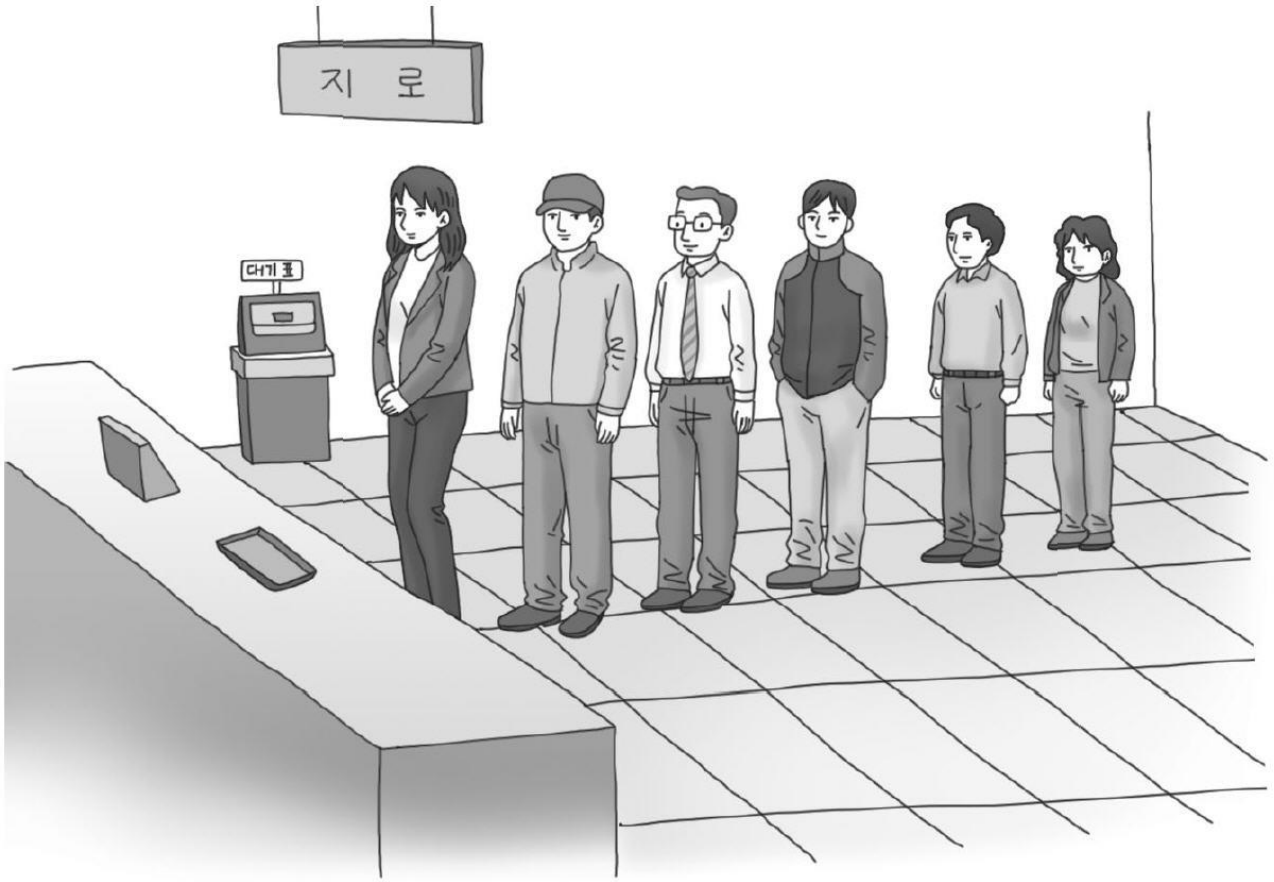
- 리스트 : 순서가 정해진 목록

순위	검색 엔진 사이트
1	네이버
2	야후코리아
3	엠파스
4	한미르
5	구글
6	야후
7	심마니
8	구글

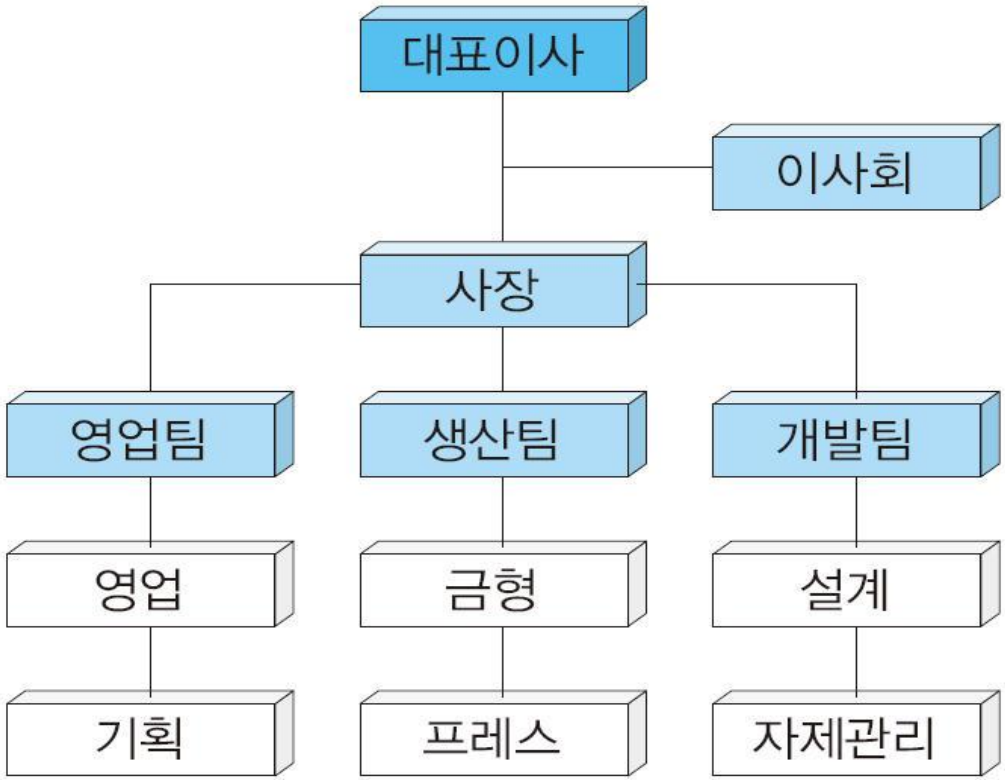
- 스택 : LIFO (last in, first out)



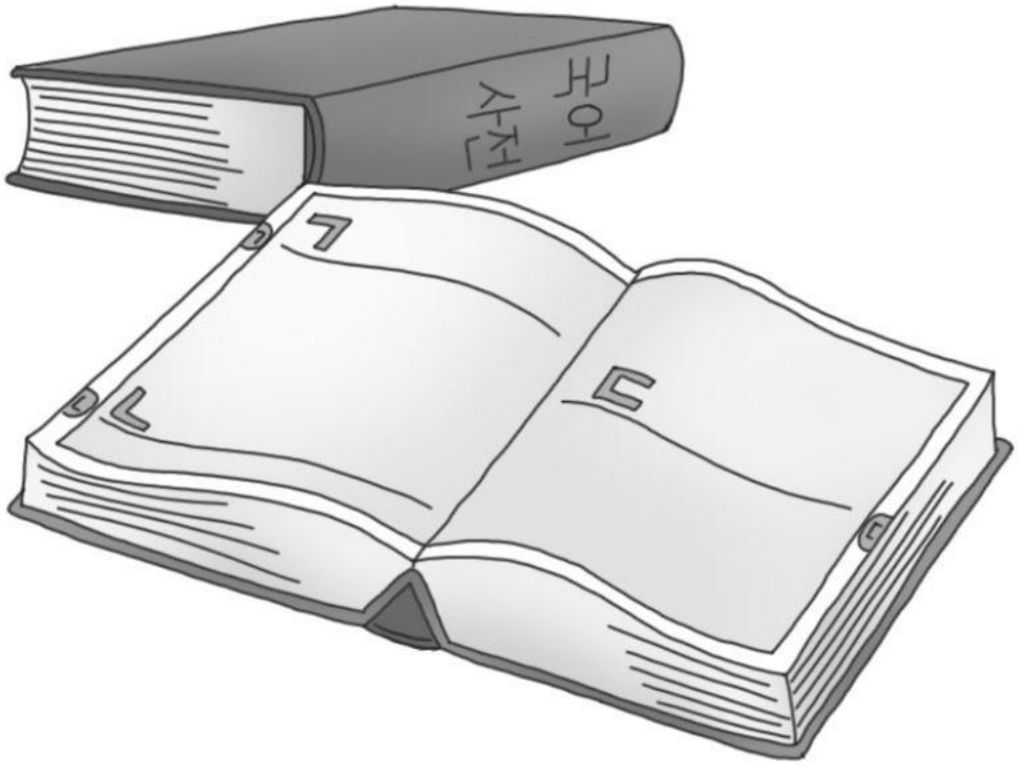
- 큐 : FIFO (first in, first out)



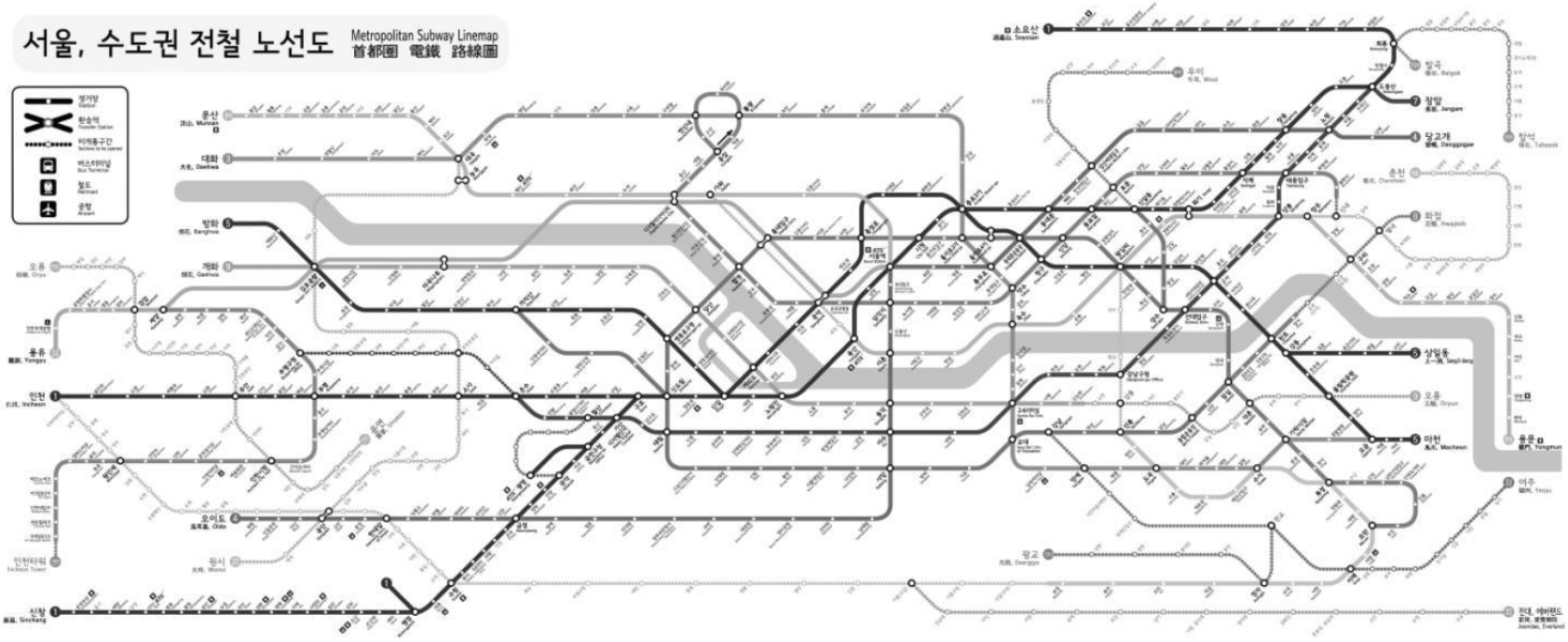
- 트리(tree) : 노드(node)와 가지(edge)로 구성
  - 뿌리 (root) 노드 를 가짐



- 정렬(sorting) : 값의 크기에 따라 나열해 놓은 것



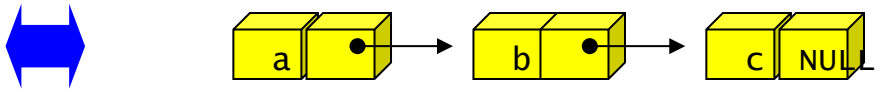
- 그래프 : 노드와 노드 사이 연결관계가 설정(path)

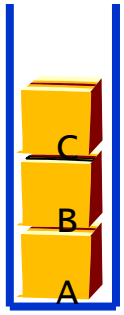


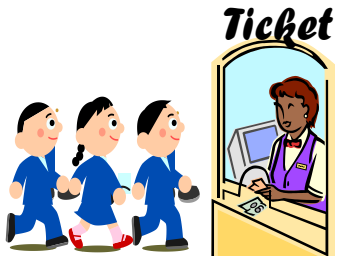
자료구조와 알고리즘

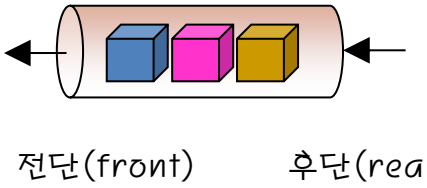
일상생활과 자료구조의 비교

일상생활에서의 예	자료구조
물건을 쌓아두는 것	스택
영화관 매표소의 줄	큐
할일 리스트	리스트
영어사전	사전, 탐색구조
지도	그래프
조직도	트리









자료구조와 알고리즘

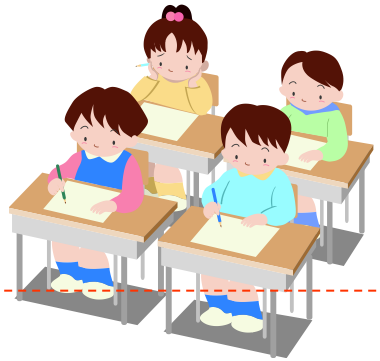
- 프로그램 = 자료구조 + 알고리즘

(예) 최대값 탐색 프로그램 = 배열+ 순차탐색

자료구조

score[]

80	70	90	...	30



알고리즘

```
tmp←score[0];
for i ← 1 to n do
        if  score[i]>tmp
                then tmp←score[i];
```

알고리즘

- 알고리즘(Algorithm)
  - 컴퓨터로 문제를 풀기 위한 단계적인 절차
  - 특정 작업을 수행하기 위한 명령어들의 집합

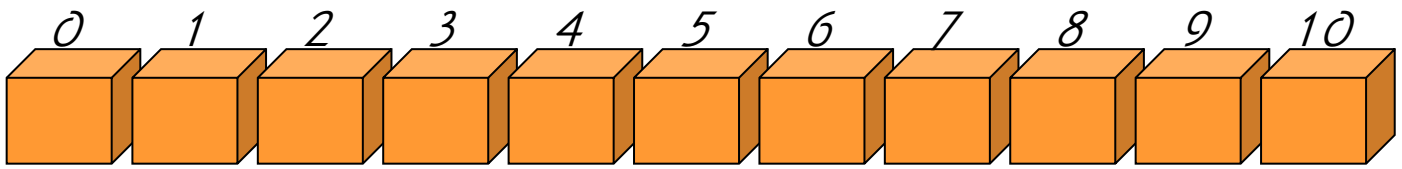
- 알고리즘이 되기 위한 5 가지 조건

조건		설명
1	입력(input)	외부에서 제공되는 데이터가 0개 이상 있다.
2	출력(output)	적어도 한 가지 이상의 결과를 생성한다.
3	명확성(definiteness)	알고리즘을 구성하는 각 명령어들은 그 의미가 명백하고 모호하지 않아야 한다.
4	유한성(finiteness)	알고리즘의 명령대로 순차적인 실행을 하면 언젠가는 반드시 실행이 종료되어야 한다.
5	유효성(effectiveness)	원칙적으로 모든 명령들은 오류가 없이 실행 가능해야 한다.

알고리즘의 기술 방법

① 영어나 한국어와 같은 자연어
② 흐름도(flow chart)
③ 유사 코드(pseudo-code)
④ C와 같은 프로그래밍 언어

(예) 배열에서 최대값 찾기 알고리즘



자연어로 표기된 알고리즘

- 인간이 읽기가 쉽다.
- 그러나 자연어의 단어들을 정확하게 정의하지 않으면 의미 전달이 모호해질 우려가 있다.

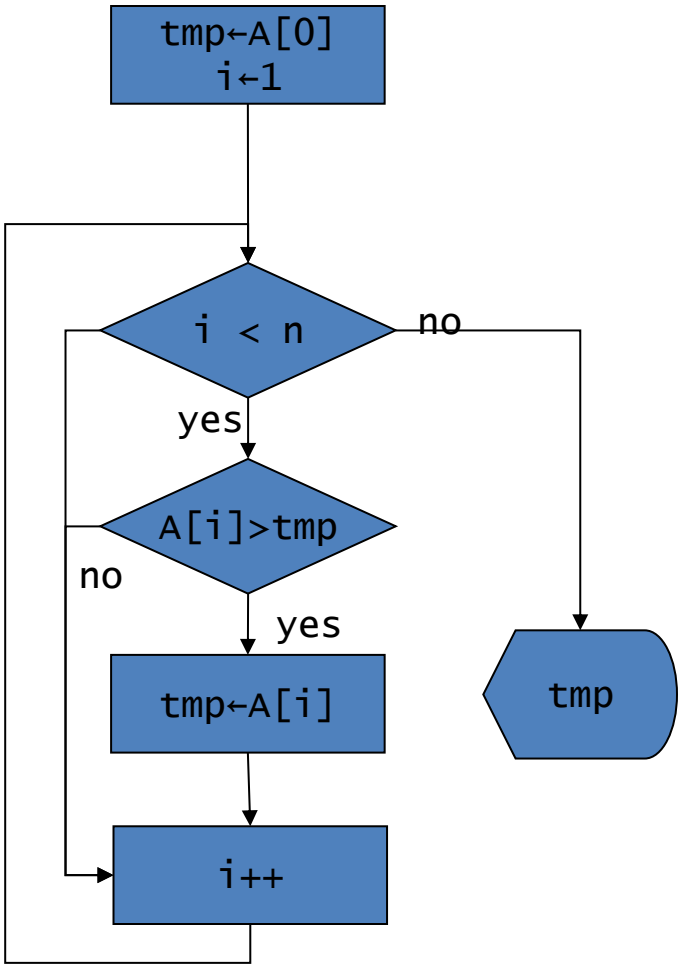
(예) 배열에서 최대값 찾기 알고리즘

ArrayMax(A,n)

1. 배열 A의 첫번째 요소를 변수 tmp에 복사
2. 배열 A의 다음 요소들을 차례대로 tmp와 비교하면 더 크면 tmp로 복사
3. 배열 A의 모든 요소를 비교했으면 tmp를 반환

흐름도로 표기된 알고리즘

- 직관적이고 이해하기 쉬운 알고리즘 기술 방법
- 그러나 복잡한 알고리즘의 경우, 상당히 복잡해짐.



유사코드로 표현된 알고리즘

- 알고리즘의 고수준 기술 방법으로 알고리즘 기술에 가장 많이 사용
- 자연어보다는 더 구조적인 표현 방법이며, 프로그래밍 언어보다는 덜 구체적인 표현 방법임
- 프로그램을 구현할 때의 여러 가지 문제들을 감출 수 있다. 즉 알고리즘의 핵심적인 내용에만 집중할 수 있다.

```
ArrayMax(A,n)

  tmp ← A[0];
  for i←1 to n-1 do
          if tmp < A[i] then
                  tmp ← A[i];
  return tmp;
```

대입 연산자가 ←임을 유의

C로 표현된 알고리즘

- 알고리즘의 가장 정확한 기술이 가능
- 반면 실제 구현시의 많은 구체적인 사항들이 알고리즘의 핵심적인 내용들의 이해를 방해할 수 있다.

```
#define MAX_ELEMENTS 100
int score[MAX_ELEMENTS];      //자료구조
int find_max_score(int n)
{
        int i, tmp;
        tmp=score[0];
        for(i=1;i<n;i++){              //알고리즘
                if( score[i] > tmp ){
                        tmp = score[i];
                }
        }
        return tmp;
}
```

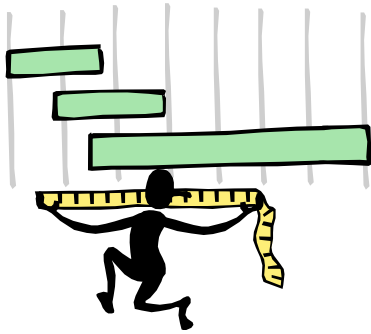
알고리즘의 성능분석

- 성능 평가
  프로그램의 성능 평가(performance evaluation)를 하기 위한 알고리즘의 효율성

- 공간적 효율성과 시간적인 효율성 고려
  - 공간 효율성: 프로그램의 공간 복잡도(space complexity)
    - 프로그램을 실행시켜 완료하는 데 필요한 공간의 양
    - 공간 복잡도 = 고정 공간 + 가변 공간
  - 시간 효율성: 프로그램의 시간 복잡도(time complexity)
    - 프로그램을 실행시켜 완료하는데 필요한 컴퓨터 시간의 양
    - 시간 복잡도 = 컴파일 시간 + 실행 시간

알고리즘의 성능분석

- 알고리즘의 성능 분석 기법
  - 수행 시간 측정
    - 두개의 알고리즘의 실제 수행 시간을 측정하는 것
    - 실제로 구현하는 것이 필요
    - 동일한 하드웨어를 사용하여야 함



  - 알고리즘의 복잡도 분석
    - 시간 복잡도 분석: 수행 시간 분석
      - 직접 구현하지 않고서도 수행 시간을 분석하는 것
      - 알고리즘이 수행하는 연산의 횟수를 측정하여 비교
      - 일반적으로 연산의 횟수는 n의 함수
    - 공간 복잡도 분석:
      - 수행시 필요로 하는 메모리 공간 분석

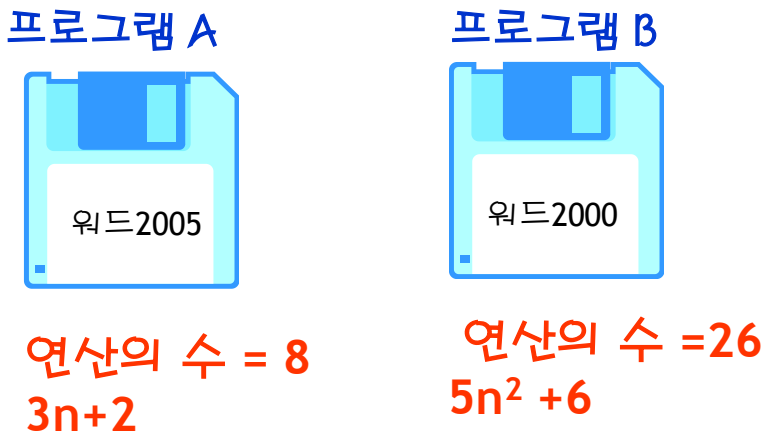
수행시간측정

- 컴퓨터에서 수행시간을 측정하는 방법에는 주로 clock 함수가 사용된다.
- clock_t clock(void);
  - clock 함수는 호출되었을 때의 시스템 시각을 CLOCKS_PER_SEC 단위로 반환
- 수행시간을 측정하는 전형적인 프로그램

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void main( void )
{
   clock_t start, finish;
   double  duration;
   start = clock();
    // 수행시간을 측정하고 하는 코드....
     // ....
   finish = clock();
   duration = (double)(finish - start) / CLOCKS_PER_SEC;
   printf("%f 초입니다.₩n", duration);
}
```

복잡도 분석

- 시간 복잡도는 알고리즘을 이루고 있는 연산들이 몇 번이나 수행되는지를 숫자로 표시
- 알고리즘이 수행하는 연산의 개수를 계산하여 두 개의 알고리즘을 비교할 수 있다.
  - 산술 연산, 대입 연산, 비교 연산, 이동 연산의 기본적인 연산:
  - 연산의 실행횟수
- 연산의 수행횟수는 고정된 숫자가 아니라 입력의 개수 n에 대한 함수 ->시간복잡도 함수라고 하고 T(n) 이라고 표기한다.



복잡도 분석의 예

- n을 n번 더하는 문제:

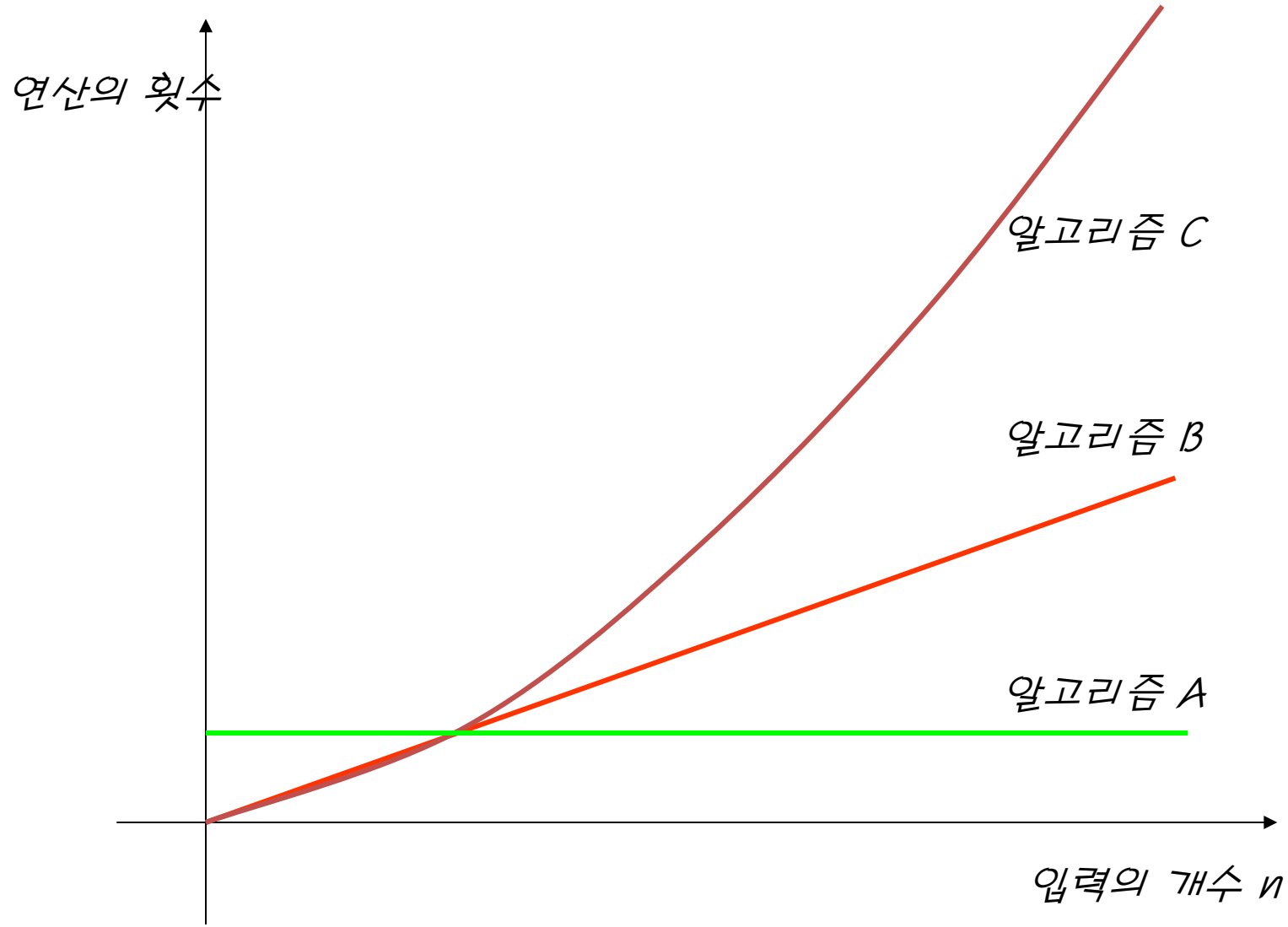
  각 알고리즘이 수행하는 연산의 개수를 세어 본다.
  단 for 루프 제어 연산은 고려하지 않음.

알고리즘 A	알고리즘 B	알고리즘 C

sum ←n*n;	sum ← 0;
	for i ← 1 to n do
	  sum ←sum + n;
		sum ← 0;
		for i←1 to n do
		  for ←1 to n do
		  sum ←sum + 1;

	알고리즘 A	알고리즘 B	알고리즘 C
대입연산	1	n + 1	n*n + 1
덧셈연산		n	n*n
곱셈연산	1		
나눗셈연산			
전체연산수	2	2n + 1	$2n^2 + 1$

연산의 횟수를 그래프로 표현



시간복잡도 함수 계산

- 코드를 분석해보면 수행되는 연산들의 횟수를 입력 크기의 함수로 만들 수 있다.

```
ArrayMax(A,n)

  tmp ← A[0];
  for i←1 to n-1 do
            if tmp < A[i] then
                        tmp ← A[i];
  return tmp;
```

1번의 대입 연산
루프 제어 연산은 제외
n-1번의 비교 연산
n-1번의 대입 연산(최대)
1번의 반환 연산

총 연산수= 2n(최대)

빅오 표기법

- 자료의 개수가 많은 경우에는 차수가 가장 큰 항이 가장 영향을 크게 미치고 다른 항들은 상대적으로 무시될 수 있다.
  - (예) *n*=1,000 일 때, *T(n)*의 값은 1,001,001이고 이중에서 첫 번째 항인 n^2 의 값이 전체의 약 99%인 1,000,000이고 두 번째 항의 값이 1000으로 전체의 약 1%를 차지한다.
- 따라서 보통 시간복잡도 함수에서 가장 영향을 크게 미치는 항만을 고려하면 충분하다.

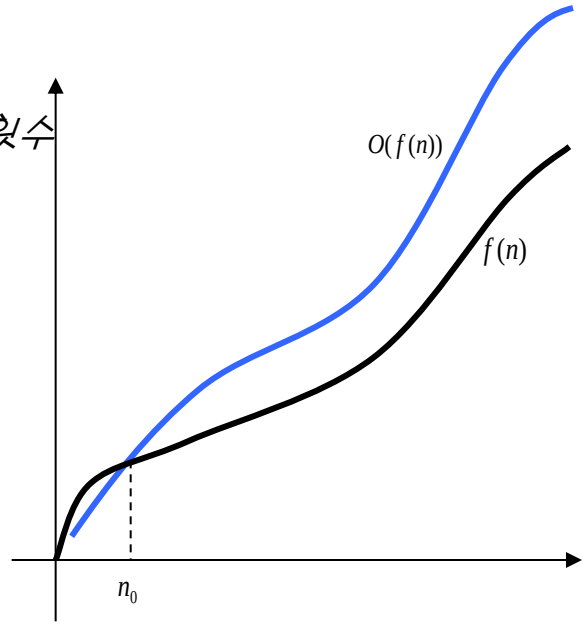
n=1000인 경우

$T(n)= n^2 + n + 1$

99%

1%

- 빅오표기법: 연산의 횟수를 대략적(점근적)으로 표기한 것
- 두개의 함수 f(n)과 g(n)이 주어졌을 때,
  모든 $n \geq n_0$에 대하여 $|f(n)| \leq c|g(n)|$을 만족하는 2개의 상수 c와 n_0가 존재하면 f(n)=O(g(n))이다.
- 빅오는 함수의 상한을 표시한다.
  - (예) n≥5 이면 2n+1 <10n 이므로 2n+1 = O(n)



빅오 표기법

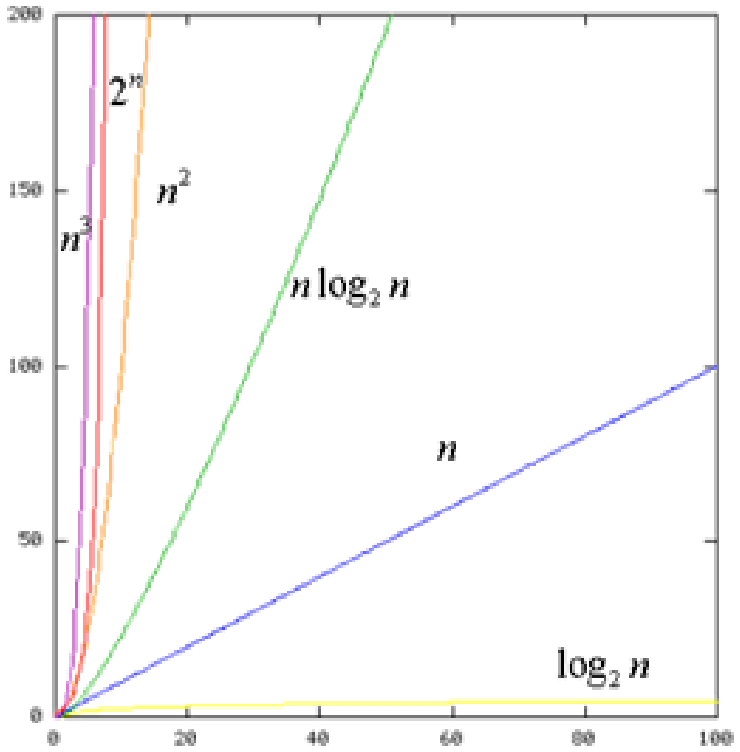
- 빅-오(Big-Oh) 표기법 순서
  - 실행 빈도수를 구하여 실행시간 함수 찾기
  - 실행시간 함수의 값에 가장 큰 영향을 주는 n에 대한 항을 선택하여
  - 계수는 생략하고 O (Big-Oh)의 오른쪽 괄호 안에 표시

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

성능 분석

- 다음의 빅오 표기법은 가장 많이 사용되는 것으로서 실행시간이 빠른 순서대로 기술한 것이다.

빅오 표기법	명칭	실행시간
O(1)	상수	위로 갈수록 실행시간이 빠르고 효율적이다.
O(log n)	로그형	
O(n)	선형	
O(nlog n)	선형로그형	
$O(n^2)$	2차형	
$O(n^3)$	3차형	
$O(2^n)$	지수형	
O(n!)	팩토리얼형	



빅오 표기법의 종류

시간복잡도	*n*					
	1	**2**	**4**	**8**	**16**	**32**
1	1	1	1	1	1	1
logn	0	1	2	3	4	5
n	1	2	4	8	16	32
nlogn	0	2	8	24	64	160
n^2	1	4	16	64	256	1024
n^3	1	8	64	512	4096	32768
2^n	2	4	16	256	65536	4294967296
n!	1	2	24	40326	20922789888000	26313×10^{33}

빅오 표기법이외의 표기법

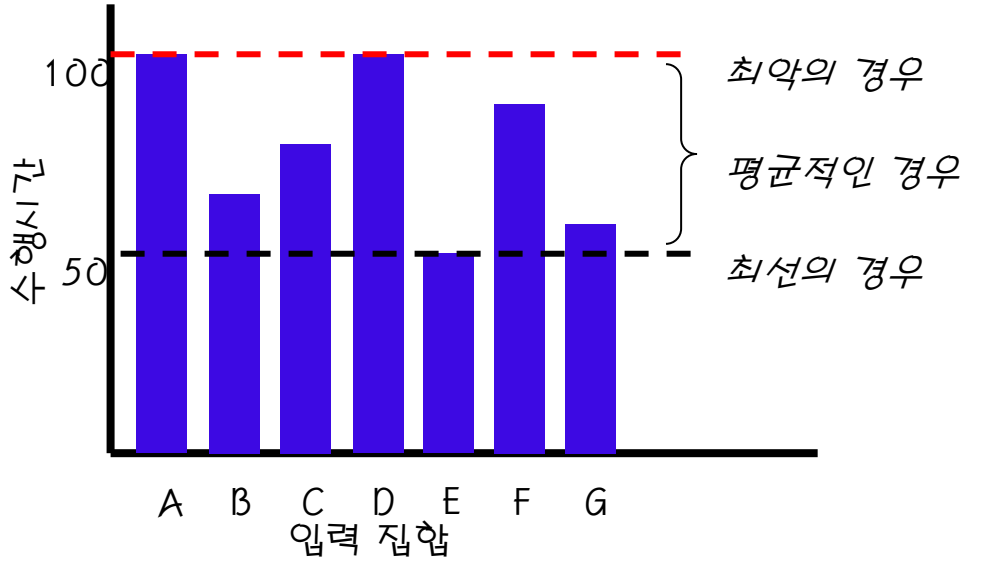
- 빅오메가 표기법(Big Omega, Ω)
  - 빅오메가(Big Omega, Ω) 표기법은 빅오 기호의 반대 개념이다. 알고리즘 수행 시간의 하한(Lower Bound)으로서 “최소한 이만한 시간은 걸린다”라는 의미이다.

최선, 평균, 최악의 경우

- 알고리즘의 수행시간은 입력 자료 집합에 따라 다를 수 있다.

(예) 정렬 알고리즘의 수행 시간은 입력 집합에 따라 다를 수 있다.

- 최선의 경우(best case): 수행 시간이 가장 빠른 경우
- 평균의 경우(average case): 수행시간이 평균적인 경우
- 최악의 경우(worst case): 수행 시간이 가장 늦은 경우



- 최선의 경우: 의미가 없는 경우가 많다.
- 평균적인 경우: 계산하기가 상당히 어려움.
- 최악의 경우: 가장 널리 사용된다. 계산하기 쉽고 응용에 따라서 중요한 의미를 가질 수도 있다.

최선, 평균, 최악의 경우

- (예) 순차탐색

- 최선의 경우: 찾고자 하는 숫자가 맨앞에 있는 경우

  ∴ O(1)

- 최악의 경우: 찾고자 하는 숫자가 맨뒤에 있는 경우

  ∴ O(n)

- 평균적인 경우: 각 요소들이 균일하게 탐색된다고 가정하면

  (1+2+…+n)/n=(n+1)/2

  ∴ O(n)

자료 구조의 C언어 표현방법

- 자료구조와 관련된 데이터들을 구조체로 정의
- 연산을 호출할 경우, 이 구조체를 함수의 파라미터로 전달

(예)

```
// 자료구조 스택과 관련된 자료들을 정의
typedef int element;
typedef struct {
          int top;
          element stack[MAX_STACK_SIZE];
} StackType;


// 자료구조 스택과 관련된 연산들을 정의
void push(StackType *s, element item)
{
          if( s->top >= (MAX_STACK_SIZE -1)){
                    stack_full();
                    return;
          }
          s->stack[++(s->top)] = item;
}
```

자료구조의 요소

관련된 데이터를 구조체로 정의

연산을 호출할때 구조체를 함수의 파라미터로 전달

자료구조 기술규칙

- 상수
  - 대문자로 표기
  - (예) #define MAX_ELEMENT 100

- 변수의 이름
  - 소문자를 사용하였으며 언더라인을 사용하여 단어와 단어를 분리
  - (예) int increment; int new_node;

- 함수의 이름
  - 동사를 이용하여 함수가 하는 작업을 표기
  - (예) int add(ListNode *node) // 혼동이 없는 경우
  - int list_add(ListNode *node) //혼동이 생길 우려가 있는 경우

- typedef의 사용
  - C언어에서 사용자 정의 데이터 타입을 만드는 경우에 쓰이는 키워드

(예) typedef int element;

```
typedef struct ListNode {
    element data;
    struct ListNode *link;
} ListNode;
```

typedef <새로운 타입의 정의> <새로운 타입 이름>;