CSE2010 자료구조론

Week 2: Algorithm, Simple Review of C

ICT융합학부 조용우

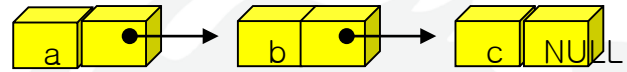
목차

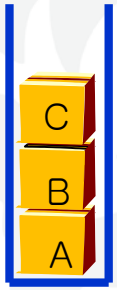
- 자료구조와 알고리즘
- C 언어 리뷰

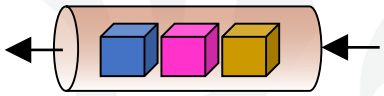
자료구조

일상생활에서 자료구조

일상 생활에서의 예	자료구조
물건을 쌓아 두는 것	스택
영화관 매표소의 줄	큐
할일 리스트	리스트
영어사전	사전, 탐색구조
지도	그래프
조직도	트리



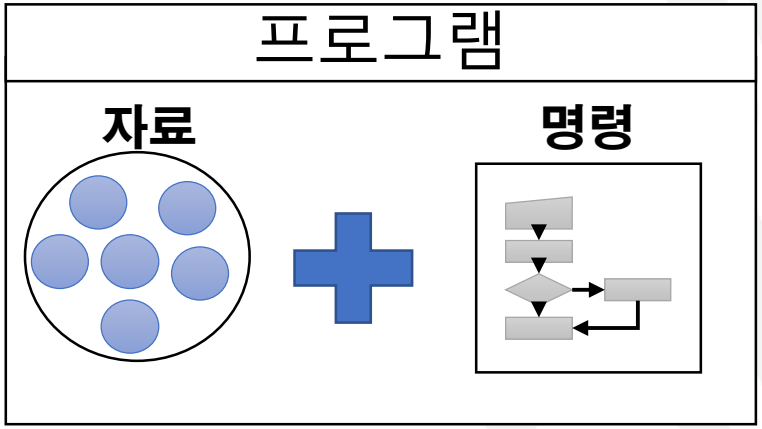




- 사람들이 사물을 정리하는 것과 마찬가지로 프로그램에서도 자료들을 정리하는 여러 구조들이 있음 ➔ 자료구조(data structure)

자료구조 in SW

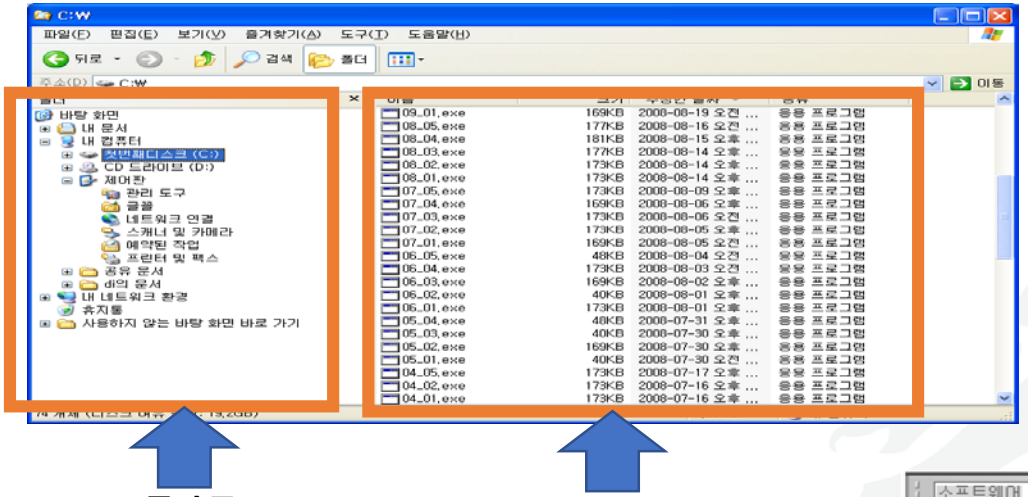
- 프로그램
  - 자료(data) + 명령(algorithm)



예	프로그램 구성	
	명령	자료
윈도우 탐색기	파일의 복사, 이동 및 삭제	파일 및 폴더의 계층 구조 정보
사전 소프트웨어	단어 검색	단어의 철자 및 의미

자료구조의 예 in SW

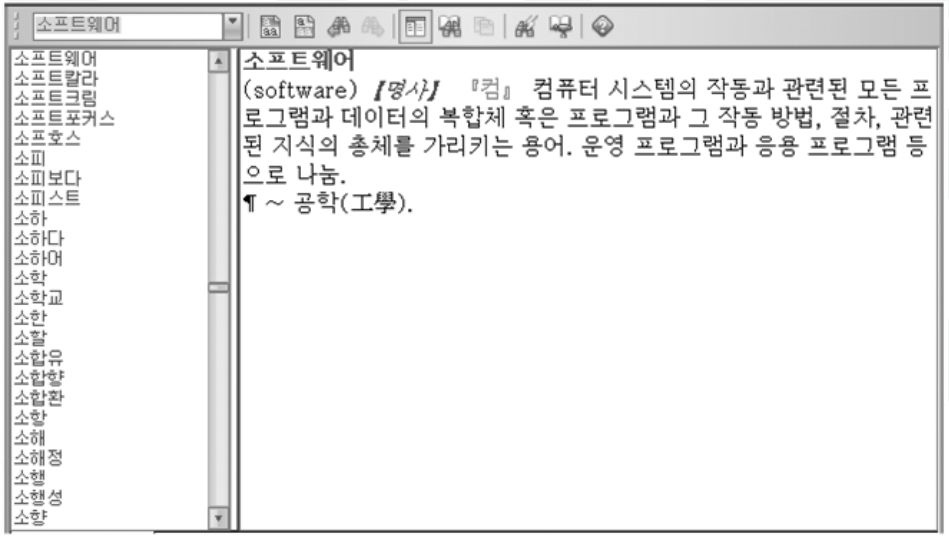
- 윈도우 탐색기



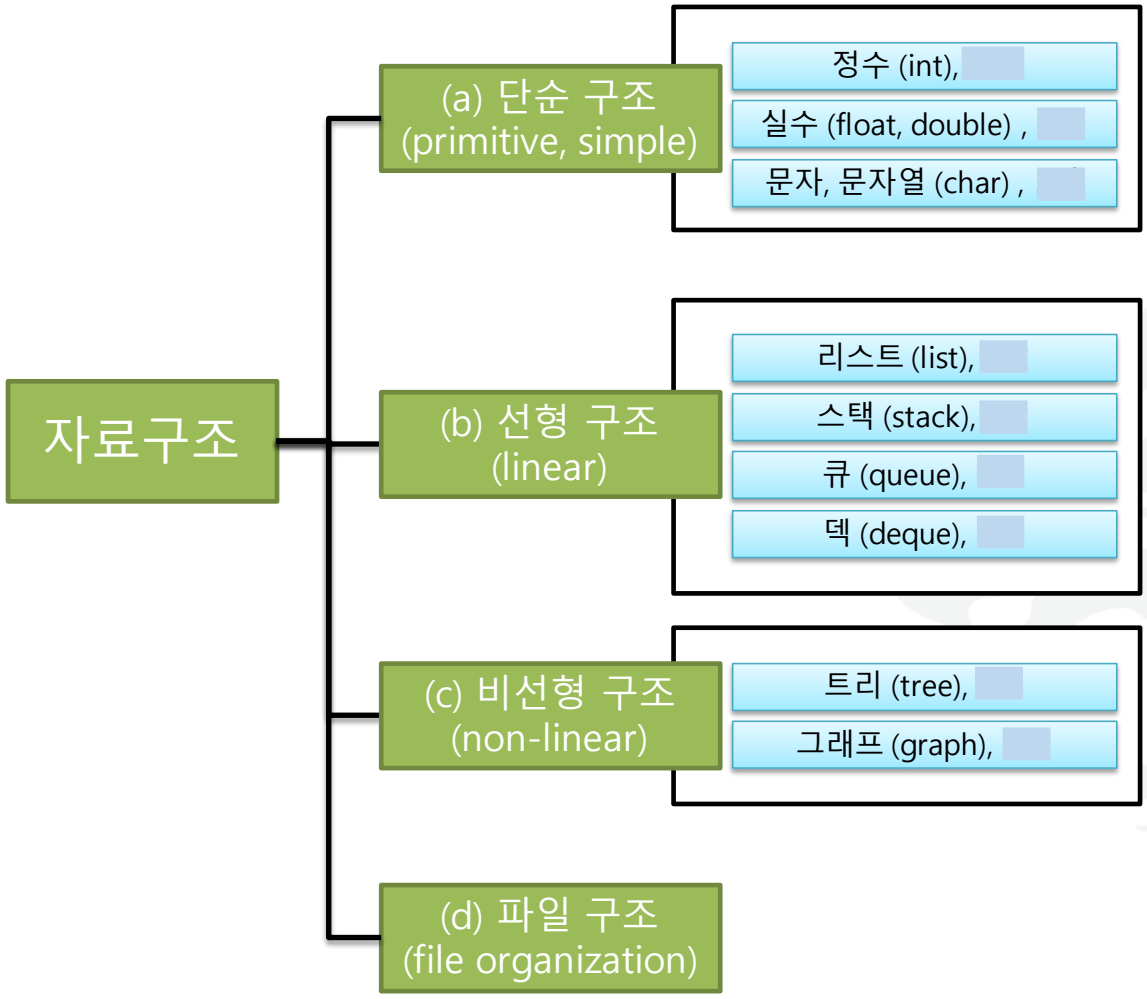
(a) **폴더 구조** (트리)

(b) **파일 목록** (리스트)

- 사전 소프트웨어

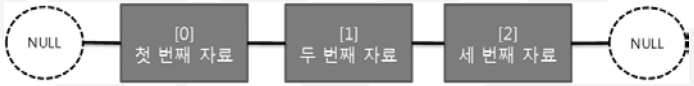


자료구조의 분류

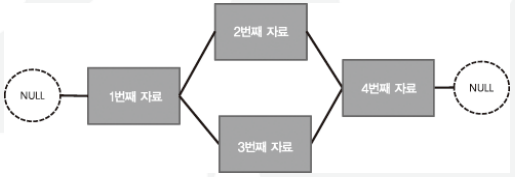


프로그래밍 언어에서 제공하는 기본적인 데이터 타입

자료들 사이의 앞뒤 관계가 일대일(1:1)인 경우



자료들 사이의 앞뒤 관계가 계층 구조(Hierarchical Structure) 혹은 망 구조(Network Structure)를 가지는 경우



데이터 타입

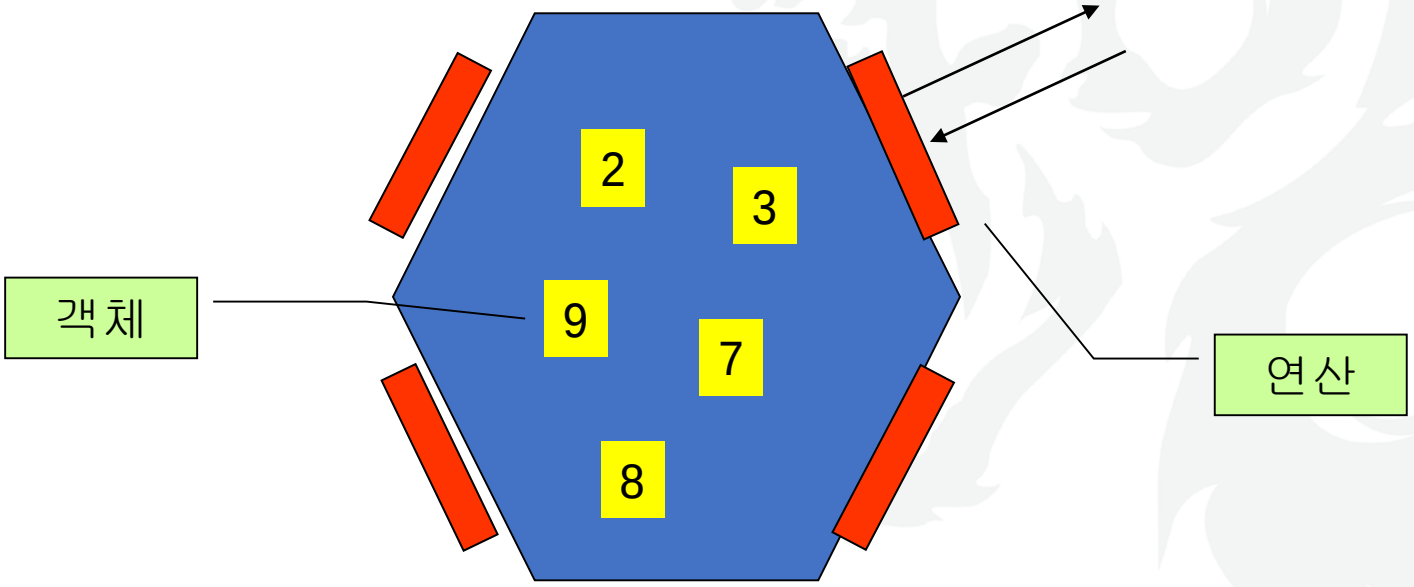
- 데이터 타입(data type, 자료형)
  - 데이터의 집합과 연산의 집합

(예) int 데이터 타입
- 데이터: { …,-2,-1,0,1,2,…}
- 연산: +, -, /, *, %

- 추상 데이터 타입(ADT: Abstract Data Type)
  - 데이터 타입을 추상적(수학적)으로 정의한 것
    - 추상화: 시스템의 간략화 된 기술/명세, 시스템의 핵심 구조와 동작에만 집중
    - 좋은 추상화란?
      - » 중요한 정보는 강조되고, 중요하지 않은 세부 사항은 제거
  - 데이터나 연산이 무엇(what)인가는 정의됨
  - 데이터나 연산을 어떻게(how) 컴퓨터 상에서 구현할 것인지는 정의되지 않음

추상 데이터 타입의 정의

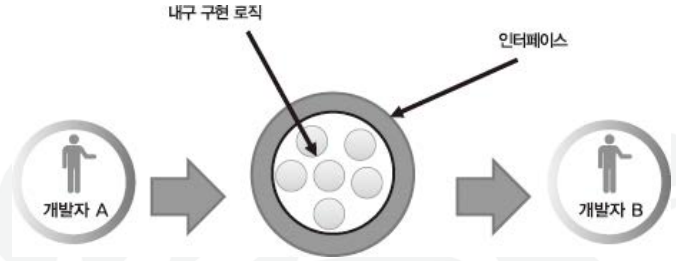
- 객체(Object)
  - 추상 데이터 타입에 속하는 객체 정의
- 연산(Operation)
  - 객체들 사이의 연산을 정의
  - 추상 데이터 타입과 외부를 연결하는 인터페이스의 역할을 함

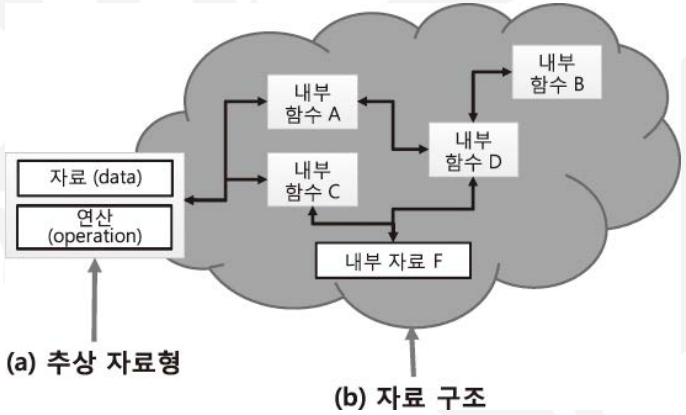


추상 데이터 타입

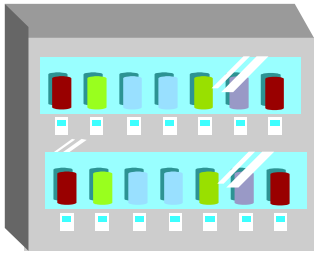
추상 데이터 타입의 특징

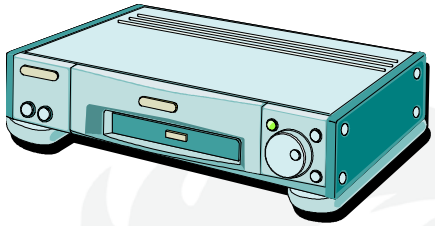
- 자료구조를 기술(표현)하는 가장 대표적인 방법
- 정보 은닉(Information Hiding)
  - 중요한 정보만을 나타내고, 중요하지 않은 정보는 숨김
  - 자료구조를 사용할 수 있는 인터페이스(Interface) 정의
    - 혹은 자료구조의 연산(Operation)들의 정의
- 자료구조의 활용 관점
  - 자료구조를 이용하는 사용자의 관점
    - 자료구조의 내부 구현 소스에 대한 분석 없이 신속하게 자료구조를 이용 가능
    - 예) 이름, 입력, 출력
  - 자료구조를 공급하는 개발자 관점
    - 자료구조를 구현하기 전에 설계도를 미리 그려보는 것





추상 데이터 타입의 이해





- 사용자들은 추상 데이터 타입이 제공하는 연산만을 사용할 수 있다. → VCR의 인터페이스가 제공하는 특정한 작업만을 할 수 있다.
- 사용자들은 추상 데이터 타입을 어떻게 사용하는지를 알아야 한다. → 사용자는 이러한 작업들을 이해해야 한다. 즉 비디오를 시청하기 위해서는 무엇을 해야 하는지를 알아야 한다.
- 사용자들은 추상 데이터 타입 내부의 데이터를 접근할 수 없다. → VCR의 내부를 볼 수는 없다.
- 사용자들은 어떻게 구현되었는지 몰라도 이용할 수 있다. → VCR의 내부에서 무엇이 일어나고 있는지 몰라도 이용할 수 있다.
- 만약 다른 사람이 추상 데이터 타입의 구현을 변경하더라도 인터페이스가 변경되지 않으면 사용할 수 있다. → 누군가가 VCR의 내부의 기계장치를 교환한다고 하더라도 인터페이스만 바뀌지 않는 한 그대로 사용이 가능하다.

추상 데이터 타입의 예

- 자연수 정의

```
Nat_No
객체: 0에서 시작하여 INT_MAX까지의 순서화된 정수의 부분범위
연산:
     zero()     ::=  return 0;
     is_zero()  ::=  if (x) return FALSE;
                     else return TRUE;
     add(x,y)   ::=  if( (x+y) <= INT_MAX ) return x+y;
                     else return INT_MAX
     sub(x,y)   ::=  if ( x<y ) return 0;
                     else return x-y;
     equal(x,y)::=   if( x=y ) return TRUE;
                   else return FALSE;
     successor(x)::= if( (x+y) <= INT_MAX )
                 return x+1;
```

알고리즘의 성능 분석

알고리즘 성능 분석 기법

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

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

수행시간측정

- clock() 함수 사용
  - 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);
}
```

알고리즘 효율성과 복잡도 분석

- 어떤 문제를 해결하는데 사용될 수 있는 알고리즘은?
  - 매우 다양함
- 다양한 알고리즘 중에서 우리는 어떤 것을 골라야 할까?
  - 효율적인 알고리즘
  - 그래서 알고리즘 분석이 필요! 즉, 알고리즘의 효율성을 분석해야 함!
- 알고리즘의 효율성 분석 방법은?
  - 알고리즘의 복잡도 분석(Complexity Analysis)
    - 직접 구현하지 않고 모든 입력을 고려하여 대략적으로 알고리즘 효율성을 비교하는 방법
    - 실행 하드웨어나 소프트웨어 환경과 관계없이 알고리즘의 효율성을 평가할 수 있음

시간 복잡도 분석(1)

- 시간 복잡도 계산 방법
  - 알고리즘을 이루고 있는 연산들이 몇 번이나 수행되는지를 숫자로 표
    - 알고리즘의 절대적인 실행 시간을 분석하는 것이 아님!
  - 산술 연산, 대입 연산, 비교 연산, 이동 연산 등 기본적인 연산 고려
  - 연산의 수행횟수는 고정된 숫자가 아니라 입력의 개수 n에 대한 함수(시간복잡도 함수) → T(n) 이라고 표기함

- 시간 복잡도 함수(time complexity function) $T(n)$
  - n : 문제의 크기(입력자료의 개수)
  - 알고리즘을 수행하는데 필요한 시간을 추정한 것

시간 복잡도 분석(2)

- 최악의 경우의 시간 복잡도 (worst-case time complexity)
  - 크기가 n인 문제에 대한 알고리즘을 분석할 때, 그 알고리즘에 의해 수행되는 기본 연산의 회수를 최대로 하는 입력에 대한 시간복잡도 함수
  - D_n : 크기가 n 인 문제에 대한 모든 입력 집합
  - I : D_n의 원소
  - $t(I)$: 입력 I에 대하여 수행하는 기본 연산 수
  - $T_W(n)$: 최악의 경우의 시간 복잡도, $T_W(n) = \max\{t(I) | I \in D_n\}$
- 평균적인 경우의 시간 복잡도(expected-case time complexity)
  - 알고리즘이 평균적으로 얼마 만큼의 기본 연산을 수행하는 가를 분석
  - $P(I)$: 입력 I 가 일어날 확률
  - $T_E(n)$: 평균적 경우의 시간 복잡도, $T_E(n) = \sum_{I \in D_n} p(I)t(I)$

복잡도 분석의 예

- n을 n번 더하는 문제: 각 알고리즘이 수행하는 연산의 개수를 세어 봄. 단, for 루프 제어 연산은 고려하지 않음.

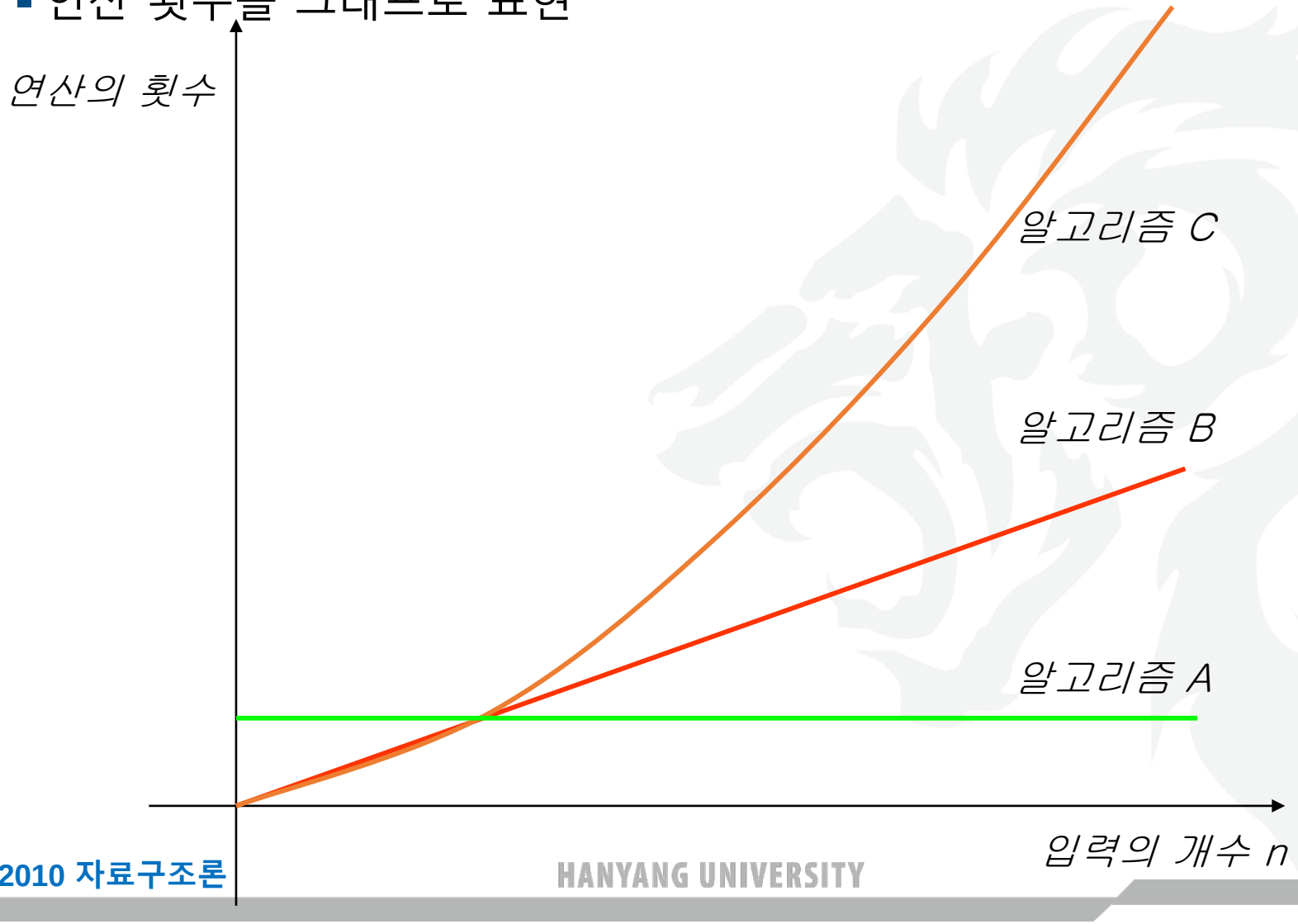
알고리즘 A	알고리즘 B	알고리즘 C

`sum ←n*n;`	`sum ← 0;`
	`for i ← 1 to n do`
	`  sum ←sum + n;`
		`sum ← 0;`
		`for i←1 to n do`
		`  for j←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**	**$2n^2 + 1$**

복잡도 분석의 예

- 연산 횟수를 그래프로 표현



시간 복잡도 함수 계산 예

- 코드를 분석해보면 수행되는 수행되는 연산들의 횟수를 입력 크기의 함수로 만들 수 있음

```
ArrayMax(A,n)

  tmp ← A[0];              1번의 대입 연산
  for i←1 to n-1 do            루프 제어 연산은 제외
    if tmp < A[i] then         n-1번의 비교 연산
        tmp ← A[i];                n-1번의 대입 연산(최대)
  return tmp;                      1번의 반환 연산

                                   총 연산수= 2n(최대)
```

빅오 표기법(1)

- 자료의 개수가 많은 경우에는 차수가 가장 큰 항이 가장 영향을 크게 미치고 다른 항들은 상대적으로 무시될 수 있음
  - (예) n=1,000 일 때, $T(n)$의 값은 1,001,001이고 이중에서 첫 번째 항인 의 값이 전체의 약 99%인 1,000,000이고 두 번째 항의 값이 1000으로 전체의 약 1%를 차지함
  - 따라서 보통 시간 복잡도 함수에서 가장 영향을 크게 미치는 항만을 고려하면 충분함

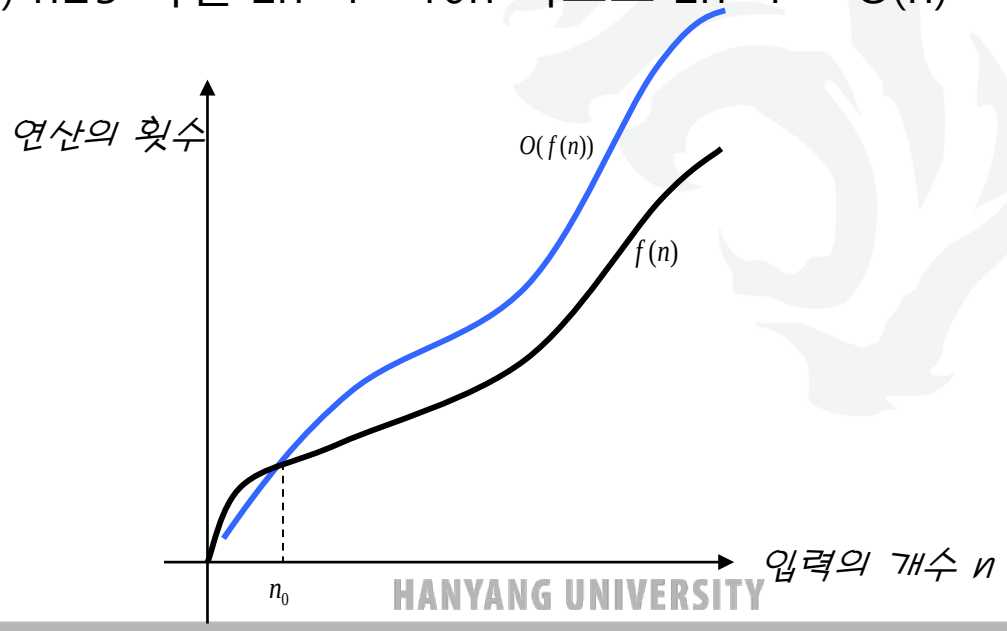
n=1000인 경우

$T(n) = n^2 + n + 1$ 입력의 개수 n

99% 1%

빅오 표기법(2)

- **빅오(big-oh) 표기법**: 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 두개의 상수 c와 n_0가 존재하면 f(n)=O(g(n))
  - "빅오"는 **함수의 상한**을 표시
    - (예) $n \geq 5$ 이면 $2n+1 < 10n$ 이므로 $2n+1 = O(n)$ ← Big-oh of n

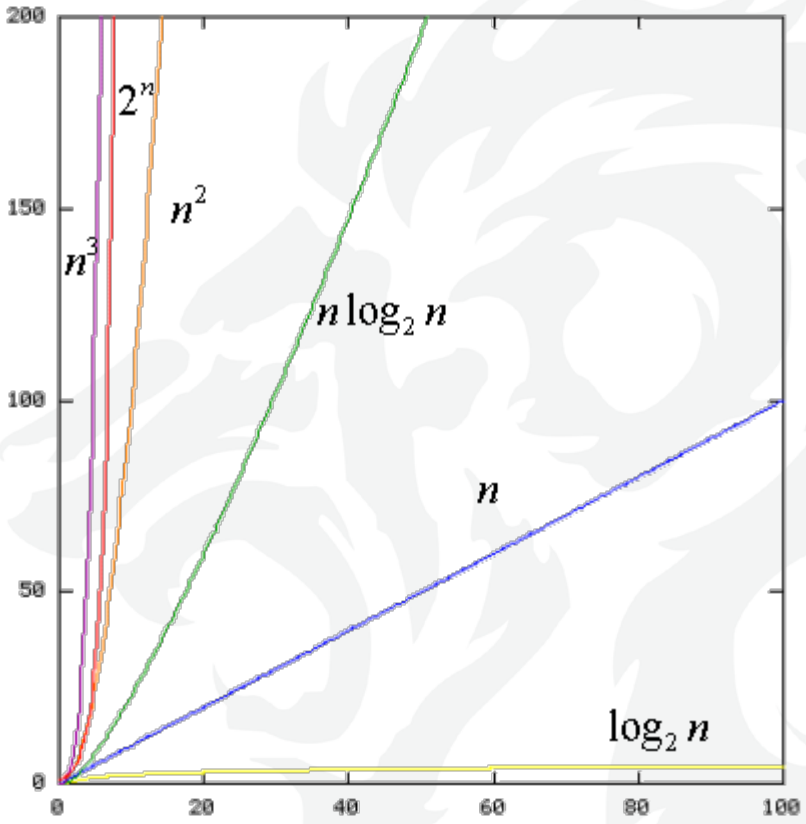


빅오 표기법 예제

- $f(n)=5$이면 O(1)이다. 왜냐하면 $n_0=1$, $c=10$일 때, $n \geq 1$에 대하여 $5 \leq 10 \cdot 1$이 되기 때문이다.
- $f(n)=2n+1$이면 O(n)이다. 왜냐하면 $n_0=2$, $c=3$일 때, $n \geq 2$에 대하여 $2n+1 \leq 3n$이 되기 때문이다.
- $f(n)=3n^2+100$이면 O(n**2)이다. 왜냐하면 $n_0=100$, $c=5$일 때, $n \geq 100$에 대하여 $3n^2+100 \leq 5n^2$이 되기 때문이다.
- $f(n)=5 \cdot 2^n+10n^2+100$이면 O(2**n)이다. 왜냐하면 $n_0=1000$, $c=10$일 때, $n \geq 1000$에 대하여 $5 \cdot 2^n+10n^2+100 \leq 10 \cdot 2^n$이 되기 때문이다.

빅오 표기법 종류

- O(1) : 상수형
- O(logn) : 로그형
- O(n) : 선형
- O(nlogn) : 로그선형
- $O(n^2)$: 2차형
- $O(n^3)$: 3차형
- $O(n^k)$: k차형
- $O(2^n)$: 지수형
- O(n!) : 팩토리얼형



함수 수행속도 비교(1)

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}

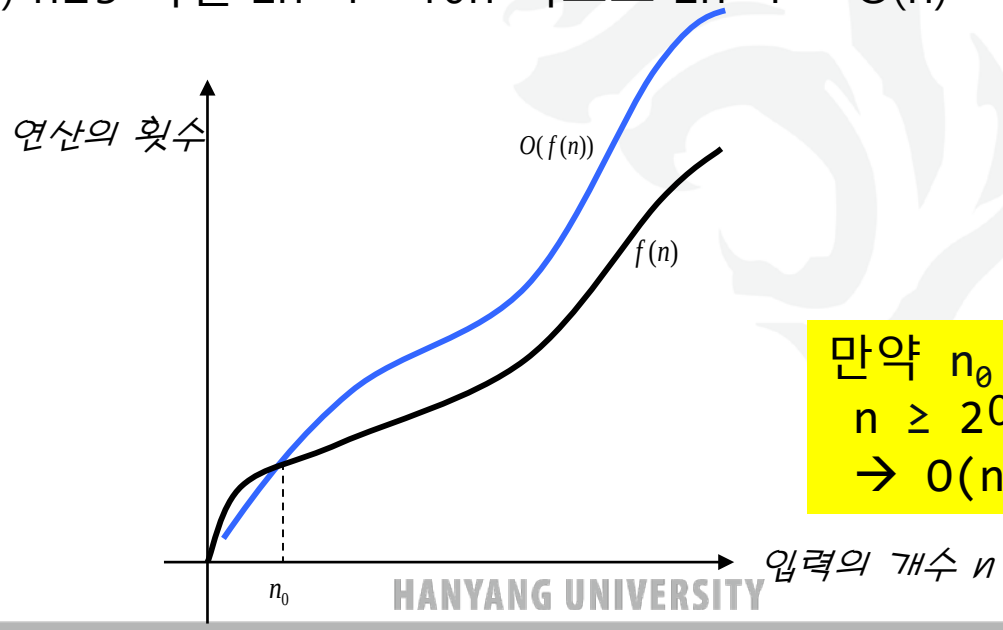
함수 수행속도 비교(2)

(단위 연산속도 : 10^{-6}초)

입력 / 알고리즘 / 복잡도 / 크기 n	A	B	C	D	E
	$100n$	$10nlog_2n$	$5n^2$	n^3	2^n
10	10^{-3} 초	1.5*10^{-3} 초	5*10^{-4} 초	10^{-3} 초	10^{-3}초
100	10^{-2} 초	0.03 초	5*10^{-2} 초	1 초	4*10^{14}세기
1,000	10^{-1} 초	0.45 초	5 초	1.6 분	***
10,000	1 초	6.1 초	8.3 분	11.57 일	***
100,000	10 초	1.5 분	13.8 시간	31.7 년	***

빅오 표기법 – 함수의 상한 표시

- **빅오(big-oh) 표기법**: 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 두개의 상수 c와 n_0가 존재하면 f(n)=O(g(n))
  - "빅오"는 **함수의 상한**을 표시
    - (예) $n \geq 5$ 이면 $2n+1 < 10n$ 이므로 $2n+1 = O(n)$ ← Big-oh of n



만약 $n_0 = 2$, $c = 2$로 잡으면,
$n \geq 2$에 대해서 $2n+1 \leq 2n^2$
→ $O(n^2)$

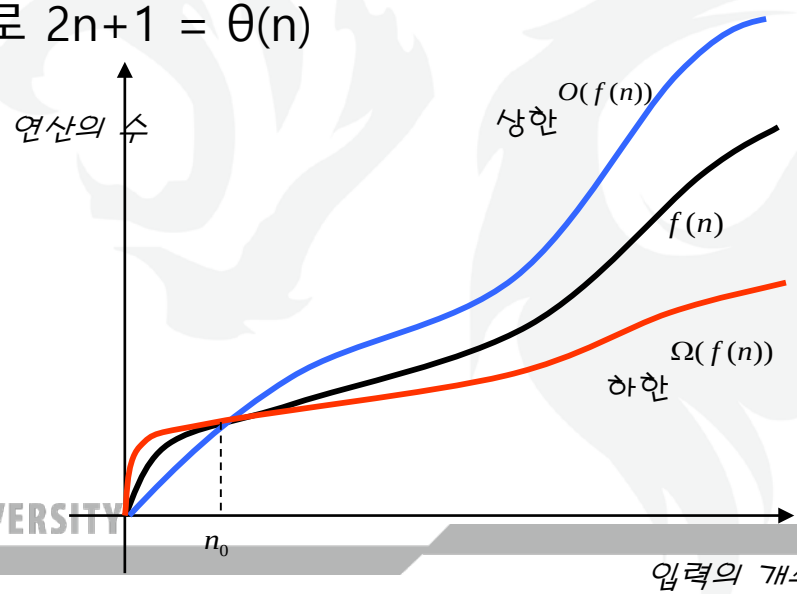
빅오메가 표기법

- **빅오메가(big omega) 표기법**
  - 두개의 함수 f(n)과 g(n)이 주어졌을 때,

    모든 $n \geq n_0$에 대하여 $|f(n)| \geq c|g(n)|$을 만족하는 두개의 상수 c와 n_0가 존재하면 $f(n) = \Omega(g(n))$
  - "빅오메가"는 **함수의 하한**을 표시
    - (예) $n \geq 1$ 이면 $2n+1 \geq n$ 이므로 $2n+1 = \Omega(n)$

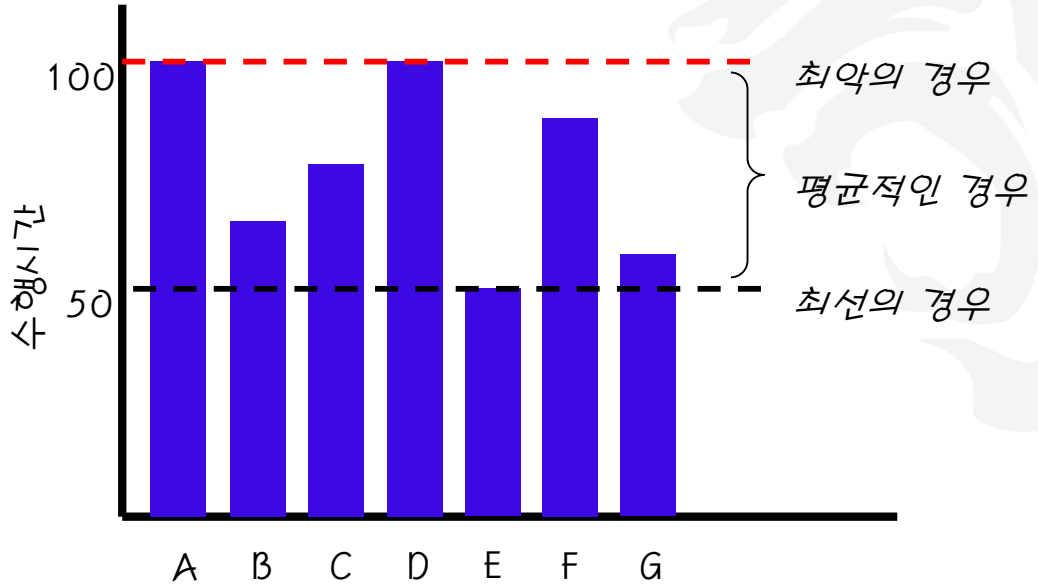
빅세타 표기법

- **빅세타(big theta) 표기법**
  - 두개의 함수 f(n)과 g(n)이 주어졌을 때,
    모든 $n \geq n_0$에 대하여 $c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|$ 을 만족하는 세개의 상수 c_1, c_2 와 n_0가 존재하면 $f(n) = \theta(g(n))$
  - "빅세타"는 **함수의 하한인 동시에 상한**을 표시
  - f(n)=O(g(n))이면서 $f(n) = \Omega(g(n))$이면, $f(n) = \theta(n)$
  - (예) $n \geq 1$이면 $n \leq 2n+1 \leq 3n$이므로 $2n+1 = \theta(n)$



최선, 평균, 최악의 경우

- 알고리즘의 수행시간은 입력 자료 집합에 따라 다를 수 있음
  - (예) 정렬 알고리즘
- **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



순차탐색 알고리즘 예

5	9	10	17	21	29	33	37	38	43

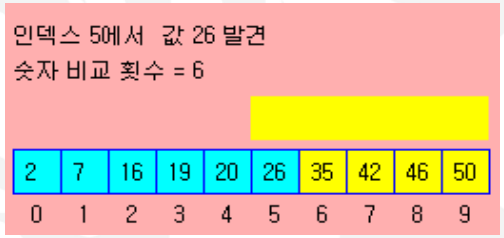
- 최선의 경우
  - 찾고자 하는 숫자가 맨앞에 있는 경우
    ∴ O(1)

- 최악의 경우
  - 찾고자 하는 숫자가 맨뒤에 있는 경우
    ∴ O(n)

- 평균적인 경우
  - 각 요소들이 균일하게 탐색된다고 가정하면 모든 숫자들이 탐색되었을 경우의 비교 연산 수행횟수를 더한 후, 전체 숫자 개수로 나누면 됨

    (1+2+…+n)/n=(n+1)/2

    ∴ O(n)



C언어 리뷰

1. main()

```
int main(void)
```

- 식별자 다음에 괄호()가 오면 그 식별자는 함수라는 것을 나타냄
- 프로그램은 함수로 구성됨
- 프로그램의 수행은 항상 main() 함수로부터 시작함
- main() 함수 정의
  - → void - 인수를 받지않음
  - → int형 return값을 가짐

printf()

```
printf("서식지정문자열", 변수);
```

- 화면출력용 함수
- printf()는 출력된 문자의 수를 int형으로 리턴
  (오류 발생 시 음수값 리턴)
- 서식지정 문자열
  → 일반문자열, 변환문자열(%), 확장문자열(\)

```
printf("%변환문자", 변수);
```

- printf()의 변환문자열

printf() 변환 문자

printf() 변환문자	
c	as a character (문자)
d	as a decimal integer (10진 정수)
ld	as a long type decimal integer (long형 10진 정수)
e	as a floating point number in scientific notation (지수형)
f	as a floating point number (float, double)
g	in the e-format or f-format, whichever is shorter
s	as a string (문자열)

printf()의 사용

```
printf("abc");
printf("%s", "abc");
printf("%c%c%c", 'a','b','c');
```

- 화면에 abc 출력
- 'a'는 소문자 a에 해당하는 문자 상수이다.

변환문자의 옵션

변환문자의 옵션 지정		
%[필드폭].[자릿수][변환문자]		
%d	→	123
%5d	→	␣␣123
%10d	→	␣␣␣␣␣␣␣123
%2d	→	123 (지정 필드폭의 칸수가 필요한 자릿수보다 작아도 필요한 숫자는 모두 출력)
%f	→	654.321000 (표준출력, 소수점 이하 6자리)
%12f	→	␣␣654.321000 (12칸에 출력, 소수점 이하는 6자리로 표준출력)
%9.2f	→	␣␣␣654.32 (9칸에 출력, 소수점 이하는 2자리로 출력)

scanf()

```
scanf("서식지정문자열", 변수주소);
```

- 키보드 입력용 함수
- scanf()함수는 성공적으로 입력된 횟수를 int형을 리턴

```
scanf("%d", &x);
```

- &기호는 주소연산자로 &x는 "x의 주소"라고 읽음
- %d는 x가 해석될 방식에 상응하는 형식으로, 입력 문자열을 10진 정수로 해석하여 x의 주소에 결과값을 저장함

scanf() 변환문자

scanf() 변환문자	
c	to a character (문자)
d	to a decimal integer (10진 정수)
ld	to a long type decimal integer (long형 10진 정수)
f	to a floating point number (float)
lf	to a floating point number (double)
LF	to a floating point number (long double)
s	to a string (문자열)

printf()와 scanf()의 사용 예

```
/* echo.c */
#include <stdio.h>

int main(void)
{
    char    c1, c2, c3;
    int     i;
    float   x;
    double  y;

    printf("\n%s\n%s", "Input three characters,"
        "an int, a float, and a double: ");
    scanf("%c%c%c%d%f%lf", &c1, &c2, &c3, &i, &x, &y);
    printf("\nHere is the data that you typed in:\n");
    printf("%3c%3c%3c%5d%17e%17e\n\n", c1, c2, c3, i, x, y);
    return 0;
}
```

printf()와 scanf()의 사용 예

```
Input three characters,
an int, a float, and a double:  ABC␣␣3␣␣55␣␣77.7
Here is the data that you typed in:
  A  B  C    3     5.500000e+01     7.770000e+01
```

변수, 수식, 배정

```
/* The distance of a marathon in kilometers. */

#include <stdio.h>

int main(void)
{
    int     miles, yards;
    float   kilometers;

    miles = 26;
    yards = 385;
    kilometers = 609 * (miles + yards / 1760.0);
    printf("\nA marathon is %f kilometers.\n\n", kilometers);
    return 0;
}
```

```
A marathon is 42.185970 kilometers.
```

변수, 수식, 배정

```
int miles, yards;
```

- 선언문: 변수 miles, yards는 정수값을 가지는 변수

```
float kilometers;
```

- 선언문: 변수 kilometers는 실수값(유효숫자 6자리)을 가지는 변수
- 모든 변수는 선언하고 나서 사용

```
miles = 26;
yards = 385;
```

- 배정문: 정수형 상수 26과 385가 변수 miles와 yards에 배정

변수, 수식, 배정

```
kilometers = 609 * (miles + yards / 1760.0);
```

- 배정문
- *, +, /: 연산자 (-, %, …)

```
printf("\nA marathon is %f kilometers.\n\n", kilometers);
```

- 변환형식 %f와 인자 kilometers는 짝을 이루며, kilometers의 값이 보통 소수점(float) 형식 %f의 위치에 출력됨
- 변수의 값을 출력하려면 서식지정이 필요함
- 수식의 변환 규칙(conversion rule)
  - 7/2 → 3
  - 7.0/2 → 3.5

#define과 #include의 사용

```
#define LIMIT 100
#define PI 3.14159
#define C 299792.458 /* speed of light in km/sec */
```

- #: 전처리기 지시자(preprocessing directive)
- LIMIT, PI, C: 심볼릭 상수(symbolic constant)

```
#include "my_file.h"
```

- 코드에 my_file.h 파일의 사본 포함
- C에서 제공하는 표준 헤더파일
  - → stdio.h, string.h, math.h, …, <xxx.h>

#define과 #include의 사용

```
/* pacific_sea.h */
#include <stdio.h>

#define   AREA                        2337
#define   SQ_MILES_PER_SQ_KILOMETER   0.3861021585424458
#define   SQ_FEET_PER_SQ_MILE         (5280 * 5280)
#define   SQ_INCHES_PER_SQ_FOOT       144
#define   ACRES_PER_SQ_MILE           640
```

#define과 #include의 사용

```
/* pacific_sea.c */
#include "pacific_sea.h"

int main(void)
{
    const int    pacific_sea = AREA;     /* in sq kilometers */
    double       acres, sq_miles, sq_feet, sq_inches;

    printf("\nThe Pacific Sea covers an area");
    printf(" of %d square kilometers.\n", pacific_sea);
    sq_miles = SQ_MILES_PER_SQ_KILOMETER * pacific_sea;
    sq_feet = SQ_FEET_PER_SQ_MILE * sq_miles;
    sq_inches = SQ_INCHES_PER_SQ_FOOT * sq_feet;
    acres = ACRES_PER_SQ_MILE * sq_miles;
    printf("In other units of measure this is:\n\n");
    printf("%22.7e acres\n", acres);
    printf("%22.7e square miles\n", sq_miles);
    printf("%22.7e square feet\n", sq_feet);
    printf("%22.7e square inches\n", sq_inches);
    return 0;
}
```

#define과 #include의 사용

assign이 불가능하다

```
const int pacific_sea = AREA; /* in sq kilometers */
```

- const: ANSI C에 소개된 한정자, 초기화 이후 값 변경 불가

```
double acres, sq_miles, sq_feet, sq_inches;
```

- double: 유효숫자 15자리 (float는 6자리)

```
printf("%22.7e acres\n",acres);
5.7748528e+05 acres
```

-> 5.7748528×10^5

if 문

- 일반적인 형태

```
if (expr)
    statement
```

  - 조건식(*expr*)이 참(true)이면 문장(*statement*) 실행
  - false: zero, true: non-zero
  - 단문이면 {} 생략

```
a = 1;
if (b == 3)
    a = 5;
printf("%d", a);
```

  - b가 3이면 a=5
  - b가 3이 아니면 문장(a=5) 실행 안함, printf() 문 실행 시 1 출력

if-else 문

- 일반적인 형태

```
if (expr)
    statement1
else
    statement2
```

  - 조건식이 참이면 (*expr*이 0이 아니면) 문장1 실행, 그렇지 않으면 문장2 실행
  - 여러 문장을 포함해도 if-else문 전체가 하나의 문장

예문

```
if (cnt == 0) {
    a = 2;
    b = 3;
    c = 5;
}
else {
    a = -1;
    b = -2;
    c = -3;
}

printf("%d", a + b + c);
```

→ cnt 가 0값을 가지면 10 출력, 그렇지 않으면 -6 출력

while 루프

- 일반적인 형태

```
while (expr)
    statement
```

  - 조건식이 참이면(*expr*이 0이 아니면) 문장 실행 후, while 루프 처음으로 복귀, *expr*이 0이 될 때까지 반복

while 루프

```
/* consecutive_sums.c */
#include <stdio.h>

int main(void)
{
    int  i = 1, sum = 0;
    while (i <= 5) {
        sum += i;
        ++i;
    }
    printf("sum = %d\n", sum);
    return 0;
}
```

while 루프

```
while (i <= 5) {
    sum += i;
    ++i;
}
```

→ <=는 *less than or equal to*

```
variable op= expr
variable = variable op expr
```

- sum += i;
  - → sum = sum + i;
- ++i;
  - → ++i 증가 --i; 감소
  - → i = i + 1; i = i - 1;

for 루프

- 일반적인 형태

```
for (expr1; expr2; expr3)
    statement
```

```
expr1;
while (expr2) {
    statement
    expr3;
}
```

- → *expr1* 초기화 배정한 후, *expr2*를 검사하여 0이 아닌 경우, statement를 수행한 후, *expr3*으로 저장 값을 증가시키고 다시 *expr2* 검사하면서 0이 아닌 동안 반복
- → *expr3*이 루프에서 가장 마지막에 실행됨

for 루프

```
#include <stdio.h>
int main(void)
{
    int sum=0;
    for (i=1; i<=5; ++i ) {
        sum+=i;
    }
    printf("sum= %d \n", sum);
    return 0;
}
```

함수

- C프로그램은 여러 파일들을 가질 수 있으며, 각각의 파일은 여러 함수들을 가질 수 있음
- main()함수
  - → 이 함수로부터 프로그램 시작, main()함수에서 다른 함수가 호출되어 프로그램이 구성됨

```
int main(void)
{
     ...
     subfunction1(arguments);
     ...
}

...

subfunction1(arguments)
{
     ...
}
```

함수

- 함수 원형(function prototype)

```
type function_name(parameter type list);
```

  - 함수는 사용되기 전 선언되어야 하는 데, 이런 함수선언형식을 함수 원형이라고함
  - 컴파일러는 함수원형을 통해 함수에 전달될 인자의 수와 형, 그리고 함수에서 리턴될 값의 형을 알 수 있음
  - function_name이 함수의 이름, type형의 리턴값을 가짐, parameter type list는 콤마로 분리된 형들의 목록
  - 이 목록에서 식별자 사용은 옵션 (함수 원형에 영향없음)
  - 인자 혹은 리턴값이 없을 경우 void 사용
  - 인자의 개수가 가변적일 때에는 ... 사용

- 예제(stdio.h에 정의된 printf()의 원형)

```
int printf(const char * format, ...);
```

6. Functions

```c
/* maxmin.c */
#include <stdio.h>

float    maximum(float x, float y);
float    minimum(float x, float y);
void     prn_info(void);

int main(void)
{
    int      i, n;
    float    max, min, x;

    prn_info();
    printf("Input n:    ");
    scanf("%d", &n);
    printf("\nInput %d real numbers:  ", n);
    scanf("%f", &x);
    max = min = x;
    for (i = 2; i <= n; ++i) {
        scanf("%f", &x);
        max = maximum(max, x);
        min = minimum(min, x);
    }
    printf("\n%s%13f\n%s%13f\n\n",
        "Maximum value:", max,
        "Minimum value:", min);
    return 0;
}
```

6. Functions

```
float maximum(float x, float y)
{
    if (x > y)
        return x;
    else
        return y;
}

float minimum(float x, float y)
{
    if (x < y)
        return x;
    else
        return y;
}

void prn_info(void)
{
    printf("\n%s\n%s\n\n",
        "This program reads an integer value for n, and then",
        "processes n real numbers to find max and min values.");
}
```

함수의 선언과 정의

- 함수 선언(function declaration)

```
float   maximum(float x, float y);
```

  → 컴파일러에게 maximum()함구가 2개의 float형 인자를 가지고, 리턴값은 float형이라는 것을 알려줌

- 함수 정의(function definition)

```
float maximum(float x, float y)
{
    if (x > y)
        return x;
    else
        return y;
}
```

  → 이 함수가 호출될 때, 실제 실행될 작업을 명확하게 기술

Call-by-value

```
#include <stdio.h>

int main(void)
{
    int   a = 1;
    void  try_to_change_it(int);

    printf("%d\n", a);     /* 1 is printed */
    try_to_change_it(a);
    printf("%d\n", a);     /* 1 is printed again! */
    return 0;
}

void try_to_change_it(int a)
{
    a = 777;
}
```

배열 (Arrays)

- C에서 문자열(string)은 문자(character)의 배열(array)이고 배열이름 자체가 하나의 포인터(pointer)임
- 배열(Arrays)
  - → 배열은 동일한 형을 갖고 개수가 많은 변수가 요구될 때 사용

```
int a[3]
```

- 이 배열은 int형 원소 a[0], a[1], a[2]로 구성
- 배열의 첨자는 항상 0부터 시작

배열 예

```
/* sorting program */
...
int i, j, score[CLASS_SIZE], sum=0, tmp;
printf("Input %d scores: ", CLASS_SIZE);
for(i = 0; i < CLASS_SIZE; ++i) {
    scanf("%d", &score[i]);
    sum += score[i];
}
for(i = 0; i < CLASS_SIZE - 1; ++i) {
    for(j = CLASS_SIZE - 1; j > i; --j) {
        if(score[j-1] < score[j]) {
            tmp = score[j-1];
            score[j-1] = score[j];
            score[j] = tmp;
        }
    }
}
```

배열 예

```
Input 5 scores: 63 88 97 53 77
ordered scores :
    score[0] = 97
    score[1] = 88
    score[2] = 77
    score[3] = 63
    score[4] = 53
378 is the sum of all the scores
75.6 is the class average
```

포인터 (Pointers)

- 포인터는 메모리에 있는 한 대상의 주소
- 배열명은 그것 자체가 하나의 포인터
- 다음에 나오는 프로그램은 배열과 포인터의 관계에 대해 설명하기 위해 설계됨

포인터

```
/* the relationship between arrays and pointers */
...
char    c='a', *p, s[MAXSTRING];

p = &c;
printf("%c%c%c  ", *p, *p + 1, *p + 2);
strcpy(s, "ABC");
printf("%s  %c%c%s\n", s, *s + 6, *s + 7, s + 1);
strcpy(s, "she sells sea shells by the seashore");
p = s + 14;
for( ; *p != '\0'; ++p) {
    if(*p == 'e')
        *p='E';
    if(*p == ' ')
        *p='\n';
}
printf("%s\n", s);
...
```

Result

```
abc  ABC  GHBC
she sells sea shElls
by
thE
sEashorE
```

배열 & 포인터

- C에서 배열, 문자열, 포인터는 밀접하게 관련되어 있음

```
char *p, s[100];
```

- 'p' 가 포인터 변수인데 반해, s는 s[0]을 포인트하는 포인터 상수
- s[i] 와 *(s + i)는 동등
- 이와 유사하게 p[i] 와 *(p + i)도 동등

파일

```
#include <stdio.h>

int main(void)
{
    int  c;
    FILE *ifp;

    ifp = fopen("my_file", "r");
    ...
```

파일

- Main()의 몸체에서 두 번째 줄은 FILE형 포인터인 ifp(“infile pointer”의 약자)를 선언한다. FILE 형은 특별한 구조로 stdio.h 에 정의되어 있음

```
ifp = fopen("my_file", "r");
```

- fopen() 함수는 인자로 두 개의 문자열을 취하며, FILE형 포인터를 리턴함
- 첫 번째 인자는 파일명이고, 두 번째 인자는 파일이 오픈될 모드
  - → 두 번째 인자가 “w”인 경우에는 파일 쓰기를 위한 모드
- fopen() 함수는 표준 라이브러리에 존재하며, 이것이 함수원형은 stdio.h 에 있음
- 만약 어떤 이유로 파일에 접근할 수 없다면, fopen()은 NULL 포인터를 리턴

명령어라인인자

- 명령어 라인 인자를 프로그램 안에서 받아들일 수 있는 방법

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    ...
```

- 인자 argc는 "인자숫자(argument count)"를 의미
- 인자 argv는 "인자변수(argument variable)"를 의미

Week 2: Algorithm, Simple Review of C