

- 교육자료: <https://algo.datahub.pe.kr/>
- 소스코드: <https://algo.datahub.pe.kr/src/>
- Smart OJ: <https://oj.datahub.pe.kr/>

인생 첫 알고리즘 프로그래밍

< 정올 맛보기 >

충북교육연구정보원 정보영재교육원 | SW·AI교실 | 정보아카데미 강사

흥덕고등학교 교사 박정진

C언어

■ 역사

- 1963년 - ALGOL 60에서 CPL이 파생
- 1969년 - BCPL 개발
- 1970년 - Ken Thompson에 의해 B 언어
- 1972년 - 벨 연구소 (Bell Laboratories)에서 C 언어로 개발
- 1983년 - 미국 국가 표준 협회(ANSI)에서 C 언어 표준을 제정
- 1983년 12월 14일 - ANSI X3.159-1983 표준으로 제정
- 1999년 - C99 표준안이 ISO/IEC 9899:1999라는 명칭으로 출간됨
- 2000년 5월 - ANSI의 표준으로 C99가 채택됨
- 2011년 - 12월 8일 C11 표준안이 ISO/IEC 9899:2011라는 명칭으로 출간됨
- 2018년 - C17 표준안이 ISO/IEC 9899:2018이라는 명칭으로 출간됨



C언어

■ 특징 개괄

- C언어는 Unix 운영체제를 개발하기 위해 만들어 졌기 때문에 시스템 프로그래밍을 작성하는데 필요한 기능을 가지고 있다.
- 문서 및 파일 처리, 수식 계산, 그래픽 등과 같은 거의 모든 분야에서 사용할 수 있다.
- C언어는 강력한 기능과 비교적 쉽게 배울 수 있기 때문에 아주 빠른 속도로 보급되었다.
- 세세한 부분까지 프로그램으로 제어 가능하다.
- 하드웨어 제어가 가능하면서도 이식성이 좋기 때문에 전산 이론 및 실무에서 폭넓게 사용되고 있는 현대적인 언어이다.
- 컴파일러 방식을 사용한다.

Binary releases - Code::Blocks

codeblocks.org/downloads/binaries/

Code::Blocks

The IDE with all the features you need, having a consistent look, feel and operation across platforms.

News

Features

Downloads

User manual

Forums

Wiki

License

Donations

GPLv3

W3C CSS

Get Firefox

SOURCEFORGE

Support this project

Microsoft Windows

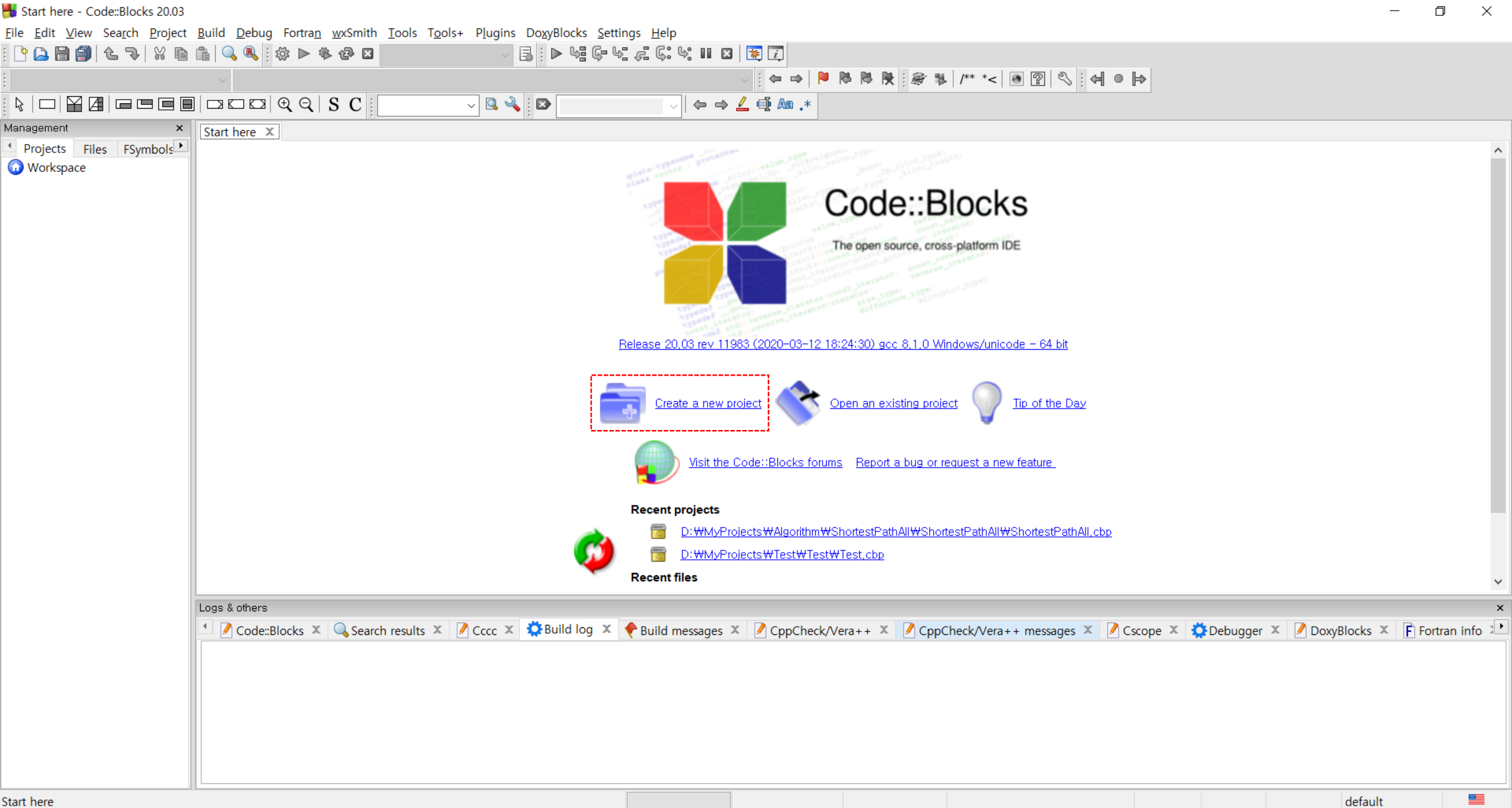
File	Download from
 codeblocks-20.03-setup.exe	FossHUB or Sourceforge.net
 codeblocks-20.03-setup-nonadmin.exe	FossHUB or Sourceforge.net
 codeblocks-20.03-nosetup.zip	FossHUB or Sourceforge.net
 codeblocks-20.03mingw-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03mingw-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-setup-nonadmin.exe	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03mingw-32bit-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03mingw-32bit-nosetup.zip	FossHUB or Sourceforge.net

NOTE: The codeblocks-20.03-setup.exe file includes Code::Blocks with all plugins. The codeblocks-20.03-setup-nonadmin.exe file is provided for convenience to users that do not have administrator rights on their machine(s).

NOTE: The codeblocks-20.03mingw-setup.exe file includes additionally the GCC/G++/GFortran compiler and GDB debugger from [MinGW-W64 project](#) (version 8.1.0, 32/64 bit, SEH).

NOTE: The codeblocks-20.03(mingw)-nosetup.zip files are provided for convenience to users that are allergic against installers. However, it will not allow to select plugins / features to install (it includes everything) and not create any menu shortcuts. For the "installation" you are on your own.

If unsure, please use codeblocks-20.03mingw-setup.exe!



New from

Project
Build t
Files
Custom
User te

TIP: Try

1. Sele
2. Sele
3. Press

Console application

 **Console**

Please select the compiler to use and which configurations you want enabled in your project.

Compiler:
GNU GCC Compiler

☒ Create "Debug" configuration: Debug

"Debug" options

Output dir.: bin\Debug\

Objects output dir.: obj\Debug\

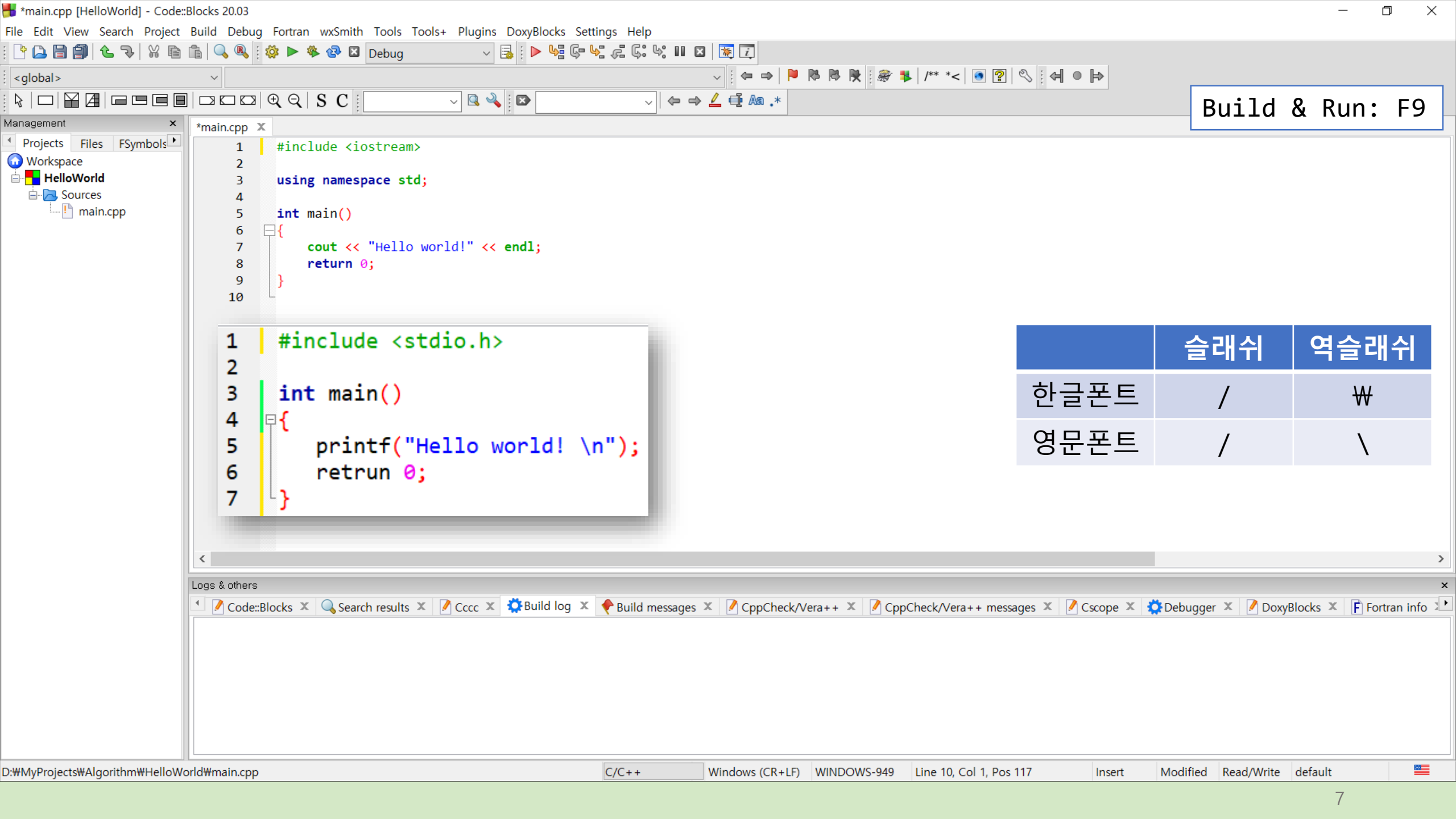
☒ Create "Release" configuration: Release

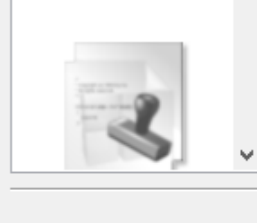
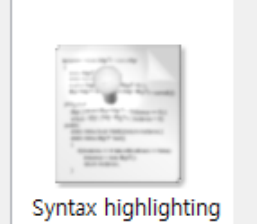
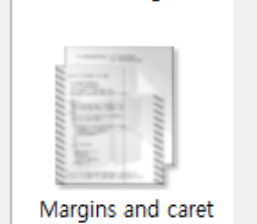
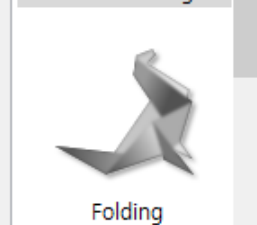
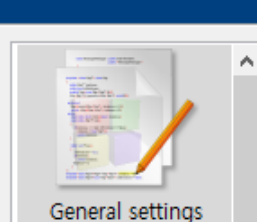
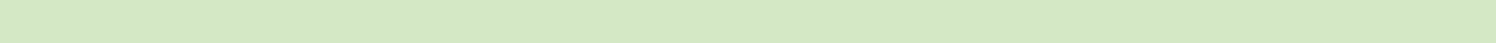
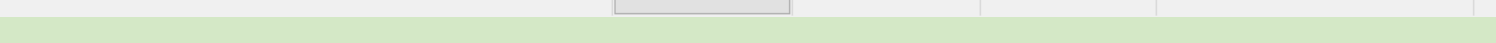
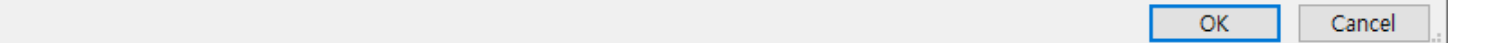
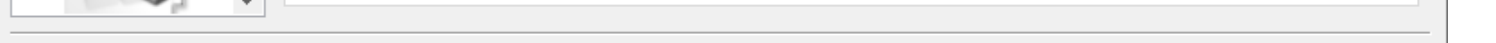
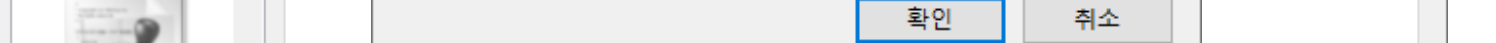
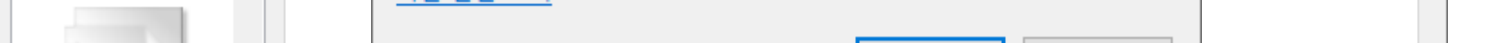
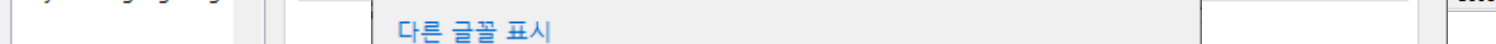
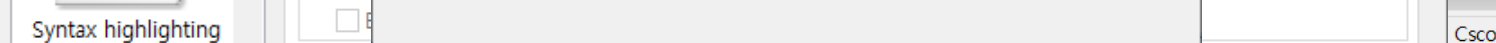
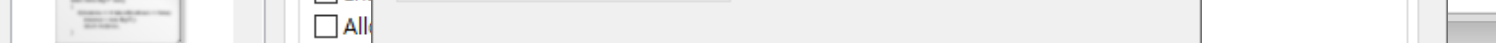
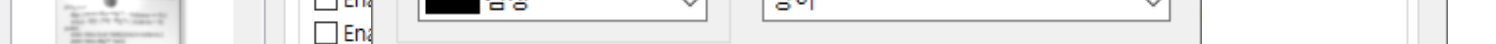
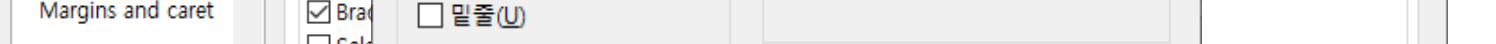
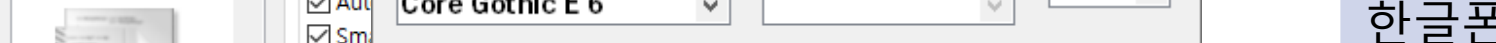
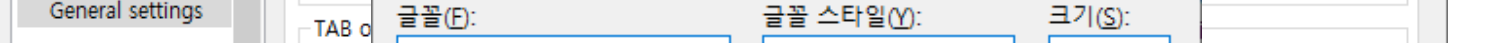
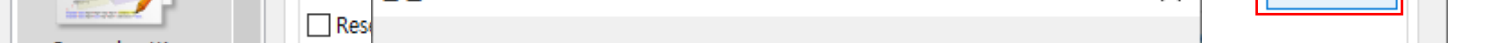
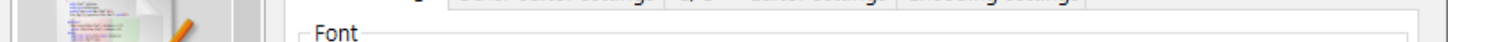
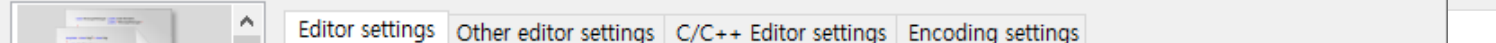
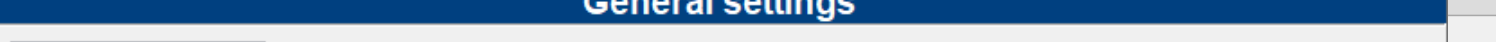
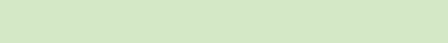
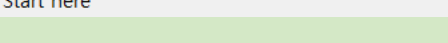
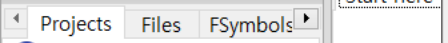
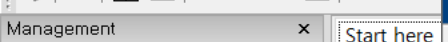
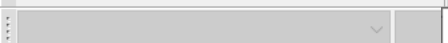
"Release" options

Output dir.: bin\Release\

Objects output dir.: obj\Release\

< Back Finish Cancel





Editor settings Other editor settings C/C++ Editor settings Encoding settings

Font

This 글꼴

☐ Res

TAB o

☐ Det

☐ Use

☒ TAB

TAB siz

Indent

☒ Aut

☒ Sm

☒ Bra

☒ Bac

☐ Sho

☒ Bra

☐ Sele

Select

☐ En

☐ En

☐ All

☐ B

색(C):

☐ 취소선(K)

☐ 밑줄(U)

스크립트(R):

영문폰트

한글폰트

슬래시

역슬래시

확인

취소

OK

Cancel

	슬래시	역슬래시
한글폰트	/	₩
영문폰트	/	\

Build & Run: F9

HelloWorld\bin\Debug\HelloWorld.exe

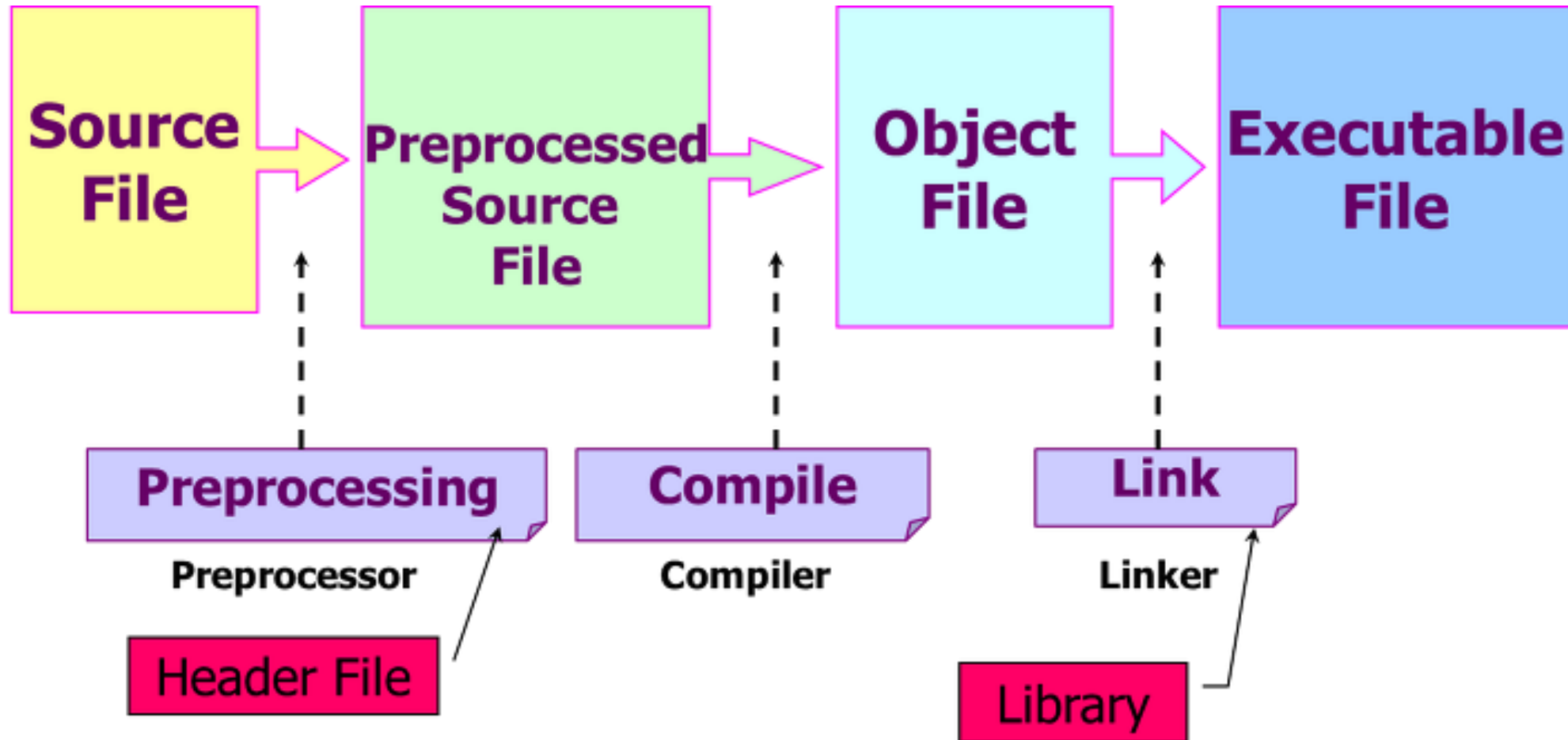
Hello world!

Process returned 0 (0x0) execution time : 0.408 s

Press any key to continue.

실행 파일 생성과정

■ Compile & Linking



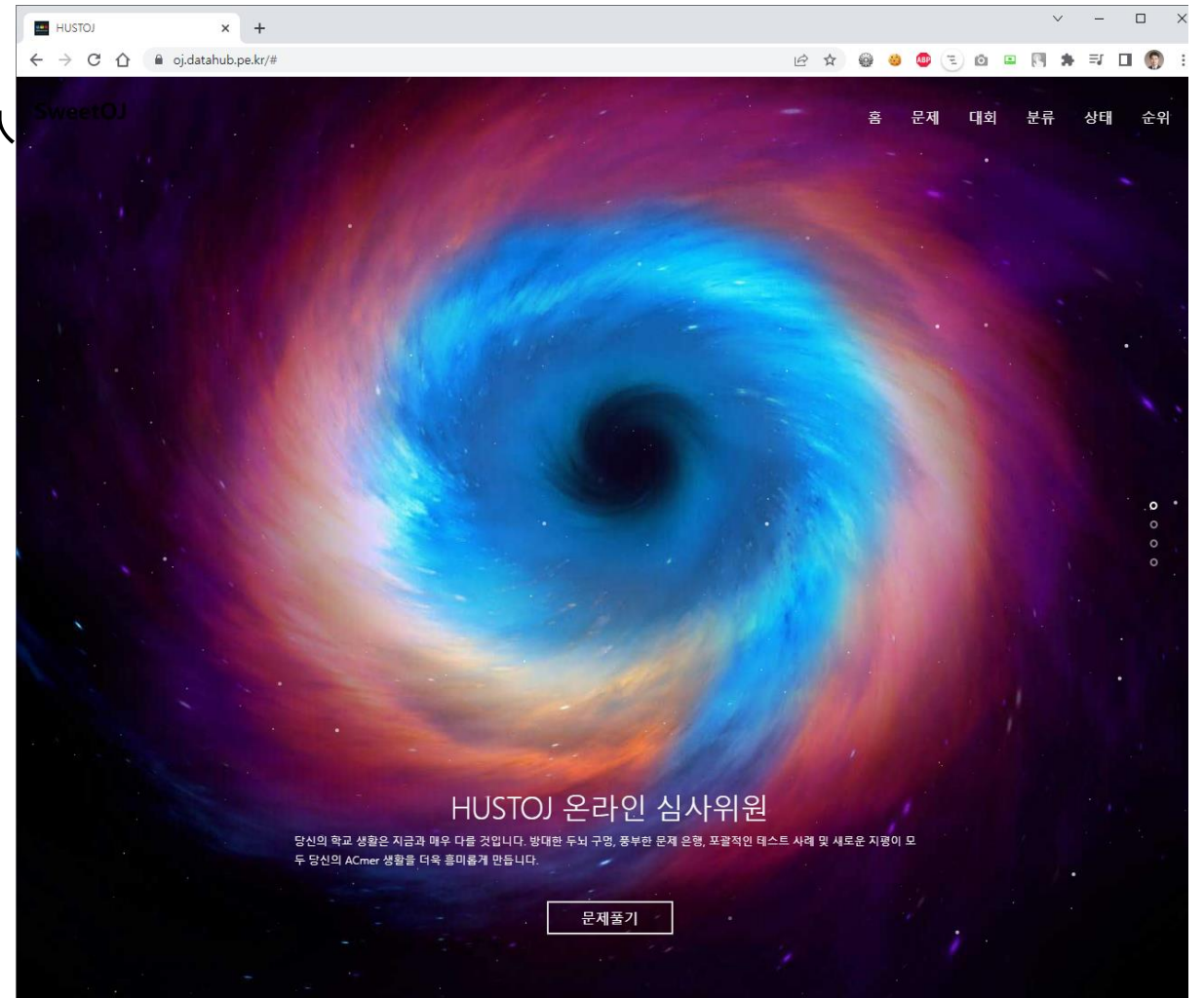
OJ 사용법

■ OJ(온라인 저지)란?

- 프로그래밍 대회에서 프로그램들을 사람과 사람이 아닌 컴퓨터로 풀 수 있는 대회 연습용으로 사용되기도 한다.

• 본 수업용 OJ

- <https://oj.datahub.pe.kr>



Online Judge 프로그램 접속 계정 정보

순번	소속학교	학년	이름	ID	PW
1	금천중학교	중1학년	구민성	cio01	ojpw01
2	제천중학교	중1학년	신현승	cio02	ojpw02
3	주성중학교	중1학년	이준원	cio03	ojpw03
4	충일중학교	중1학년	윤희찬	cio04	ojpw04
5	각리중학교	중2학년	이승민	cio05	ojpw05
6	각리중학교	중3학년	양태근	cio06	ojpw06
7	청주동중학교	중3학년	민채홍	cio07	ojpw07
8	청주동중학교	중3학년	신선호	cio08	ojpw08
9	한국교원대학교부설 미호중학교	중3학년	안승온	cio09	ojpw09
10	운호고등학교	고1학년	최정우	cio10	ojpw10

순번	소속학교	학년	이름	ID	PW
11	주성고등학교	고1학년	김단아	cio11	ojpw11
12	충주여자고등학교	고1학년	김가원	cio12	ojpw12
13	충주여자고등학교	고1학년	김시온	cio13	ojpw13
14	충주여자고등학교	고1학년	김유빈	cio14	ojpw14
15	충주여자고등학교	고1학년	원유리	cio15	ojpw15
16	충주여자고등학교	고1학년	이다연	cio16	ojpw16
17	한국교원대학교부설 고등학교	고1학년	조예원	cio17	ojpw17
18	흥덕고등학교	고1학년	남하준	cio18	ojpw18
19	흥덕고등학교	고1학년	이주원	cio19	ojpw20
20	청주IT과학고등학교	고2학년	이기준	cio20	ojpw20

OJ 사용법

■ 파일제출

- CodeBlock 등 IDE에서 프로그램 작성
- 완성된 프로그램을 OJ에 업로드 후
- 채점 결과 확인

• 채점 결과 종류

- 모두 맞춤
- 틀림 | 정확도: __%
- 실행중 에러 | 정확도: __%
- 컴파일 에러



C 언어 기초 문법

■ 함수란?

- 특정한 기능을 하는 코드들의 집합
- C언어에서는 _____() 형태

예시)

최소값을 구하는 함수: `min()`

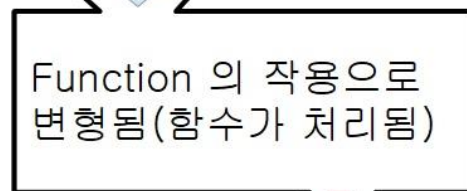
절대값을 구하는 함수: `abs()`

- `main()` 함수

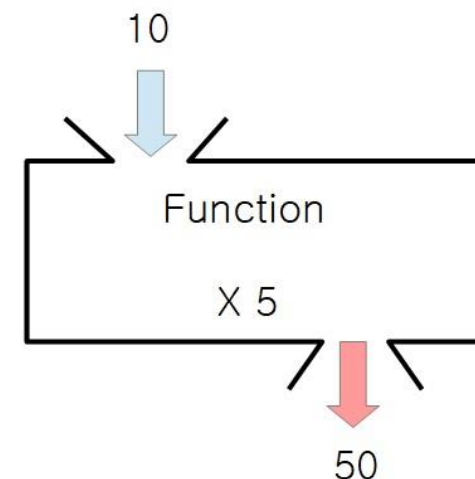
프로그램의 시작 지점

```
1 | #include <stdio.h>
2 |
3 | → int main()
4 | {
5 |     printf("Hello world! \n");
6 |     return 0;
7 | }
```

x 가 들어감,
변수 x 의 입력



f(x) 가 나옴
결과값 y 가 출력됨



■ C언어로 프로그램을 작성한다는

- 함수를 만들고, 만든 함수들의 실행 순서를 결정하는 것

C 언어 기초 문법

- `return 0;`

- 함수 결과값 0으로 함수 종료

- `printf("Hello world! \n");`

- `printf()` 함수: 표준 출력 장치에 출력하는 기능을 하는 함수
- 인수: "Hello world! \n" 를 `printf` 라는 함수에 전달

- `#include <stdio.h>`

- 표준 입출력 함수들에 대한 정보를 가지고 있는 `stdio.h` 라는 파일을 불러온다.

- 문장의 끝

- 함수 내에 존재하는 문장의 끝에는 세미콜론 문자 `;` 를 붙여준다.

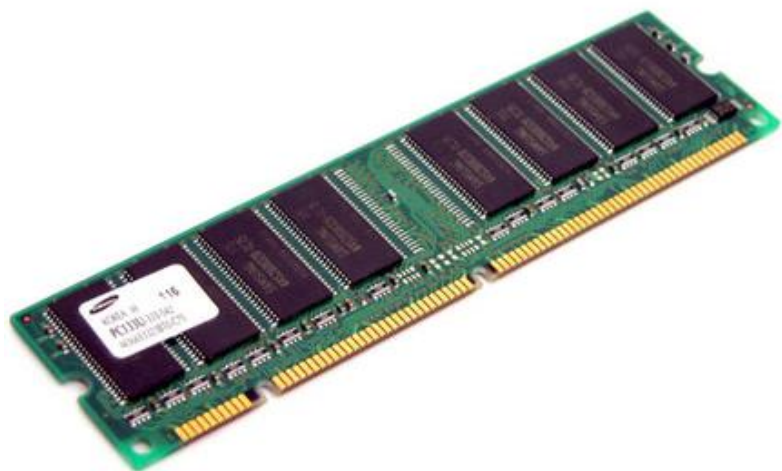
```
1 | #include <stdio.h>
2 |
3 | int main()
4 | {
5 |     printf("Hello world! \n");
6 |     retrun 0;
7 | }
```

자료의 저장

= 비트와 바이트 =

비트 (binary digit, bit)	바이트 (Byte)
0과 1, 두 가지 값만 가질 수 있는 측정 단위	여덟 개의 비트로 구성된 데이터의 양을 나타내는 단위
 0 OFF FALSE  1 ON TRUE	

■ 컴퓨터의 기억장소



= 데이터 단위 =



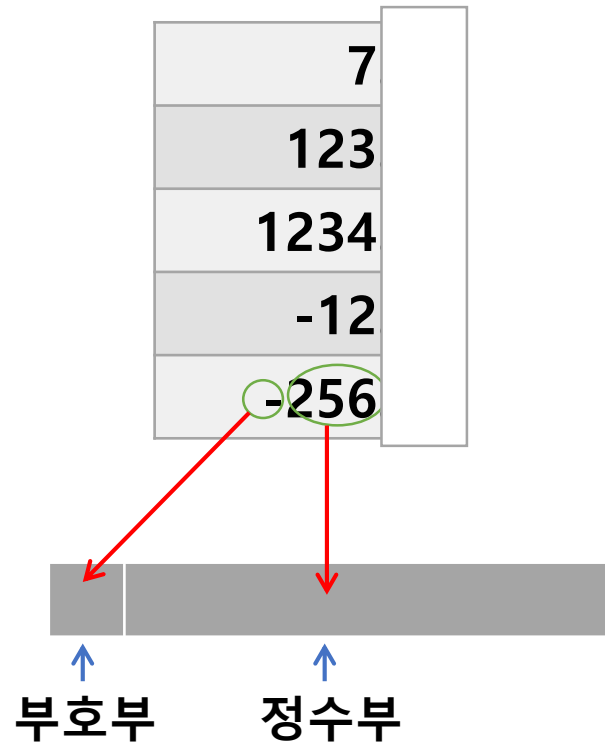
단위	정의	저장공간 크기
비트(bit)	0 또는 1	예 / 아니오
1바이트(Byte)	8 bit	알파벳과 숫자 한 개
1킬로바이트(KB)	1,024 Byte	몇 개의 문단
1메가바이트(MB)	1,024 KB	1분 길이의 MP3 노래
1기가바이트(GB)	1,024 MB	30분 길이의 HD 영화
1테라바이트(TB)	1,024 GB	약 200편의 FHD 영화

수의 표현 방식

■ 고정소수점 데이터형식

- (fixed-point data format)

- 정수 자료 저장에 사용



수의 표현 방식

- 정수 (양수, 음수)의 표현 - signed

양수

음수

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

1

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

2

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

3

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

4

:

0	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

127

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

-1

:

1	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

-124

1	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

-125

1	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

-126

1	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

-127

1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

-128

수의 표현 방식

■ 정수(양수)의 표현 - unsigned

0 0 0 0 0 0 0 0	0	1 1 1 1 1 1 1 1	255
0 0 0 0 0 0 0 1	1	:	
0 0 0 0 0 0 1 0	2	1 0 0 0 0 1 0 0	132
0 0 0 0 0 0 1 1	3	1 0 0 0 0 0 1 1	131
0 0 0 0 0 1 0 0	4	1 0 0 0 0 0 1 0	130
:		1 0 0 0 0 0 0 1	129
0 1 1 1 1 1 1 1	127	1 0 0 0 0 0 0 0	128

수의 표현 방식

■ 더 큰 수의 표현

0000000000	0000000000	0000000000	0000000000	0
0000000000	0000000000	0000000000	0000000001	1
0000000000	0000000000	0000000000	0000000010	2
0000000000	0000000000	0000000000	0000000011	3
0000000000	0000000000	0000000000	0000000100	4
0000000000	0000000000	0000000000	0000000101	5
0111111111	1111111111	1111111111	1111111111	$2^{31}-1$
1000000000	0000000000	0000000000	0000000000	-2^{31}

수의 표현 방식

■ 부동소수점 (浮動小數點) 데이터형식

- **floating-point** data format
- 지수에 의해 **소수점 이동**

ex) 12345.678

$$= 1234.5678 \times 10^1$$

$$= 123.45678 \times 10^2$$

$$= 12.345678 \times 10^3$$

$$= 1.2345678 \times 10^4$$

$$= 0.12345678 \times 10^5$$

수의 표현 방식

■ 부동소수점(浮動小數點) 데이터형식

- 2진수에서는...

ex) 10110.1001001001

$$\equiv \underline{1.01101001001001} \times 2^4$$



부호부 지수부 가수부

- 장점 : 아주 큰 수, 아주 작은 수 표현 가능
- 단점 : 계산하는데 시간이 오래 걸림

C언어의 자료형(data type)

■ 정수형

• char 

(문자형) - 숫자도 저장하지만 주로 문자를 저장함

character

• short 

short int

• int 

integer

• long 

long integer

• long long 

64bit long 

C언어의 자료형(data type)

■ 실수형

• float

- 단정도 정밀도 실수 (소수점 이하 6자리까지)
- 4byte(32bit)



• double

- 배정도 정밀도 실수 (소수점 이하 15자리까지)
- 8byte(64bit)



C언어의 자료형(data type)

용도	타입	크기		signed(부호있음)	unsigned(부호없음)
정수형 (문자형)	char	1B	8bit	$-2^7 \sim 2^7-1$ (127)	$0 \sim 2^8-1$ (256)
정수형	short	2B	16bit	$-2^{15} \sim 2^{15}-1$ (≒3.2만)	$0 \sim 2^{16}-1$ (≒6.5만)
	int	4B	32bit	$-2^{31} \sim 2^{31}-1$ (≒21억)	$0 \sim 2^{32}-1$ (≒42억)
	long	4B	32bit	$-2^{31} \sim 2^{31}-1$ (≒21억)	$0 \sim 2^{32}-1$ (≒42억)
	long long	8B	64bit	$-2^{63} \sim 2^{63}-1$ (≒922경)	$0 \sim 2^{64}-1$ (≒1844경)
실수형	float	4B	32bit	$3.4 \times 10^{-38} \sim 3.4 \times 10^{38}$	
	double	8B	64bit	$1.7 \times 10^{-308} \sim 1.7 \times 10^{308}$	

변수 (variable)

■ 변수의 의미

- '변하는 수' 라는 의미
- 컴퓨터에게 무언가를 기억시키기 위해 사용
- 자료를 저장하는 공간에 이름은 붙인 것
- 프로그래머가 이름(변수명)을 결정

■ 변수의 사용

- 선언을 한 뒤부터 사용 가능
- = 연산자로 값을 할당
- 선언과 동시에 할당 가능(초기화)
- 초기화 하지 않으면 쓰레기 값이 들어있음.

```
#include <stdio.h>
```

```
int main() {
```

```
    int age;    // 변수의 선언
```

```
    age = 20;   // 변수에 할당
```

```
    int height, weight;
```

```
    char blood = 'A';    // 초기화
```

```
    double pi = 3.141592;
```

```
    printf("age: %d \n", age);
```

```
    printf("height: %d \n", height);
```

```
    printf("blood: %c \n", blood);
```

```
    printf("pi: %1f \n", pi);
```

```
    printf("pi: %.21f \n", pi);
```

```
    return 0;
```

```
}
```

printf 함수의 기본적인 이해

- 첫 번째 인수로 전달된 문자열의 서식에 맞게 출력

```
#include <stdio.h>
```

```
int main() {
```

```
    int age = 20;
```

```
    int year = 2010, month = 10, day = 31;
```

```
    printf("my age: %d\n", age);
```

```
    printf("my birthday: %d/%d/%d\n", year, month, day);
```

```
        //      %d : decimal(10진)
```

```
    printf("Good\nmorning\neveryday\n");
```

```
        //      \n : new line
```

```
}
```

```
my age: 20
my birthday: 2010/10/31
Good
morning
everyday
```

%d 의 의미:
decimal(10진)
10진수로
출력하라는 의미

서식문자

■ 서식 문자의 종류와 그 의미

서식문자	출력 형태
%c	단일 문자
%d	부호 있는 10진 정수
%i	부호 있는 10진 정수, %d와 같음
%f, %lf	부호 있는 10진 실수
%s	문자열
%o	부호 없는 8진 정수
%u	부호 없는 10진 정수
%x	부호 없는 16진 정수, 소문자 사용
%X	부호 없는 16진 정수, 대문자 사용
%e	e 표기법에 의한 실수
%lld, %llu	부호 있는, 부호 없는 long long 정수
%g	값에 따라서 %f, %e 둘 중 하나를 선택
%G	값에 따라서 %f, %G 둘 중 하나를 선택
%%	% 기호 출력

```
#include <stdio.h>
```

```
int main() {
    char blood = 'A';
    int age = 20;
    double pi=3.1415926535;
```

```
printf("blood: %c. \n", blood);
printf("name : %s. \n", "Reinhard");
printf("age  : %d. \n", age);
printf("pi   : %lf. \n\n", pi);
```

```
printf("blood: %10c. \n", blood);
printf("name : %10s. \n", "Reinhard");
printf("age  : %10d. \n", age);
printf("pi   : %10.4lf. \n\n", pi);
```

```
printf("blood: %-10c. \n", blood);
printf("name : %-10s. \n", "Reinhard");
printf("age  : %-10d. \n", age);
printf("pi   : %-10.4lf. \n\n", pi);
```

```
}
```

[기본서식]

```
blood: A.
name : Reinhard.
age  : 20.
pi   : 3.141593.
```

[오른쪽 정렬]

```
blood:          A.
name :      Reinhard.
age  :          20.
pi   :      3.1416.
```

[왼쪽 정렬]

```
blood: A
name : Reinhard
age  : 20
pi   : 3.1416
```

상수(constant)

■ 상수의 의미

- 항상 똑같은 수 - 변하지 않는다는 뜻
- 값이 바뀌면 안되는 경우에 사용
- 변수명과 이름 짓는 방법이 동일
- 보통 모두 대문자 이름을 사용

■ 심볼릭 상수

- const 키워드를 사용
- 정의와 함께 초기화

■ 매크로 상수

- #define 문 사용

매크로
상수

심볼릭
상수

```
#include <stdio.h>
#define LENGTH 50

int main(void)
{
    const int MAX = 100;
    const double PI = 3.1415926535;
    MAX = 125; // 에러 발생

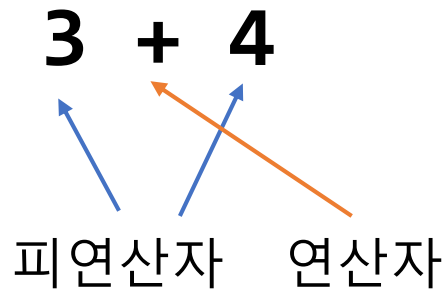
    printf("PI: %lf \n", PI);
    printf("length: %d \n", LENGTH);
}
```

상수는
대문자로
표기하는
전통이 있다.

전처리
과정에서
50으로
변경됨

연산자(operator)

■ 연산자와 피연산자



■ 이항연산자

- 피연산자가 두 개인 연산자
- +, -, x, ÷

■ 단항연산자

- 피연산자가 한 개인 연산자
- -, +

■ C언어의 연산자 표기

의미	수학	C언어	비고
덧셈	+	+	
뺄셈	-	-	
곱셈	×	*	✓
나눗셈	÷	/	✓
나머지		%	
같다	=	==	✓
다르다		!=	✓
크다	>, ≥	>, >=	
작다	<, ≤	<, <=	
그리고	And	&&	
또는	Or		

연산자(operator)

■ 대입 연산자와 산술 연산자

연산자	설명	결합방향
=	연산자 오른쪽에 있는 값을 연산자 왼쪽에 있는 변수에 대입한다. 예) num = 20;	←
+	두 피연산자의 값을 더한다. 예) num = 4+3;	→
-	왼쪽 피연산자의 값에서 오른쪽 피연산자 값을 뺀다. 예) num = 4-3;	→
*	두 피연산자의 값을 곱한다. 예) num = 4*3;	→
/	왼쪽의 피연산자 값을 오른쪽 피연산자 값으로 나눈 몫을 구한다. 예) num = 7/3;	→
%	왼쪽의 피연산자 값을 오른쪽 피연산자 값으로 나눈 나머지를 구한다. 예) num = 7%3;	→

연산자(operator)

■ 대입 연산자와 산술 연산자 실습

```
#include <stdio.h>

int main() {
    int n1 = 9, n2 = 2, res;
    res = n1 + n2;
    printf("%d + %d = %d \n", n1, n2, res);
    res = n1 - n2;
    printf("%d - %d = %d \n", n1, n2, res);

    printf("%d * %d = %d \n", n1, n2, n1*n2);
    printf("%d / %d = %d \n", n1, n2, n1/n2);
    printf("%d %% %d = %d\n", n1, n2, n1%n2);
}
```

D:\MyProjects\Algorithm\bin\Deb

9	+	2	=	11
9	-	2	=	7
9	*	2	=	18
9	/	2	=	4
9	%	2	=	1

연산자(operator)

■ 복합 대입 연산자

- 대입 연산자와 산술 연산자가 합해져서 다양한 형태의 대입 연산자 정의

<code>a = a + b;</code>	← 같은 의미 →	<code>a += b;</code>
<code>a = a - b;</code>	← 같은 의미 →	<code>a -= b;</code>
<code>a = a * b;</code>	← 같은 의미 →	<code>a *= b;</code>
<code>a = a / b;</code>	← 같은 의미 →	<code>a /= b;</code>
<code>a = a % b;</code>	← 같은 의미 →	<code>a %= b;</code>

```
int main() {  
    int a, b;  
  
    a = 6, b = 2;  
    printf("[before] a:%d, b:%d\n", a, b);  
    a = a + b;  
    printf("[ a=a+b] a:%d, b:%d\n", a, b);  
  
    a = 6, b = 2;  
    printf("[before] a:%d, b:%d\n", a, b);  
    a += b;  
    printf("[a += b] a:%d, b:%d\n", a, b);  
}
```

연산자(operator)

■ 부호 연산으로서 +, - 연산자

- 양수, 음수 표현에 사용

예) int position = -100;

■ 증감 연산자

연산자	연산의 예	의미	결합성
++a	printf("%d", ++a)	선 증가, 후 연산	←
a++	printf("%d", a++)	선 연산, 후 증가	←
--b	printf("%d", --b)	선 감소, 후 연산	←
b--	printf("%d", b--)	선 연산, 후 감소	←

```
#include <stdio.h>

int main(void)
{
    int a=10;
    a++;
    printf("%d \n", a); // 11
    ++a;
    printf("%d \n", a); // 12

    int b=10;
    b--;
    printf("%d \n", b); // 9
    --b;
    printf("%d \n", b); // 8
    return 0;
}
```

연산자(operator)

■ 연산자의 우선순위와 결합방향

우선순위	연산자유형		연산자종류	결합방향
높음 ↑ ↓ 낮음	식, 구조체, 공용체 연산자		() [] -> .	좌 → 우
	단항 연산자		! ~ ++ -- - (형명칭) * & sizeof	좌 ← 우
	이항 연산자	승, 제	* / %	좌 → 우
		가, 감	+ -	좌 → 우
		비트 이동	<< >>	좌 → 우
		대소 비교	< <= > >=	좌 → 우
		등가 판정	== !=	좌 → 우
		비트 AND	&	좌 → 우
		비트 XOR	^	좌 → 우
		비트 OR		좌 → 우
		논리 AND	&&	좌 → 우
		논리 OR		좌 → 우
	조건 연산자		? :	좌 ← 우
	대입 연산자		+ =+ =+ *= /= %=	좌 ← 우
	나열 연산자		,	좌 → 우

■ 꼭 기억해야할 연산자 우선순위

- ① ()
- ② ++ -- !
- ③ * / %
- ④ + -
- ⑤ =

연산자(operator)

■ 연산자 우선순위 실험

```
#include <stdio.h>

int main(void) {
    int a = 10+4*3-1;
    printf("%d \n", a);

    int b = 10+4*(3-1);
    printf("%d \n", b);

    int r=4+5*6/(2+1)+15-5*2;
    printf("%d \n", r);
}
```

21
18
19

■ 연산자 결합방향 실험

```
#include <stdio.h>

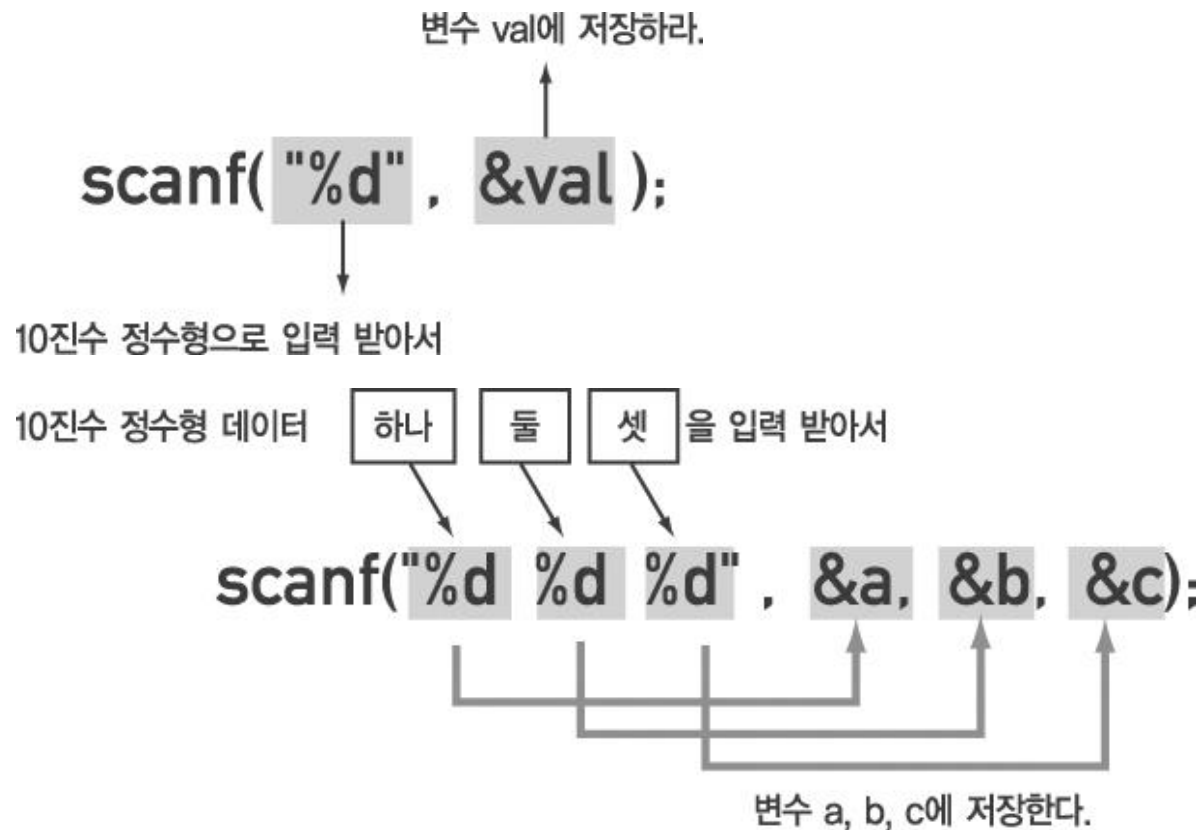
int main(void)
{
    int r = 10-1-2-3+4+5;
    printf("%d \n", r);

    int a=3, b=4, c=5, d=6;
    a = b = c = d;
    printf("%d %d %d %d", a,b,c,d);
}
```

13
6 6 6 6

scanf() 함수를 이용한 입력

- 키보드로부터 정수 입력을 위한 scanf 함수의 호출



```
#include <stdio.h>
```

OJ에 제출

```
int main(void)
{
```

```
    int n1, n2;
```

```
    printf("input n1: ");
```

```
    scanf("%d", &n1);
```

```
    printf("input n2: ");
```

```
    scanf("%d", &n2);
```

```
    printf("%d + %d = %d\n", n1, n2, n1+n2);
```

```
    printf("input two nums:\n");
```

```
    scanf("%d %d", &n1, &n2);
```

```
    printf("%d * %d = %d\n", n1, n2, n1*n2);
```

```
    return 0;
```

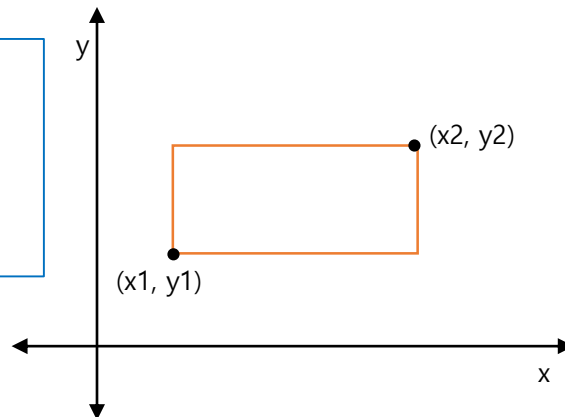
```
}
```

연습문제 – 직사각형의 넓이

■ 문제

- 두 점의 x , y 좌표를 입력 받아서, 두 점이 이루는 직사각형의 넓이를 계산하여 출력하는 프로그램을 작성하시오. (x_1, y_1) 의 좌표가 (x_2, y_2) 보다 작다고 가정
- 실행의 예

```
2 4
4 8
8
```



■ 정답

OJ에 제출

```
#include <stdio.h>

int main(void) {
    int x1, y1;
    int x2, y2;

    return 0;
}
```


제어문

■ 조건문

- if 문
 - 단순 if
 - if ... else
 - if ... else if ... else
- switch문
 - case
 - default

■ 반복문

- while 문
- do...while 문
- for문

■ 기타

- break 문
- continue 문

조건문 – if

■ 단순 if 문

- 특정 조건을 만족하는 경우 해야 할 일이 있을 때 사용
- 처리해야 할 명령어가 2개 이상인 경우 반드시 **중괄호**로 묶어야 함

```
input score: 100
Perfect!
You good!
Test passed.
```

```
input score: 60
You good!
Test passed.
```

```
#include <stdio.h>

int main() {
    int score;
    printf("input score: ");
    scanf("%d", &score);

    if(score == 100)
        printf("Perfect!\n");

    if(score >= 60) {
        printf("You good!\n");
        printf("Test passed.\n");
    }
    return 0;
}
```

연산자(operator)

■ 비교 연산자(관계 연산자)

- 두 피연산자의 관계(크다, 작다 혹은 같다)를 따지는 연산자
- true(논리적 참, 1), false(논리적 거짓, 0) 반환

연산자	연산의 예	의미	결합성
==	a == b	a와 b가 같은가	→
!=	a != b	a와 b가 같지 않은가	→
<	a < b	a가 b보다 작은가	→
>	a > b	a가 b보다 큰가	→
<=	a <= b	a가 b보다 작거나 같은가	→
>=	a >= b	a가 b보다 크거나 같은가	→

연산자(operator)

■ 비교 연산자(관계 연산자)

```
#include <stdio.h>
```

```
int main() {
```

```
    int a=6, b=2;
```

```
    printf("%d==%d : %d\n", a, b, a==b);
```

```
    printf("%d!=%d : %d\n", a, b, a!=b);
```

```
    printf("%d<%d : %d\n", a, b, a<b);
```

```
    printf("%d>%d : %d\n", a, b, a>b);
```

```
    printf("%d<=%d : %d\n", a, b, a<=b);
```

```
    printf("%d>=%d : %d\n", a, b, a>=b);
```

```
}
```

- 모든 연산자는 계산 결과를 반환한다.
- 비교연산자는 참(true) 이면 1을, 거짓(false)이면 0을 반환한다.

```
D:\MyProjects\Algorithm\bin\Debug\A
6==2 : 0
6!=2 : 1
6<2 : 0
6>2 : 1
6<=2 : 0
6>=2 : 1
```

조건문 – if

■ if ... else 문

- 특정 조건을 만족하는 경우 A를 하고 만족하지 않는 경우 B를 수행할 때 사용
- 점수를 입력 받아 60점 이상이면 'You passed!' 를 아니면 'Failed.' 'Retry it.' 을 출력하는 프로그램을 작성하시오.

```
#include <stdio.h>

int main()
{
    int score;
    printf("input score: ");
    scanf("%d", &score);

    if(score >= 60)
        printf("You passed!\n");
    else {
        printf("Failed.\n");
        printf("Retry it.\n");
    }
    return 0;
}
```

연산자(operator)

■ 논리 연산자

- and, or, not을 표현하는 연산자
- true(1), false(0) 반환

연산	C연산자	연산의 예	의미	결합성
AND	&&	a && b	true면 true 리턴	→
OR		a b	하나라도 true면 true 리턴	→
NOT	!	!a	true면 false를, false면 true 리턴	→

조건문 – if

- 필기/실기 모두 60점 이상이어야 합격

```
#include <stdio.h>

int main() {
    int pilgi, silgi;
    printf("필기와 실기 점수를 입력: ");
    scanf("%d %d", &pilgi, &silgi);

    if(pilgi>=60 && silgi>=60)
        printf("You passed!\n");
    else {
        printf("Failed.\n");
        printf("Retry it.\n");
    }
    return 0;
}
```

- 필기/실기 둘 중 하나만 60점 이상이면 합격

```
#include <stdio.h>

int main() {
    int pilgi, silgi;
    printf("필기와 실기 점수를 입력: ");
    scanf("%d %d", &pilgi, &silgi);

    if(pilgi>=60 || silgi>=60)
        printf("You passed!\n");
    else {
        printf("Failed.\n");
        printf("Retry it.\n");
    }
    return 0;
}
```

조건문 – if

- 점수가 60점 미만이면 합격

```
#include <stdio.h>

int main() {
    int score;
    printf("input score: ");
    scanf("%d", &score);

    if( !(score<60) )
        printf("You passed!\n");
    else
        printf("Failed.\n");

    return 0;
}
```

- 필기가 60점미만이 아니고, 실기도 60점 미만이면 합격

```
#include <stdio.h>

int main(){
    int pilgi, silgi;
    printf("필기와 실기 점수를 입력: ");
    scanf("%d %d", &pilgi, &silgi);

    if( !(pilgi<60) && !(silgi<60))
        printf("You passed!\n");
    else
        printf("Failed.\n");

    return 0;
}
```


조건문 – if

OJ에 제출

■ if ... else if ... else 문

- 점수를 입력 받아 90점 이상이면 'A', 80점 이상이면 'B', 70점 이상이면 'C', 그 밖의 경우 'F'를 출력하는 프로그램을 작성하시오.

```
#include <stdio.h>

int main() {
    int score;
    char grade;
    printf("input score: ");
    scanf("%d", &score);

    if(score >= 90)
        grade = 'A';
    else if(score >= 80)
        grade = 'B';
    else if(score >= 70)
        grade = 'C';
    else
        grade = 'F';

    printf("grade: %c\n", grade);
    return 0;
}
```

연습문제

OJ에 제출

■ BMI 계산

- 체질량지수는 자신의 몸무게(kg)를 키의 제곱(m)으로 나눈 값입니다.
- 몸무게(kg단위)와 키(cm단위)를 입력받아 BMI를 계산하여 소수점 둘째 자리까지 출력하고,
- BMI 수치에 따른 결과를 출력하시오.
 - 18.5 미만이면 '저체중'
 - 18.5 ~ 23미만이면 '정상'
 - 23.0 ~ 25 미만이면 '과체중'
 - 25.0 이상부터는 '비만'

```
#include <stdio.h>
int main() {
    double w; // 몸무게
    double h; // 키
    double bmi;

    scanf("%lf", &w);
    scanf("%lf", &h);

}
```

조건문 – switch

■ switch 문

- 비교·선택 할 조건이 많은 경우 유용
- switch(수식)
 - 수식은 **정수형** 변수, 정수형 수식만 가능
- case 값
 - 값은 정수만 가능
- default: 기본
- break: switch 탈출

```
#include <stdio.h>
int main() {
    int score;
    char grade;
    printf("input score: ");
    scanf("%d", &score);

    switch(score / 10) {
        case 10:
        case 9:
            grade = 'A';
            break;
        case 8:
            grade = 'B';
            break;
        case 7:
            grade = 'C';
            break;
        default:
            grade = 'F';
    }
    printf("grade: %c\n", grade);
    return 0;
}
```

score / 10
[정수] 나누기
[정수]는
결과도 정수

연습문제

- 두 개의 정수와 한 개의 사칙 연산자를 입력받아 사칙연산 결과를 처리하는 프로그램을 작성하시오.
- 입력되는 모든 숫자는 정수이고, 가운데 연산자는 + - * / % 이외는 없다.

```
예: 10 + 5
계산할 수식을 입력하세요
9 * 5
9 * 5 = 45
```

```
#include <stdio.h>

int main() {
    int n1, n2, res;
    char op;
    printf("예: 10 + 5\n");
    printf("계산할 수식을 입력하세요\n");
    scanf("%d %c %d", &n1, &op, &n2);

    printf("%d %c %d = %d\n", n1, op, n2, res);
    return 0;
}
```

반복문

- 반복문의 기능

- 특정 영역을 특정 조건이 만족되는 동안에 반복 실행하기 위한 문장

- 세 가지 형태의 반복문이 제공됨

- 1) while문에 의한 반복

- 몇 번 반복해야 하는지 모를 때 사용, ex) 답 맞출 때까지 계속

- 2) do ~ while문에 의한 반복

- 일단 한번은 실행하고 그 결과에 따라 다시 반복할 수도 있을 때 사용, ex) 메뉴 입력

- 3) for문에 의한 반복

- 반복 횟수가 정해져 있는 경우 주로 사용, ex) 10번 출력

반복문 - while

■ 형식

```
while(반복조건)  
    반복할 문장;
```

```
while(반복조건) {  
    반복할 문장1;  
    반복할 문장2;  
    :  
}
```

- 반복조건이 참인 동안 반복할 문장을 실행

■ 예시

```
int n = 1;  
while(n < 5) {  
    printf("%d \n", n);  
    n++; // n 1증가  
}
```

n

5

1
2
3
4

반복문 - while

■ 5회 반복 방법1

```
int c = 0;
while(c < 5) {
    printf("%d \n", c);
    c++;
}
```

0
1
2
3
4

■ 5회 반복 방법2

```
int c = 1;
while(c <= 5) {
    printf("%d \n", c);
    c++;
}
```

1
2
3
4
5

반복문 - while

■ 퀴즈

```
#include <stdio.h>
int main() {
    int num = 10; // 10부터 시작

    puts("Rocket lunch countdown..");
    while(num > 0) { // 0보다 크면
        printf(" %2d \n", num);
        num--; // 1씩 감소하면서...
    }
    printf("last num:%2d \n", num);
    return 0;
}
```

■ 결과

```
Rocket lunch countdown..
10
 9
 8
 7
 6
 5
 4
 3
 2
 1
```

- 카운트 다운 숫자는 얼마까지 출력될까?
- 종료 직전 num 값은? _____

0

반복문 - while

- 1부터 10까지 누계 구하기

$$1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10$$

sum	0	1	3	6	10	15	21	28	36	45	55
c	1	2	3	4	5	6	7	8	9	10	11

Diagram illustrating the execution of a while loop for calculating the sum of numbers from 1 to 10. The table shows the state of variables `sum` and `c` at each iteration. The loop continues until `c` reaches 11, at which point the condition `c < 11` becomes false. The final sum is 55.

[초기값]

sum = 0
c = 1

while(c < 11)

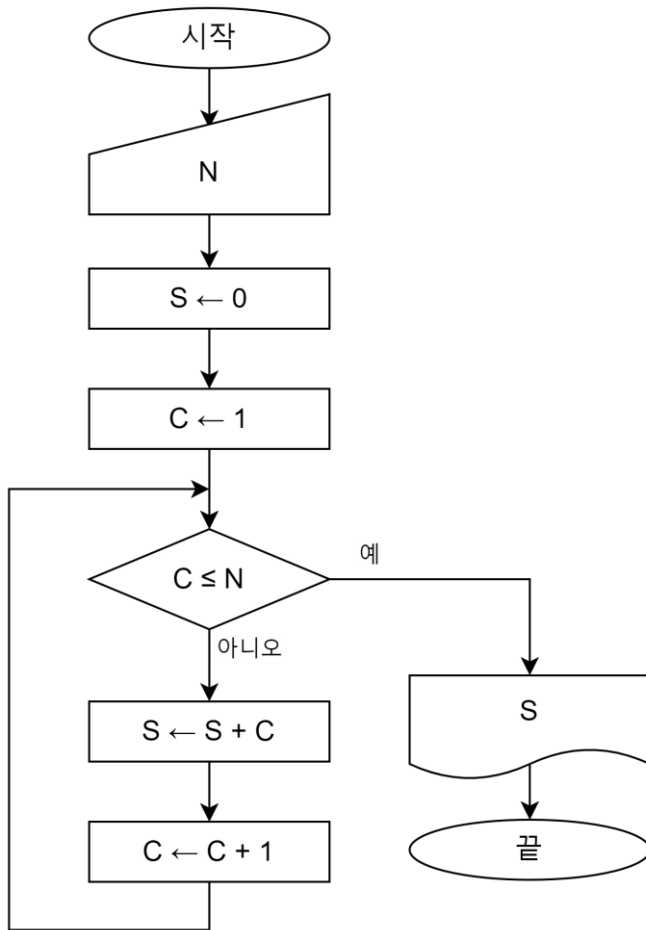
sum = sum + c
c = c + 1

while(c <= 10)

sum = sum + c
c = c + 1

반복문 - while

■ 1부터 10까지 누계 구하기



■ 소스코드

```
#include <stdio.h>
int main() {
    int sum=0, c=1;
    while(c ? 10) {
        sum=sum+c;
        c++;
    }
    printf("%d \n", sum);
}
```

■ 출력

55

STEP	sum	c
1	0	1
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		

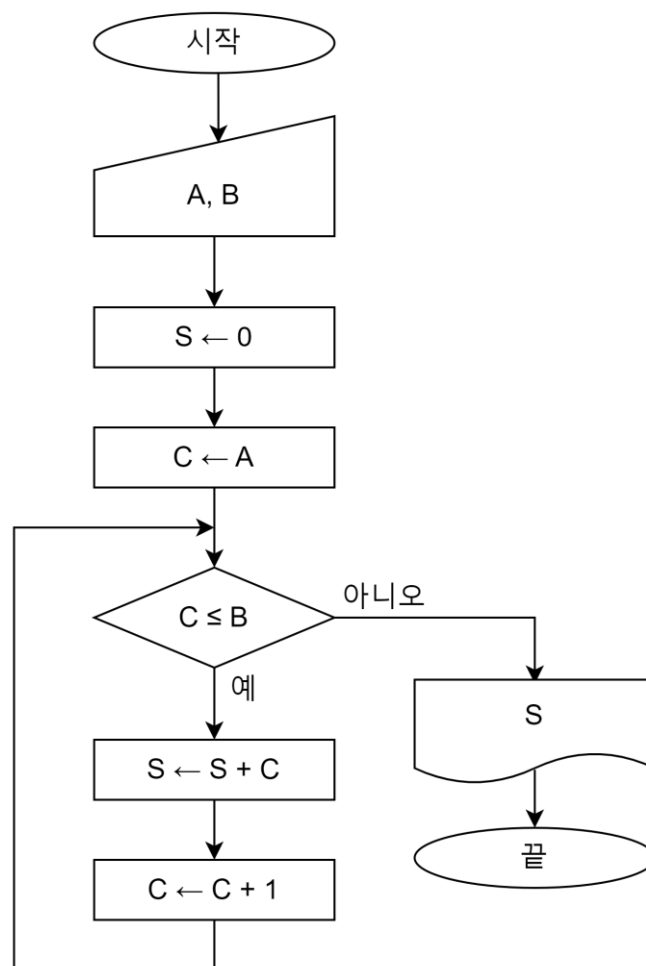
연습문제

- while문을 사용하여 두 정수 a와 b를 입력 받아 a부터 b까지 누계를 구하는 프로그램을 작성하시오.

(a < b 라고 가정)

- 예

1 10
55



```
#include <stdio.h>
```

```
int main() {  
    int a, b;
```

```
    return 0;
```

```
}
```

반복문 - while

■ 중첩된 while

```
while(반복조건1) {  
    while(반복조건2) {  
  
    }  
}
```

- 들여쓰기를 사용하여 반복 내용의 시작과 끝을 명확히 하자!

■ 00:00 부터 ~ 05:59 까지 시간 출력

```
#include <stdio.h>  
  
int main() {  
    int hour=0, min=0;  
  
    puts("clock time");  
    while(hour < 6) {  
        min = 0;  
        while(min < 60) {  
            printf("%02d:%02d ", hour, min);  
            min++;  
        }  
        printf("\n");  
        hour++;  
    }  
    return 0;  
}
```

반복문 – do ... while

■ 문법

```
do {  
    반복할 문장1;  
    반복할 문장2;  
    :  
}  
while(반복조건);
```

- while문은 조건을 먼저 확인하고 반복 할지 결정하고,
- do...while문은 일단 한 번 해보고 반복 할지 결정한다.

■ 비교

```
int main() {  
    int n = 1;  
    while(n < 1) {  
        printf("%d \n", n);  
        n++;  
    }  
}
```

```
int main() {  
    int n = 1;  
    do {  
        printf("%d \n", n);  
        n++;  
    }  
    while(n < 1);  
}
```

반복문 – do ... while

■ 사용 예

```
#include <stdio.h>
int main()
{
    int n;
    do {
        printf("\n[MENU]\n");
        printf("1. New Game\n");
        printf("2. Load Game\n");
        printf("3. Save Game\n");
        printf("4. Quit\n");
        printf("what's your choice: ");
        scanf("%d", &n);
    }
    while(n != 4);
}
```

```
[MENU]
1. New Game
2. Load Game
3. Save Game
4. Quit
what's your choice: 1
```

```
[MENU]
1. New Game
2. Load Game
3. Save Game
4. Quit
what's your choice: 3
```

```
[MENU]
1. New Game
2. Load Game
3. Save Game
4. Quit
what's your choice: 4
```

```
Process returned 0 (0x0)
```

반복문 - for

■ for문 형식

```
for(①초기식; ②조건식; ④증감식) {  
    ③반복할 문장;  
}
```

■ 예시

```
for(int i=0; i<3; i++) {  
    printf("%d\n", i);  
}
```

■ 실행순서

- 1) ①초기식 ②조건식 ③반복할 문장 ④증감식
- 2) ②조건식 ③반복할 문장 ④증감식
- 3) :
- 4) ②조건식

■ 실행순서

①초기식	②조건식	③반복문장	④증감식	i
i=0	i<3	printf(0)	i++	1
	i<3	printf(1)	i++	2
	i<3	printf(2)	i++	3
	i<3			

반복문 - for

■ 5회 반복 방법1

```
for(int i=0; i<5; i++) {  
    printf("%d\n", i);  
}
```

■ 출력

```
0  
1  
2  
3  
4
```

■ 5회 반복 방법2

```
for(int i=1; i<=5; i++) {  
    printf("%d\n", i);  
}
```

■ 출력

```
1  
2  
3  
4  
5
```


반복문 - for

- 1부터 15사이 3의 배수 출력

```
#include <stdio.h>
int main() {
    for(int i=3; i<=15; i=i+3)
        printf("%d", i);
}
```

- 출력

```
3
6
9
12
15
```

- 10부터 1까지 카운트다운

```
#include <stdio.h>
int main() {
    puts("Rocket launch countdown...");
    for(int i=10; i>=1; i--)
        printf("%d ", i);
}
```

- 출력

```
Rocket launch countdown...
10 9 8 7 6 5 4 3 2 1
```

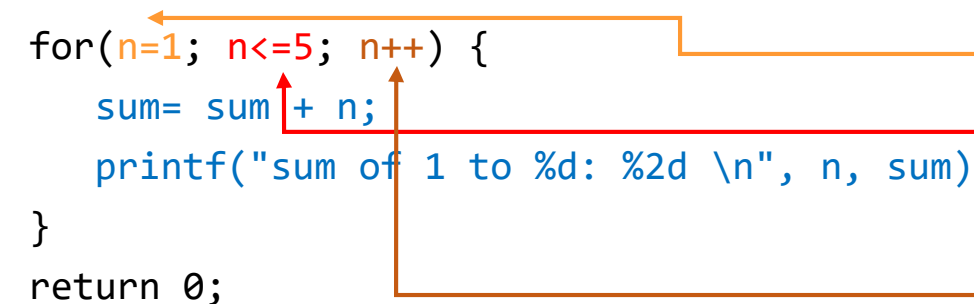
반복문 – for 문과 while 문 비교

■ for 문

```
// 1부터 5까지의 합계를 구하는 프로그램
#include <stdio.h>

int main() {
    int sum = 0;
    int n;

    for(n=1; n<=5; n++) {
        sum= sum + n;
        printf("sum of 1 to %d: %2d \n", n, sum);
    }
    return 0;
}
```

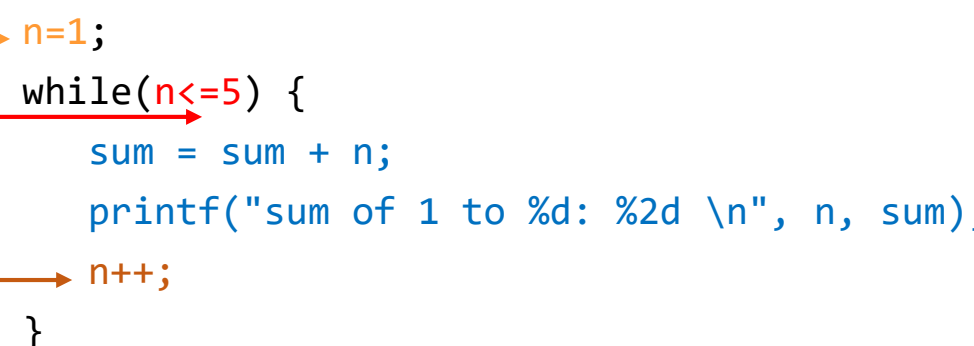


■ while 문

```
// 1부터 5까지의 합계를 구하는 프로그램
#include <stdio.h>

int main() {
    int sum = 0;
    int n;

    n=1;
    while(n<=5) {
        sum = sum + n;
        printf("sum of 1 to %d: %2d \n", n, sum);
        n++;
    }
    return 0;
}
```



3의 배수 게임

■ 문제

3의 배수 게임을 하던 정올이는 3의 배수 게임에서 작은 실수를 계속해서 벌칙을 받게 되었다.

3의 배수 게임의 왕이 되기 위한 수련 프로그램을 작성해 보자.

** 3의 배수 게임이란?

여러 사람이 순서를 정해 순서대로 수를 부르는 게임이다.

만약 3의 배수를 불러야 하는 상황이라면, 그 수 대신 "박수"를 친다.

■ 입력

첫 줄에 하나의 정수 n 이 입력된다.
(n 은 50미만의 자연수이다)

■ 출력

1부터 n 까지 순서대로 공백을 두고 수를 출력하는데, 3의 배수(3, 6, 9 ...)인 경우 수 대신 영문 대문자 X 를 출력한다.

■ 입력과 출력의 예

입력 예	출력 예
7	1 2 X 4 5 X 7

약수의 합 구하기

■ 문제

한 정수 n 을 입력 받아서 n 의 모든 약수의 합을 구하는 프로그램을 작성하시오.

예를 들어 10의 약수는 1, 2, 5, 10이므로 이 값들의 합인 18이 10의 약수의 합이 된다.

■ 입력

첫번째 줄에 정수 n 이 입력된다.
(단, $1 \leq n \leq 100,000$)

■ 출력

n 의 약수의 합을 출력한다

■ 입력과 출력의 예

입력 예	출력 예
5	6

입력 예	출력 예
10	18

■ 고찰

n 의 약수들을 어떻게 알아낼 수 있을까?

약수의 합 구하기

■ 문제

한 정수 n 을 입력 받아서 n 의 모든 약수의 합을 구하는 프로그램을 작성하시오.

예를 들어 10의 약수는 1, 2, 5, 10이므로 이 값들의 합인 18이 10의 약수의 합이 된다.

■ 입력

첫번째 줄에 정수 n 이 입력된다.
(단, $1 \leq n \leq 100,000$)

■ 출력

n 의 약수의 합을 출력한다

■ 답안 예시

```
#include <stdio.h>

int main() {
    int n;
    scanf("%d", &n);

    int ans = 0;

    printf("%d\n", ans);
    return 0;
}
```

반복문 - 중첩된 for

1) 한 학급 1번 부터 30번까지 출력한다.

```
int main() {  
    for(int n=1; n<=30; n++) {  
        printf("%4d ", n);  
    }  
}
```

2) 열 개 학급에 대하여 출력한다.

```
int main() {  
    for(int c=1; c<=10; c++) {  
        printf("[%d반]\n", c);  
        for(int n=1; n<=30; n++) {  
            printf("%4d ", n);  
        }  
        printf("\n");  
    }  
}
```

3) 세 개 학년에 대하여 출력한다.

반복문 - 중첩된 for

- 중첩된 for 문을 이용하여 구구단 출력하기

```
#include <stdio.h>

int main() {
    for(int d=1; d<=5; d++) {    // 1단부터 5단까지
        printf(" %d 단\n", d);
        for(int x=1; x<=9; x++) {    // x1부터 x9까지
            printf("%d x %d = %2d \n", d, x, d*x);
        }
        printf("\n");
    }
    return 0;
}
```

```
1 단
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
1 x 4 = 4
1 x 5 = 5
1 x 6 = 6
1 x 7 = 7
1 x 8 = 8
1 x 9 = 9
```

```
2 단
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
```

```
3 단
3 x 1 = 3
```

반복문 - 중첩된 for

■ 연습문제

삼각형의 밑변 길이 정수 a 를 입력 받아 중첩된 반복문을 아래 그림과 같은 직각 삼각형 모양을 출력하는 프로그램을 작성하시오.

*	1개
**	2개
***	3개
****	4개
*****	5개
*****	:
*****	a 개

■ 정답

```
#include <stdio.h>

int main() {

}
```

```
int main(void) {
```


반복문 - 중첩된 for

■ 연습문제

삼각형의 밑변 길이 정수 a 를 입력 받아 중첩된 반복문을 아래 그림과 같은 직각 삼각형 모양을 출력하는 프로그램을 작성하시오.

*	공백?+ 별n개=a개
**	$\therefore ? = a-n$

■ 정답

```
#include <stdio.h>

int main() {

}

}
```

```
#include <stdio.h>
```

```
#include <stdio.h>
```

제어문 – break

■ break 문

- switch, for, while, do ~ while문의 영역을 빠져 나오기 위해 사용
- 가장 가까운 루프를 벗어난다.

■ 사용 예

- $1+2+3+\dots+n$ 의 합이 처음으로 100이상이 될 때, 그 때의 합과 n을 구하는 프로그램을 작성하시오.

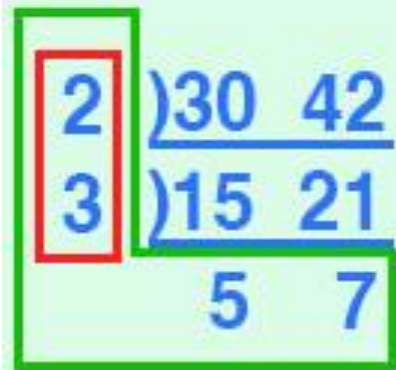
```
#include <stdio.h>

int main() {
    int sum=0, n;

    for(n=1; true; n++) {
        sum = sum + n;
        if(sum >= 100)
            break;
    }
    printf("n: %d, sum: %d \n", n, sum);
    return 0;
}
```

■ 최대공약수

- Greatest Common Divisor, GCD
- 공약수: 여러 수의 공통된 약수
- 최대공약수: 여러 수의 공약수 중 최대인 수



최대공약수: 2×3

최소공배수: $2 \times 3 \times 5 \times 7$

- $G = \gcd(30, 42) = 6$
- $L = \text{lcm}(30, 42) = 210$

■ 최소공배수

- Lowest Common Multiple, LCM
- 공배수: 여러 수의 공통된 배수
- 최소공배수: 공배수 중 최소인 수

$$\begin{array}{l} 30 = 2 \times 3 \times 5 \\ 42 = 2 \times 3 \times 7 \end{array}$$

최대공약수: 2×3
최소공배수: $2 \times 3 \times 5 \times 7$

- $A = 30, B = 42$
- $A \times B = G \times L$
- $30 \times 42 = 6 \times 210$

최대공약수 구하기

■ 공약수 탐색 전략

- $a < b$ 조건 이용
- 공약수는 1 부터 a 사이에 존재
- a부터 1순서로 탐색

■ 소스코드

```
int main() {  
    int a, b;  
    scanf("%d %d", &a, &b);  
  
    printf("test at ");  
    for(int i=a; i>=1; i--) {  
        printf("%d ", i);  
        if(a%i==0 && b%i==0) {  
            printf("\nfound: %d\n", i);  
            break;  
        }  
    }  
}
```

■ 실행결과

```
30 42  
test at 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13  
12 11 10 9 8 7 6  
found: 6
```

최대공약수 구하기 <유클리드 호제법>

① 수가 크면 계산이 복잡

$$\begin{array}{r}
 2 \mid 2304 \quad 1440 \\
 \hline
 2 \times 1152 \quad 720 \\
 2 \times 576 \quad 360 \\
 2 \times 288 \quad 180 \\
 2 \times 144 \quad 90 \\
 2 \times 72 \quad 45 \\
 3 \times 24 \quad 15 \\
 3 \times 8 \quad 5 \\
 \hline
 288 \quad 5
 \end{array}$$

② 약수 찾기가 어려운 경우

$$\begin{array}{r}
 31 \mid 403 \quad 155 \\
 \hline
 13 \quad 5
 \end{array}$$

a b 몫(s), 나머지(r)
① 큰수를, 작은수로 나눈다

② 나누는 수를, 나머지로 계속 나눈다
나머지가 0 이 나오면,
나누는 수가 최대공약수

⑥ 1512, 1008 최대공약수

$$\begin{array}{r}
 a \quad b \quad r \\
 ① \quad 1512 \div 1008 = 1 \dots 504 \\
 \quad \quad \quad \swarrow \quad \searrow \\
 ② \quad 1008 \div \boxed{504} = 2 \dots \underline{0}
 \end{array}$$

<유클리드 호제법>

a와 b의 최대공약수는

b와 r의 최대공약수와 같다.

$$\begin{array}{r}
 a \quad b \quad r \\
 ① \quad 2304 \div 1440 = 1 \dots 864 \\
 \quad \quad \quad \swarrow \quad \searrow \\
 ② \quad 1440 \div 864 = 1 \dots 576 \\
 \quad \quad \quad \swarrow \quad \searrow \\
 \quad \quad 864 \div 576 = 1 \dots 288 \\
 \quad \quad \quad \swarrow \quad \searrow \\
 \quad \quad 576 \div \boxed{288} = 2 \dots \underline{0}
 \end{array}$$

$$\begin{array}{r}
 ① \quad 403 \div 155 = 2 \dots 93 \\
 \quad \quad \quad \swarrow \quad \searrow \\
 ② \quad 155 \div 93 = 1 \dots 62 \\
 \quad \quad \quad \swarrow \quad \searrow \\
 \quad \quad 93 \div 62 = 1 \dots 31 \\
 \quad \quad \quad \swarrow \quad \searrow \\
 \quad \quad 62 \div \boxed{31} = 2 \dots \underline{0}
 \end{array}$$

최대공약수 구하기

■ 문제

두 수 a , b 를 입력 받아 최대공약수를 출력하는 프로그램을 작성하시오.

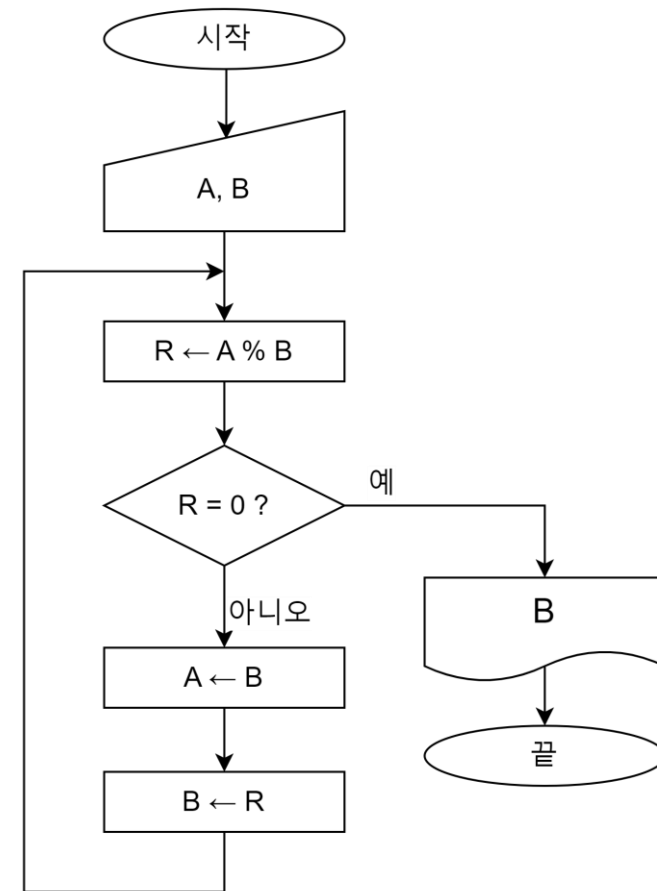
예를 들어, 12과 16의 최대공약수는 4이다.

12 16
4

■ 고찰

- while 문이 적합한가?
- do ... while 문이 적합한가?

■ 유클리드 호제법 순서도



최대공약수 구하기 (유클리드 호제법이용)

- while 문으로 구현

- do ... while 문으로 구현

최대공배수 구하기

■ 문제

두 수 a , b 를 입력 받아 최대공배수를 출력하는 프로그램을 작성하시오.

예를 들어, 6과 8의 최소공배수는 24이다.

6 8
24

■ 전략

- 방금 전에 만든 최대공약수 프로그램을 약간 개조하자!

$A \times B = G \times L$

제어문 – continue

■ continue 문

- 반복문을 진행하던 중 조건 판단 부분으로 제어를 이동

■ 사용 예

- 1부터 20까지의 정수 중에서 홀수만을 출력하시오. (for, continue 문을 사용할 것.)

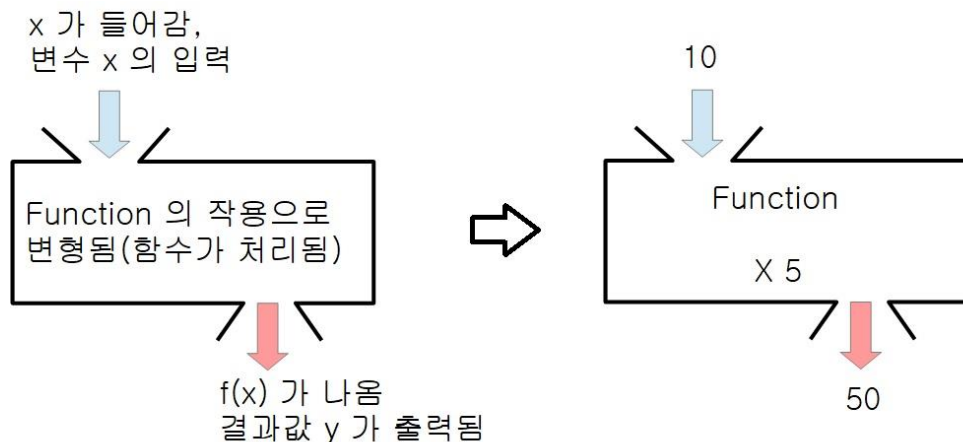
```
#include <stdio.h>

int main() {
    int i;
    for(i=1; i<=20; i++) {
        if(i%2==0)
            continue;
        printf("%3d ", i);
    }
    return 0;
}
```

함수

■ 함수(function) 란?

- 특정한 처리.기능을 수행하는 코드를 하나로 묶어 둔 것.
- 특정 인자를 받아 결과값을 반환하는 개체를 말하기 때문에 서브루틴(subroutine)이라고도 한다.



■ 함수 사용의 효과

- 코드들을 기능 단위로 묶을 수 있기 때문에 프로그램을 이해하고 만들기 쉽게 한다.

■ 함수의 종류

- 내장 함수
- 사용자정의 함수
- 매크로 함수

■ 함수=프로시저=메소드

내장 함수

■ 헤더파일의 종류

종류	기능	내장함수
stdio.h	표준 입출력 함수 등을 정의	printf() , scanf() , gets() , getchar(), puts() , putchar(), fgetc(), fgets(), fputc(), fputs(), fopen(), fclose() 등
conio.h	직접 콘솔 입출력 함수 등을 정의	getch(), getch(), putch(), cgets() 등
math.h	수학 함수와 매크로 정의	sin(), cos(), tan(), exp(), log(), sqrt() , abs() , fabs(), pow() , fmod() 등
string.h	문자열 처리 함수 정의	strcat() , strcpy() , strcmp() , strncat() 등
ctype.h	문자 검사 매크로 정의	isalpha(), islower(), isupper(), tolower(), toupper() 등

사용자 정의 함수

■ 형식

[함수의 리턴형] 함수명([인수1, 인수2...])

{

문장1;

문장2;

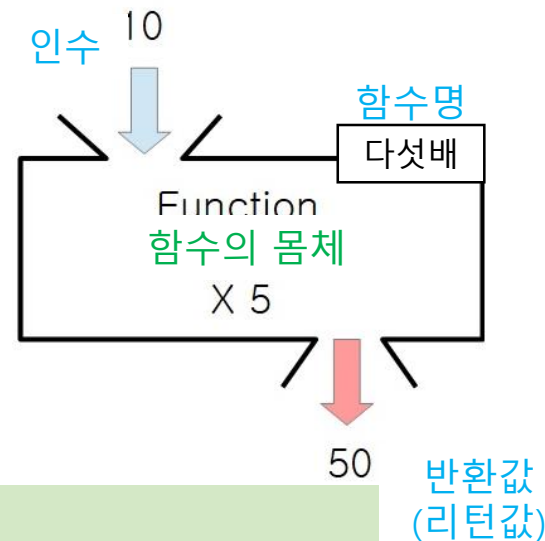
...

...

문장n;

[return] [리턴값]

}



■ 예

[리턴형] 함수명 (인수)

int main () {

함수의 몸체

}

int add(int a, int b) {

int sum = a + b;

return sum;

}

char upper(char ch) {

return ch-32;

}

사용자 정의 함수

■ 함수의 형태

- 인수
 - 없는가?
 - 있는가? 있다면 한 개인가, 두 개인가?
- 리턴값
 - 없는가?
 - 있는가? (한 개만 리턴 가능)
- 다양한 형태의 함수 모양이 나올 수 있음

■ 형태

인수	리턴	형태
X	X	void func() void func(void)
X	O	int func() double func(void)
O	X	void func(int ar) void func(char c)
O	O	int func(int ar) int func(int a, int b)

사용자 정의 함수

■ Case1: 인수 x, 리턴값 x

- 기능: "Copyright" 출력
- 함수명: output
- 인수: 없음
- 리턴값: 없음

■ 함수 구현

```
void output() {  
    printf("-----\n");  
    printf(" function test\n");  
    printf("-----\n");  
    return;  
}
```

사용자 정의 함수

■ Case2: 인수 0, 리턴 0

- 기능: $x+y$ 값을 구한다
- 함수명: add
- 인수: x, y 2개
 $x:\text{int}, y:\text{int}$
- 리턴값: $x+y$
 리턴형: int

■ 함수 구현

```
int add(int x, int y) {  
    int sum = x + y;  
    return sum  
}
```

사용자 정의 함수

■ 함수의 호출

```
#include <stdio.h>
int add(int x, int y) {
    int sum = x + y;
    return sum;
}

int main() {
    int a=3, b=4;
    int sum=0;

    sum = add(a, b);
    printf("%d \n", sum);
}
```

■ 메모리에서 일어나는 일

- 지역변수는 함수 호출 시 생성 되고, 함수가 종료되면 자동으로 파괴된다.

소속	변수	값
add	sum	7
	y	4
	x	3

소속	변수	값
main	sum	7
	b	4
	a	3

copy

사용자 정의 함수

```
#include <stdio.h>

int add(int a, int b) {
    int sum = a + b;
    return sum;
}

int pow(int x, int y) {
    int r=1;
    for(int i=1; i<=y; i++)
        r = r*x;
    return r;
}

char upper(char ch) {
    return ch-32;
}
```

```
void output() {
    printf("-----\n");
    printf(" function %cest\n", upper('t')); //함수호출
    printf("-----\n");
    printf("2+3 = %d\n", add(2,3)); //함수호출
    printf("2^3 = %d\n", pow(2,3)); //함수호출
    return;
}

int main() {
    output(); //함수호출
}
```

Debugging: Step into [shift]+F7

main.cpp [HelloWorld] - Code::Blocks 20.03

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

Debug

<global> add(int a, int b) : int

S C

Management

Projects Files FSy

Workspace

Start

Sources

main.cpp

HelloWorld

Sources

```
2
3 int add(int a, int b) {
4     int sum = a + b;
5     return sum;
6 }
7
8 int pow(int x, int y) {
9     int r=1;
10    for(int i=1; i<=y; i++)
11        r = r*x;
12    return r;
13 }
14
15 char upper(char ch) {
16     return ch-32;
17 }
18
19 void output() {
20     printf("-----\n");
21     printf(" function %cest\n", upper('t'));
22     printf("-----\n");
23     printf("2+3 = %d\n", add(2,3));
24     printf("2^3 = %d\n", pow(2,3));
25     return;
26 }
27
28 int main() {
29     output();
```

step into
디버깅 기능을
통해 함수 호출
순서를
차례대로 관찰

Watches

Function

a 2

b 3

Locals

sum 5

ch Not avail

a 2 int

Logs & others

Code::Blocks Search results Cccc Build log Build messages CppCheck/Vera++ CppCheck/Vera++ messages Cscope Debugger DoxyBlocks Fortran info

At D:\MyProjects\Algorithm\HelloWorld\main.cpp:23

At D:\MyProjects\Algorithm\HelloWorld\main.cpp:4

At D:\MyProjects\Algorithm\HelloWorld\main.cpp:5

Command:

사용자 정의 함수

함수의
프로토타입을
미리 알려준다

```
#include <stdio.h>
```

```
int add(int a, int b);  
int pow(int x, int y);  
char upper(char ch);
```

```
void output() {  
    printf("-----\n");  
    printf(" function %cest\n", upper('t'));  
    printf("-----\n");  
    printf("2+3 = %d\n", add(2,3));  
    printf("2^3 = %d\n", pow(2,3));  
    return;  
}
```

```
int main() {  
    output();  
}
```

```
int add(int a, int b) {  
    int sum = a + b;  
    return sum;  
}
```

```
int pow(int x, int y) {  
    int r=1;  
    for(int i=1; i<=y; i++)  
        r = r*x;  
    return r;  
}
```

```
char upper(char ch) {  
    return ch-32;  
}
```

Logs & others

Code::Blocks x Search results x Cccc x Build log x

File	Line	Message
D:\MyProjects...	5	error: 'upper' was not declared in this scope
D:\MyProjects...	5	note: suggested alternative: '_popen'
D:\MyProjects...	7	error: 'add' was not declared in this scope
D:\MyProjects...	8	error: 'pow' was not declared in this scope
D:\MyProjects...	8	note: suggested alternative: 'putw'

지역변수 / 전역변수

■ 지역변수

- 함수 안에서 선언된 변수
- 해당 함수 안에서만 사용가능
- 초기값이 쓰레기 값이다
- 함수가 호출되면 생성되고 함수가 종료되면 사라진다
- 동일한 이름의 전역/지역변수가 존재하면 지역변수가 우선한다
- 스택에 저장

■ 전역변수

- 함수 외부에서 선언된 변수
- 어느 함수에서든 사용가능
- 초기값이 0 이다
- 프로그램이 실행 중이면 항상 존재한다
- 프로그램을 이해하기 어렵게 만드므로 꼭 필요한 경우에만 사용하자
- 전역공간에 저장

지역변수 / 전역변수

■ 지역변수 예시

```
#include <stdio.h>
// add의 sum과 main의 sum은 동명이인

int add(int a, int b) {
    int sum = a + b;
    return sum;
}

int main(){
    int sum=0;
    add(1, 2);
    printf("%d\n", sum);
}
```

■ 전역변수 예시

```
#include <stdio.h>
// add의 sum과 main의 sum은 동일변수

int sum;
void add(int a, int b) {
    sum = a + b;
}

int main(){
    add(1, 2);
    printf("%d\n", sum);
}
```

팩토리얼 계산

팩토리얼

$$n! = n \times (n-1) \times (n-2) \times (n-3) \times \dots \times 3 \times 2 \times 1$$



계승: 계단을 내려가듯 위에서 아래로 순서대로 곱함, 또는 계단을 올라가듯 아래에서 위로 순서대로 곱함.

팩토리얼 계산

■ 비 재귀적 해결

- ex) 팩토리얼 계산

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 1$$

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

$$3! = 3 \times 2 \times 1 = 6$$

■ 답안

```
#include <stdio.h>
int factorial(int n) {

    ?

    return r;
}
int main() {
    printf("6! = %d\n", factorial(6));
}
```

재귀함수

■ 재귀함수

- 실행 도중 자기 자신을 호출(재귀 호출) 하는 함수
- ex) 팩토리얼 계산

$$5! = 5 \times 4 \times 3 \times 2 \times 1$$

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2!$$

$$2! = 2 \times 1!$$

$$1! = 1$$

$$f(n) = \begin{cases} 1 & \dots n=1 \\ n \times f(n-1) & \dots n \geq 2 \end{cases}$$

■ 함수 예시

- 탈출조건이 없으면 무한루프가 되므로 유의

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

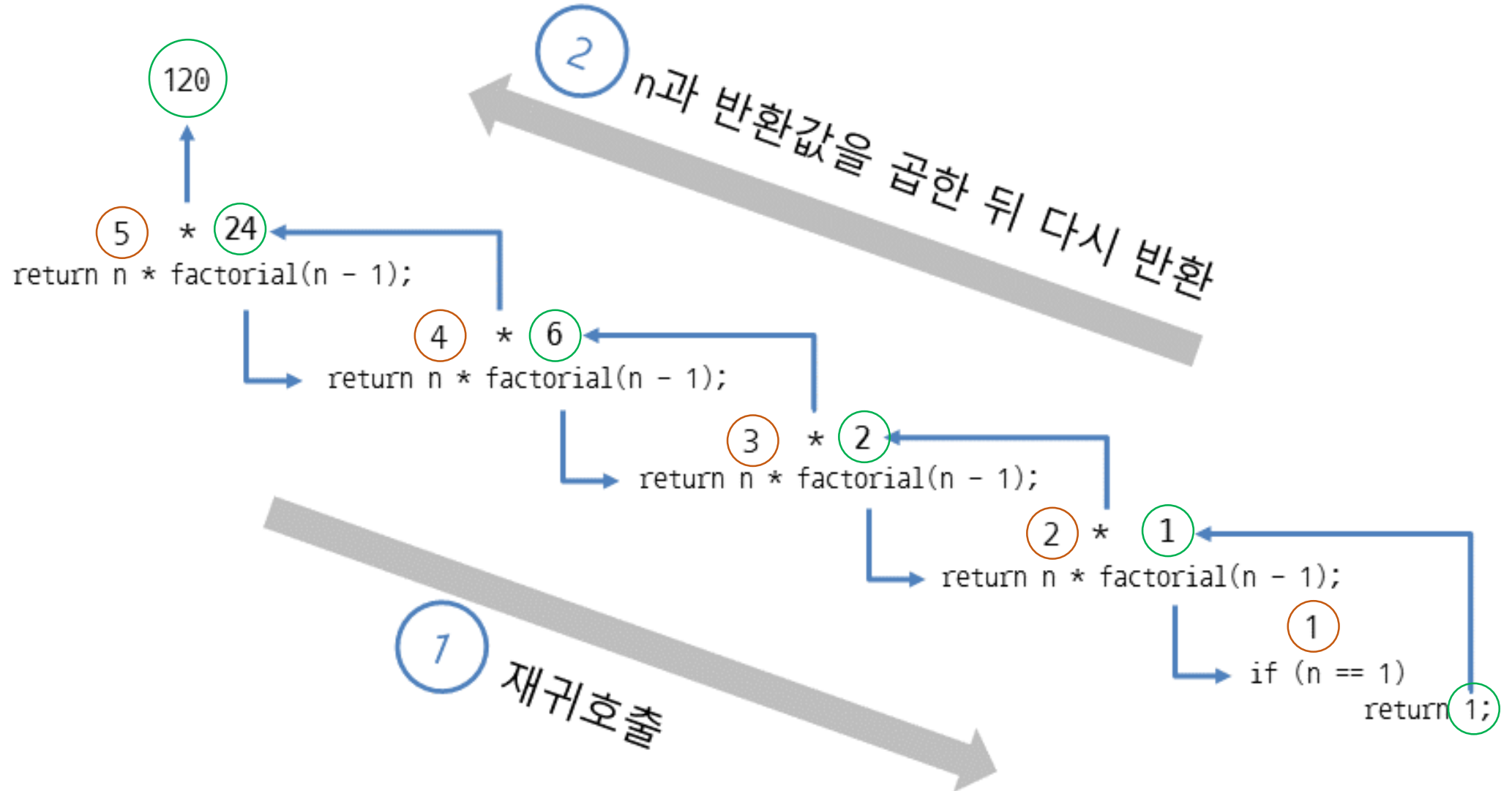
```
// 3항 조건 연산자를 활용하여  
// 아래와 같이 표현해도 동일한 효과  
int fact(int n) {  
    return (n<=1)? 1 : n*fact(n-1);  
}
```


재귀함수

factorial(5)

계산과정

묘사



재귀함수

■ 재귀함수 호출 관찰

```
#include <stdio.h>

int fact(int n) {
    if(n <= 1)    // 탈출조건
        return 1;
    else {
        printf("[%d x %d! ", n, n-1);
        int f = fact(n-1);
        printf("(d!=%d)] ", n-1, f);
        return n*f;
    }
}

int main() {
    printf("%d", fact(5));
}
```

■ 함수 예시

[5 x 4!
[4 x 3!
[3 x 2!
[2 x 1!
(1!=1)]
(2!=2)]
(3!=6)]
(4!=24)]
120

연습문제

■ 표보나치 수 찾기

- 첫째 항 및 둘째 항이 1이며, 그 뒤의 모든 항은 바로 앞 두 항의 합인 수열
- 처음 여섯 항은 각각 1, 1, 2, 3, 5, 8 이다.
- 숫자 k 를 입력 받아 k 번째에 해당하는 피보나치 수를 출력하는 알고리즘을 작성하시오.

■ 함수 표현

```
int fibo(int n) {
```

재귀 함수 - 연습문제

■ 계단을 오르는 방법

- 계단을 한 번에 한 칸 또는 두 칸 만 오를 수 있다고 할 때 n 칸으로 되어 있는 계단 전체를 오르는 방법은 몇 가지가 있는가?

• 힌트1

- 1칸 계단: 1가지 방법
- 2칸 계단: 2가지 방법
- 3칸 계단은?

• 힌트2: n 칸 계단에 오르는 방법

- $n-2$ 칸 까지 올라온 다음 두 칸 오른다 +
- $n-1$ 칸 까지 올라온 다음 한 칸 오른다

■ 함수로 표현

예를 들어,

$f(n)$: n 개의 계단일 때 오르는 방법의 수

$$f(1) = 1$$

$$f(2) = 2$$

$$f(3) = f(2) + f(1)$$

$$f(4) = f(3) + f(2)$$

$$f(5) = f(4) + f(3)$$

:

$$f(n) = f(n-1) + f(n-2)$$

재귀 함수 - 연습문제

■ 계단을 오르는 방법

- 계단을 한 번에 한 칸 또는 두 칸만 오를 수 있다고 할 때 n 칸으로 되어 있는 계단 전체를 오르는 방법은 몇 가지가 있는가?

• 힌트

- 1칸 계단: 1가지 방법
- 2칸 계단: 2가지 방법
- 3칸 계단은?

```
#include <stdio.h>

int count(int stairs) {

}

int main() {
    int stairs;

    printf("input number of stairs: ");
    scanf("%d", &stairs);
    printf("total method of climbing: %d\n", count(stairs));
}
```

배열

■ 배열

- 같은 형식의 여러 데이터를 하나의 변수에 긴 띠 모양으로 저장하여 사용하는 자료의 집합체
- 줄줄이 연결된 타입이 동일한 변수들의 집합

■ 선언

- 형식

데이터형 배열명[원소의 수]

- 선언 예

```
int kor[5];
```

kor변수 5개 만듦

kor[0] ~ kor[4]

- 배열의 구조

- 0번 인덱스부터 시작됨에 유의

kor[0]	kor[1]	kor[2]	kor[3]	kor[4]
--------	--------	--------	--------	--------

배열

■ 대입

```
kor[0] = 60;
```

```
kor[1] = 60;
```

```
kor[2] = 60;
```

```
kor[3] = 60;
```

```
kor[4] = 60;
```

■ 반복문 이용한 대입

```
for(i=0; i<5; i++)
```

```
    kor[i] = 60;
```

■ 초기화

```
int a[5] = {3,2,7,6,9};
```

```
int b[] = {3,6,5,4};
```

```
int c[5] = {5,8,3};
```

```
int d[5] = {4,};
```

```
static int e[5];
```

배열

■ 배열의 순회1

```
#include <stdio.h>

int main() {
    int a[10]={1,3,7,6,4,8,9,12,2,10};

    // 0번부터 시작하여 n-1에서 끝남에 유의
    for(int i=0; i<10; i++) {
        printf("%4d", a[i]);
    }
    return 0;
}
```

■ 배열의 순회2

```
#include <stdio.h>

int main() {
    int a[10]={1,3,7,6,4,8,9,12,2,10};
    int i;
    // 0번부터 시작하여 n-1에서 끝남에 유의
    i=0;
    while(i<10) {
        printf("%4d", a[i]);
        i++;
    }
    return 0;
}
```


배열

■ 배열의 합

```
#include <stdio.h>

void main() {
    int i, sum=0;
    int a[10]={1,3,7,6,4,8,9,12,2,10};

    for(i=0; i<10; i++) {
        sum = sum + a[i];
    }
    printf("sum = %d\n", sum);
}
```

■ 피보나치수

```
#include <stdio.h>

void main() {
    int i, fibo[10]={1,1};

    for(i=2; i<10; i++) {
        fibo[i]=fibo[i-1]+fibo[i-2];
    }

    for(i=0; i<10; i++) {
        printf("%4d\n", fibo[i]);
    }
}
```

숫자 목록에서 수 찾기(선형탐색)

■ 문제

n개로 이루어진 정수 목록에서 원하는 수의 위치를 찾으시오.
단, 입력되는 정수 목록에 같은 수는 없다.

■ 입력

첫 줄에 한 정수 n이 입력된다.
($2 \leq n \leq 100,000$)
둘째 줄에 n개의 정수가 공백으로 구분되어 입력된다.
(입력되는 모든 정수는 21억 보다 작다)
셋째 줄에는 찾고자 하는 수가 입력된다.

■ 출력

찾고자 하는 원소의 위치를 출력한다.
없으면 -1을 출력한다.

■ 입력과 출력의 예

입력 예	출력 예
8 1 2 3 5 7 9 11 15 11	7

최댓값 찾기

■ 문제

9개의 서로 다른 자연수가 주어질 때, 이들 중 최댓값을 찾고 그 값이 몇 번째 수 인지를 구하는 프로그램을 작성하시오. 예를 들어, 서로 다른 9개의 자연수가 각각 3, 29, 38, 12, 57, 74, 40, 85, 61 라면, 이 중 최댓값은 85이고, 이 값은 8번째 수이다

■ 입력

첫째 줄부터 아홉째 줄까지 한 줄에 하나의 자연수가 주어진다. 주어지는 자연수는 100보다 작다.

■ 출력

첫째 줄에 최댓값을 출력하고, 둘째 줄에 최댓값이 몇 번째 수인지를 출력한다.

■ 입력과 출력의 예

입력 예	출력 예
3	85
29	8
38	
12	
57	
74	
40	
85	
61	

■ 출처

한국정보올림피아드(2007 지역본선 초등부)

배열

■ 에라토스테네스의 체

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

isPrime[]	idx	0	1	2	3	4	5	6	...
	val	1	1	0	0	0	0	0	...

■ 소스코드

```

#include <stdio.h>
#define MAX 101
int main() {
    int cnt=0;
    int isPrime[MAX] = {1, 1, 0, };
    // 1: 소수아님, 0: 소수
    for(int i=2; i<MAX; i++) {
        if(isPrime[i]==0) { // 현재수만 소수이고
            printf("%5d", i);
            cnt++;
            for(int j=i; j<MAX; j+=i) // 배수들은
                isPrime[j]=1; // 소수아님으로 셋팅
        }
    }
    printf("1~100 %d개의 소수를 찾아냄\n", cnt);
}

```

SWAP

- 두 변수 내용물을 서로 교환하는 연산



- 잘못된 구현

```
#include <stdio.h>

int main() {
    int a=5, b=7;
    printf("%d %d\n", a, b);

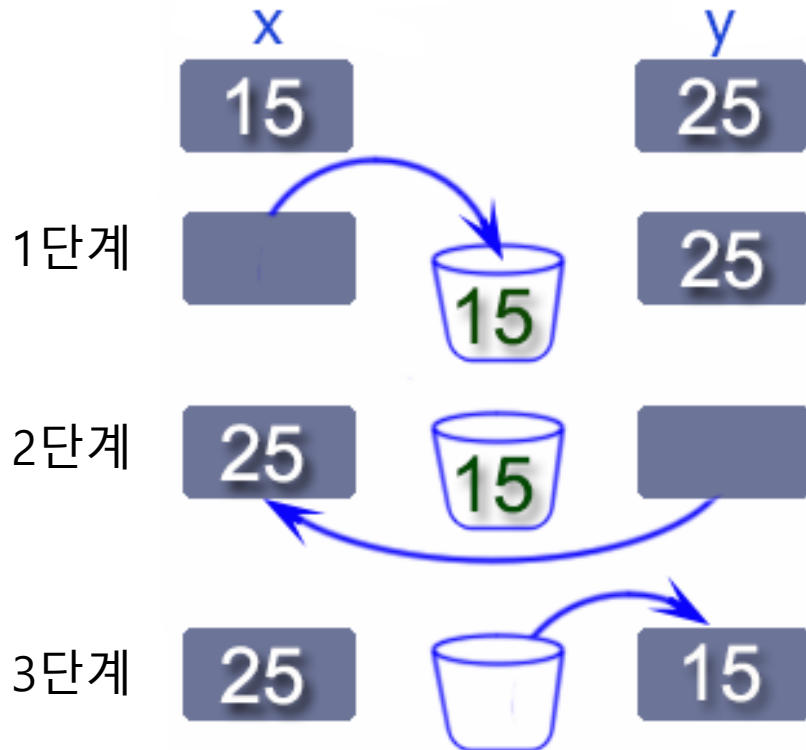
    a=b;
    b=a;

    printf("%d %d\n", a, b);
    return 0;
}
```

5	7
7	7

SWAP

- 두 변수 내용물을 서로 교환하는 연산
- 임시 변수가 필요함.



- 올바른 구현

```
#include <stdio.h>

int main() {
    int a=5, b=7;
    printf("%d %d\n", a, b);

    int t=a;
    a=b;
    b=t;

    printf("%d %d\n", a, b);
    return 0;
}
```

5 7
7 5

SWAP

■ 고급 구현

```
#include <stdio.h>

// C++의 Generic과 참조자를 사용
template <class T>
inline void SWAP(T& a, T& b)
{
    T temp = a;
    a = b;
    b = temp;
}
```

제네릭과
참조자는 본
수업의 범위를
벗어나는
내용이므로
자세한 설명은
생략한다.

```
int main() {
    int a=5, b=7;
    printf("%d %d\n", a, b);

    SWAP(a, b);

    printf("%d %d\n", a, b);
    return 0;
}
```

5	7
7	5

정렬 알고리즘

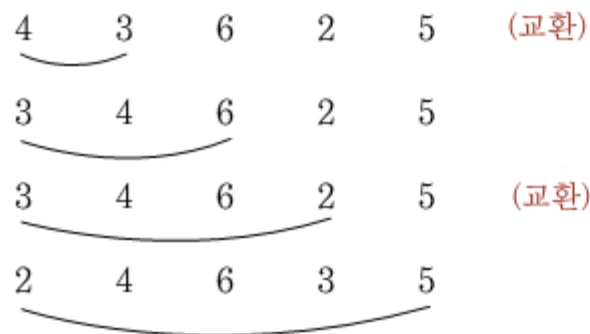
■ 선택 정렬

- 각 회전마다 최소 값을 찾아 1번째 부터 차례대로 배열
- 1회전이 종료될 때 마다 가장 앞쪽부터 차례대로 숫자가 결정됨

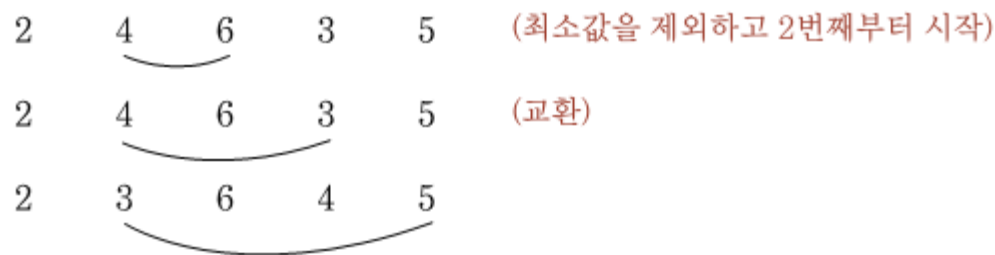
• 예

- 4 3 6 2 5 를 선택정렬

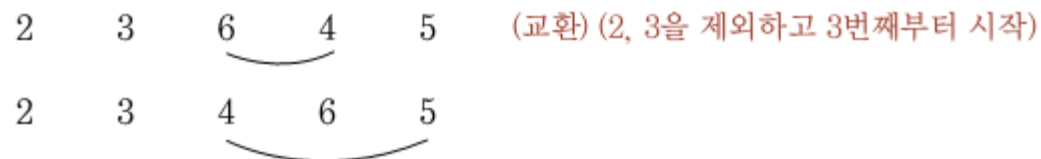
- 1회전 (첫 번째 숫자를 결정하기 위함)



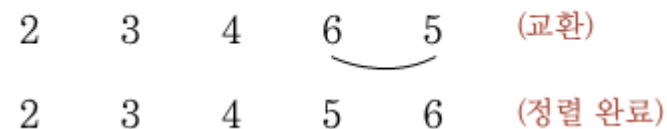
- 2회전 (두 번째 숫자를 결정하기 위함)



- 3회전 (세 번째 숫자를 결정하기 위함)



- 4회전 (네 번째 숫자를 결정하기 위함), 5개를 정렬 하려면 4회전 필요



정렬 알고리즘

■ 구현

```
void selection_sort(int a[], int len)
{
    int i, j, t;

    for(i=0; i<len-1; i++) {
        for(j=i+1; j<len; j++) {
            if(a[i] > a[j]) {
                t = a[i];
                a[i] = a[j];
                a[j] = t;
            }
        }
    }
}
```

■ 1회전

i	j			
4	3	6	2	5

i		j		
3	4	6	2	5

i			j	
3	4	6	2	5

i				j
2	4	6	5	5

정렬 알고리즘

■ 구현

```
void selection_sort(int a[], int len)
{
    int i, j, t;

    for(i=0; i<len-1; i++) {
        for(j=i+1; j<len; j++) {
            if(a[i] > a[j]) {
                t = a[i];
                a[i] = a[j];
                a[j] = t;
            }
        }
    }
}
```

■ 2회전

	i	j		
2	4	6	3	5

	i		j	
2	4	6	3	5

	i			j
2	3	6	4	5

정렬 알고리즘

■ 구현

```
void selection_sort(int a[], int len)
{
    int i, j, t;

    for(i=0; i<len-1; i++) {
        for(j=i+1; j<len; j++) {
            if(a[i] > a[j]) {
                t = a[i];
                a[i] = a[j];
                a[j] = t;
            }
        }
    }
}
```

■ 3회전

		i	j	
2	3	6	4	5

		i		j
2	3	4	6	5

■ 4회전

			i	j
2	3	4	6	5

			i	j
2	3	4	5	6

정렬 알고리즘

■ 선택 정렬 테스트

```
#include <stdio.h>

// 길이가 len인 배열 a를 오름차순 정렬
void selection_sort(int a[], int len) {
    int i, j, t;
    for(i=0; i<len-1; i++) {
        for(j=i+1; j<len; j++) {
            if(a[i] > a[j]) {
                t = a[i];
                a[i] = a[j];
                a[j] = t;
            }
        }
    }
}
```

Quiz) 만약 내림차순 정렬로 바꾸려면
어느 곳을 수정하며 될까?

```
// 길이가 len인 배열 a의 모든 원소를 출력
void show_array(int a[], int len) {
    for(int i=0; i<len; i++)
        printf("%5d", a[i]);
    printf("\n\n");
}

int main() {
    int a[10] = {7, 5, 8, 1, 4, 9, 2, 10, 6, 3};
    show_array(a, 10);

    selection_sort(a, 10);
    show_array(a, 10);
    return 0;
}
```

STL sort() 함수 사용하기

■ sort() 함수의 사용

- C++의 STL 사용
- `#include <algorithm>` 필요
- `sort(배열시작, 배열끝, [비교함수])`
- 기본 비교함수는 오름차순

■ 비교함수(true일 때 SWAP 됨)

// 오름차순용

```
bool asc_order(int a, int b) {  
    return a < b;  
}
```

// 내림차순용

```
bool desc_order(int a, int b) {  
    return a > b;  
}
```

```
#include <stdio.h>  
#include <algorithm>  
using namespace std;  
  
void show_array(int a[], int len) {  
    for(int i=0; i<len; i++)  
        printf("%5d", a[i]);  
    printf("\n\n");  
}  
  
int main() {  
    int a[10] = {7, 5, 8, 1, 4, 9, 2, 10, 6, 3};  
    show_array(a, 10);  
  
    sort(a, a+10); // 오름차순 정렬  
    show_array(a, 10);  
  
    sort(a, a+10, desc_order); // 내림차순 정렬  
    show_array(a, 10);  
    return 0;  
}
```

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <algorithm>
using namespace std;

#define LEN 100

void selection_sort(int a[], int len) {
    int i, j, t;

    for(i=0; i<len-1; i++) {
        for(j=i+1; j<len; j++) {
            if(a[i] > a[j]) {
                t = a[i];
                a[i] = a[j];
                a[j] = t;
            }
        }
    }
}

void show_array(int a[], int len) {
    for(int i=0; i<len; i++)
        printf("%5d", a[i]);
    printf("\n\n");
}

```

```

void rand_array(int a[], int len, int max) {
    srand(time(NULL));

    for(int i=0; i<len; i++)
        a[i] = rand() % max;
}

bool desc_order(int a, int b) {
    return a > b;
}

int main() {
    int a[LEN];
    rand_array(a, LEN, LEN*10);
    show_array(a, LEN); // before sorting

    selection_sort(a, LEN); // selection ASC sort
    show_array(a, LEN);    // after sorting

    sort(a, a+LEN, desc_order); // DESC sort
    show_array(a, LEN);        // after sorting

    return 0;
}

```

정렬하여 k번째 수 찾기

■ 문제

n개의 정수를 배열에 입력 받아 정렬한 뒤, k번째로 큰 숫자를 찾는 프로그램을 작성하시오. 만약 네 개의 정수 1, 2, 3, 4가 입력되었다면, 3번째로 큰 수는 2이다.

■ 입력

첫 번째 줄에 입력 받을 자료의 개수 n이 입력된다. 두 번째 줄부터 정수 n개가 한 줄에 하나씩 차례대로 입력된다. 마지막 줄에는 k가 입력된다.

■ 출력

입력된 자료들 가운데 k번째로 큰 숫자를 출력한다.

■ 입력과 출력의 예

입력 예	출력 예
4 1 2 3 4 3	2

■ 고찰

이 문제를 풀려면 오름차순 정렬을 사용해야 하는가? 내림차순 정렬을 사용해야 하는가?

다차원 배열

■ 2차원 배열의 선언

- 1차원 배열을 여러 개 겹쳐 놓은 것
- 행과 열의 평면 구조를 가진 배열
- 선언

데이터형 배열명[행 수][열 수]

- 선언 예

int a[4][5];

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]
a[3][0]	a[3][1]	a[3][2]	a[3][3]	a[3][4]

■ 2차원 배열의 초기화

int a[2][3] = {3, 2, 7, 6, 9, 8};

3	2	7
6	9	8

int b[2][3] = {5,8,3,7};

5	8	3
7	쓰레기	쓰레기

int c[2][3] = {4, };

4	0	0
0	0	0

int d[2][3] = { {3, }, {7,6,9} };

3	0	0
7	6	9

int e[][3] = { {3,2,6}, {7,6,9} };

다차원 배열

■ 2차원 배열의 순회(행 우선)

```
#include <stdio.h>
```

```
#define ROW 3
```

```
#define COL 4
```

1	2	3	4
5	6	7	8
9	10	11	12

```
int main() {
```

```
    int a[ROW][COL] = {  
        {1,2,3,4},{5,6,7,8},{9,10,11,12} };
```

```
    for(int r=0; r<ROW; r++) {    //행 순회
```

```
        for(int c=0; c<COL; c++) { //열 순회
```

```
            printf("%4d", a[r][c]);
```

```
        }
```

```
        printf("\n");
```

```
    }
```

```
    return 0;
```

```
}
```

1	2	3	4
5	6	7	8
9	10	11	12

■ 2차원 배열의 순회(열 우선)

```
#include <stdio.h>
```

```
#define ROW 3
```

```
#define COL 4
```

1	2	3	4
5	6	7	8
9	10	11	12

```
int main() {
```

```
    int a[ROW][COL] = {  
        {1,2,3,4},{5,6,7,8},{9,10,11,12} };
```

```
    for(int c=0; c<COL; c++) {    //열 순회
```

```
        for(int r=0; r<ROW; r++) { //행 순회
```

```
            printf("%4d", a[r][c]);
```

```
        }
```

```
        printf("\n");
```

```
    }
```

```
    return 0;
```

```
}
```

1	5	9
2	6	10
3	7	11
4	8	12

격자판의 최댓값

■ 문제

<그림1>과 9x9 격자판에 쓰여진 81개의 자연수가 주어질 때, 이들 중 최댓값을 찾고 그 최댓값이 몇 행 몇 열에 위치한 수인지 구하는 프로그램을 작성하시오.

1열 2열 3열 4열 5열 6열 7열 8열 9열

1행	3	23	85	34	17	74	25	52	65
2행	10	7	39	42	88	52	14	72	63
3행	87	42	18	78	53	45	18	84	53
4행	34	28	64	85	12	16	75	36	55
5행	21	77	45	35	28	75	90	76	1
6행	25	87	65	15	28	11	37	28	74
7행	65	27	75	41	7	89	78	64	39
8행	47	47	70	45	23	65	3	41	44
9행	87	13	82	38	31	12	29	29	80

예를 들어, 왼쪽과 같이 81개의 수가 주어질 경우에는 이들 중 최댓값은 90이고, 이 값은 5행 7열에 위치한다.

<그림 1>

출처: 한국정보올림피아드(2007 지역예선 중고등부)

■ 입력

첫째 줄부터 아홉째 줄까지 한 줄에 아홉 개씩 자연수가 주어진다. 주어지는 자연수는 100보다 작다.

■ 출력

첫째 줄에 최대값을 출력하고, 둘째 줄에 최댓값이 위치한 행 번호와 열번호를 빈칸을 사이에 두고 차례로 출력한다. 최댓값이 두 개 이상인 경우 행 숫자가 가장 작은 위치를 출력한다.

입력 예	출력 예
3 23 85 34 17 74 25 52 65 10 7 39 42 88 52 14 72 63 87 42 18 78 53 45 18 84 53 34 28 64 85 12 16 75 36 55 21 77 45 35 28 75 90 76 1 25 87 65 15 28 11 37 28 74 65 27 75 41 7 89 78 64 39 47 47 70 45 23 65 3 41 44 87 13 82 38 31 12 29 29 80	90 5 7

격자판의 최댓값

■ 문제

```
#include <stdio.h>

#define ROW 9
#define COL 9

int a[ROW][COL];

void input() {
    for(int r=0; r<ROW; r++)
        for(int c=0; c<COL; c++) {
            scanf("%d", &a[r][c]);
        }
}
```

```
int main() {
    input();

    int mr, mc, max=-1;

    printf("%d\n", max);
    printf("%d %d\n", mr+1, mc+1);
    return 0;
}
```

구조체

■ 구조체란?

- C언어의 기본 타입을 가지고 새롭게 정의하는 사용자 정의 타입
- 기본 타입만으로는 나타낼 수 없는 복잡한 데이터를 표현 가능
- 배열이 같은 타입의 변수 집합이라고 한다면, 구조체는 다양한 타입의 변수 집합을 하나의 타입으로 나타낸 것
- 구조체를 구성하는 변수를 구조체의 멤버(member)라고 부른다

■ 구조체의 정의와 선언 예

```
#include <stdio.h>

typedef struct {
    int x, y; // 원의 중심점
    double r; // 원의 반지름
} circle;

int main() {
    circle c1 = {1, 2, 5};
    printf(" (%d, %d) %lf", c1.x, c1.y, c1.r);
}
```

알고리즘의 효율성

■ 이해의 복잡도

- difficulty
- 알고리즘 이해와 구현에 필요한 시간과 노력의 양



■ 시간 복잡도

- time complexity
- 문제를 해결하는데 걸리는 시간과의 함수 관계
- 반복문, 중첩된 반복문의 구조와 개수에 의해 결정

■ 공간 복잡도

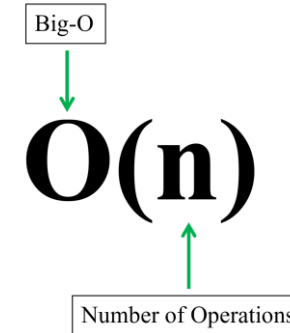
- space complexity
- 문제를 해결하기 위해 필요한 메모리(저장) 공간의 양
- 변수 및 배열의 개수와 크기에 의해 결정
- 함수가 호출될 때마다 사용되는 스택 공간이 늘어남

알고리즘의 시간 복잡도

■ 빅오(O)표기법

- 빅오 표기법은 알고리즘의 성능 평가 방법 중 가장 많이 사용하는 방법 중 하나
- 가장 많이 사용하는 이유는 최악의 성능을 표시하기 때문
- 최악의 성능 지표는, 적어도 이 정도의 성능은 보장한다는 의미
- 실행 횟수를 점근적 표기법으로 표시

■ 표기 형식



최소 n번은
연산해야
답이 나온다.

■ 실행 횟수 계산

- 프로그램은 첫번째 줄부터 마지막 줄까지 차례로 실행된다고 가정.
- 헤더 파일은 알고리즘의 성능에 영향을 주지 않는다.
- 함수 진입, 함수 반환은 알고리즘 성능에 영향을 주지 않는다.

알고리즘의 시간 복잡도

■ 프로그램 예시

```
#define N 100          // 영향을 주지 않는다.  
#include <stdio.h>     // 영향을 주지 않는다.  
  
void main(int)         // 영향을 주지 않는다.  
{  
    int sum = 0;        // 실행 횟수: 1회  
    int i;              // 실행 횟수: 1회  
  
    for(i=1; i<=N; i++) { // 실행 횟수: N+1회  
        sum = sum + i;    // 실행 횟수: N회  
    }  
  
    printf("sum:%d\n",sum); // 실행 횟수: 1회  
    // 총 횟수: 1 + 1 + N+1 + N + 1 = 2N + 4회  
}
```

■ 실행 횟수 계산

- 상수항은 무시
 - $O(101) \rightarrow O(1)$
 - $O(2N + 1) \rightarrow O(N)$
- 지배적이지 않은 항은 무시
 - $O(N^2 + N) \rightarrow O(N^2)$
 - $O(N + \log N) \rightarrow O(N)$
 - $O(100 \times 2^N + 500N^2) \rightarrow O(2^N)$

■ 예시 프로그램의 Big-O: $O(N)$

빅오 표기의 종류

■ $O(1)$

- 상수시간(constant time)
- 데이터 양과 상관없이 문제 해결에 항상 정해진 시간이 걸림
- 평가: 최상의 알고리즘
- 알고리즘 예
 - 정수의 홀짝 판별
 - 가우스의 1~N 누계 구하기
 - (시작수+마지막수) x n / 2

■ $O(\log N)$

- 로그시간(logarithmic)
- 데이터 양이 증가함에 따라 실행 시간이 로그 함수 그래프로 나타남
- 데이터가 많이 늘어나도 실행시간은 약간만 증가하는 특징
- 평가: 좋은 알고리즘
- 알고리즘 예
 - 이진탐색
 - 10개 일때, $\log_2 10 = 3.x$
 - 100개 일때, $\log_2 100 = 6.x$
 - 1000개 일때, $\log_2 1000 = 9.x$

빅오 표기의 종류

■ $O(N)$

- 선형시간(linear time)
- 데이터 양의 증가에 따라 실행 시간이 일차 함수 그래프로 나타남
- 데이터 증가량과 정비례하여 실행 시간이 증가하는 특징
- 평가: 준수한 알고리즘
- 알고리즘 예
 - 정렬되지 않은 배열에서 최댓값 찾기

■ $O(N \log N)$

- 선형 로그 시간(linearithmic time)
- 데이터 양의 증가에 따라 실행 시간이 일차 함수 + 로그함수 형태의 그래프로 나타남
- 데이터의 증가량보다 실행시간이 더 많이 증가하는 특징
- 평가: 그저 그런 알고리즘
- 알고리즘 예
 - 힙 정렬
 - 자이델(Seidel)의 다각형 삼각

빅오 표기의 종류

■ $O(N^2)$

- 이차식 시간(quadratic time)
- 데이터 양의 증가에 따라 실행 시간이 이차 함수(N^2) 그래프로 나타남
- 데이터 증가량에 제곱으로 비례하여 실행 시간이 증가하는 특징
- 평가: 나쁜 알고리즘
- 알고리즘 예
 - 선택정렬
 - 버블정렬

■ $O(N^3)$

- 삼차식 시간(cubic time)
- 데이터 양의 증가에 따라 실행 시간이 삼차 함수(N^3) 그래프로 나타남
- 데이터 증가량과 정비례하여 실행 시간이 증가하는 특징
- 평가: 끔찍한 알고리즘
- 알고리즘 예
 - 행렬 2개의 무식한 곱셈

빅오 표기의 종류

■ $O(2^N)$

- 지수 시간(exponential time)
- 데이터 양의 증가에 따라 실행 시간이 지수 함수(2^N) 그래프로 나타남
- 데이터 증가량에 따라 실행 시간이 지수 형태로 증가하는 특징
- 평가: 최악의 알고리즘
- 알고리즘 예
 - 2^N 을 재귀 호출로 계산

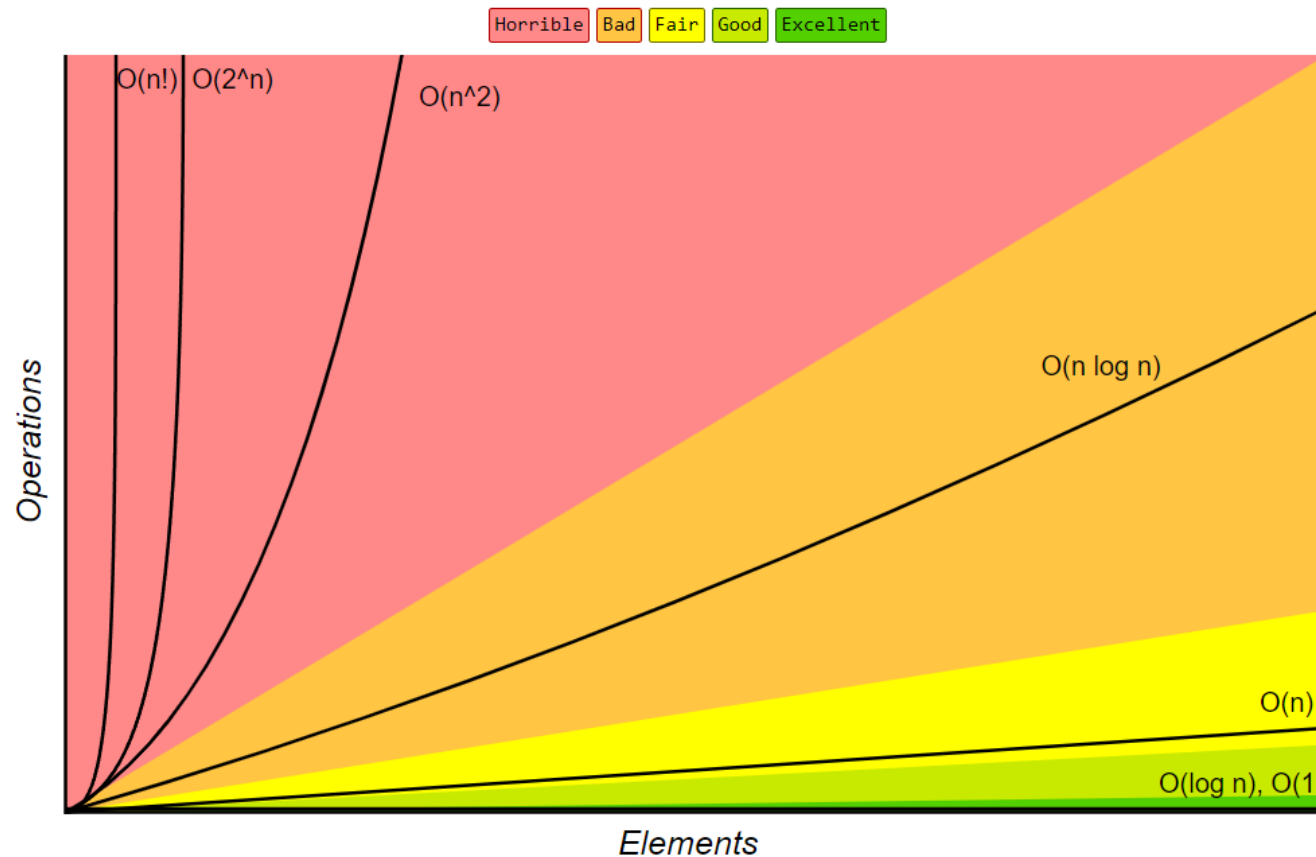
■ $O(N!)$

- 계승 시간(factorial time)
- 데이터 양의 증가에 따라 실행 시간이 팩토리얼 함수($N!$) 그래프로 나타남
- 평가: 노코멘트
- 알고리즘 예
 - 브루트포스 탐색을 통한 외판원 문제 해결방법

빅오 표기의 종류

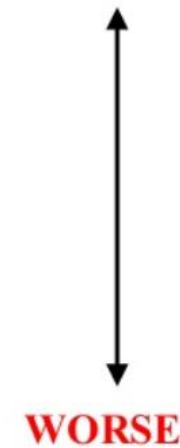
■ 알고리즘 성능 비교

• $O(1) > O(\log N) > O(N) > O(N \log N) > O(N^2) > \dots > O(2^N) > O(N!)$



Big-O: functions ranking

BETTER



WORSE

- $O(1)$ constant time
- $O(\log n)$ log time
- $O(n)$ linear time
- $O(n \log n)$ log linear time
- $O(n^2)$ quadratic time
- $O(n^3)$ cubic time
- $O(2^n)$ exponential time

시간제한 피하기

- 주어진 입력 N 의 크기에 따른 허용 시간 복잡도

N의 크기	시간복잡도
$N \leq 11$	$O(N!)$
$N \leq 25$	$O(2^N)$
$N \leq 100$	$O(N^4)$
$N \leq 500$	$O(N^3)$
$N \leq 3,000$	$O(N^2 \log N)$
$N \leq 5,000$	$O(N^2)$
$N \leq 1,000,000$	$O(N \log N)$
$N \leq 10,000,000$	$O(N)$
$N > 10,000,000$	$O(\log N), O(1)$

- 활용 방법

- 컴퓨터는 대략 1초에 1억회의 연산 수행한다고 가정하고 왼쪽 표를 얻어냄
- 시간 제한은 대부분 1~5초
- 입력 데이터가 5000개 이하로 주어진다면 $O(N^2)$ 또는 그보다 빠른 알고리즘을 설계하여 문제를 풀어야 함.
- 입력 데이터가 25개 이하로 주어진다면 $O(2^N)$ 알고리즘만 되어도 통과 가능할 것임.

알고리즘의 공간 복잡도

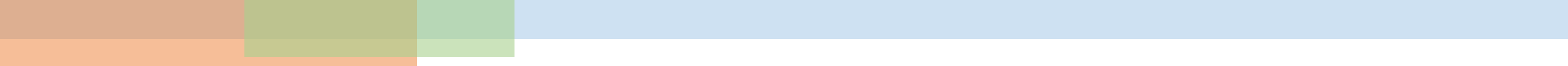
■ 공간 복잡도(Space Complexity)

란?

- 프로그램을 실행시킨 후 완료하는 데 필요로 하는 자원 공간의 양
- $S(P) = c + S_p(N)$
 - 총 공간 요구 = 고정 공간 요구 + 가변 공간 요구
 - 고정 공간: 입출력 횟수나 크기와 관계없는 공간 요구
 - 가변 공간: 문제 해결을 위해 필요한 공간 + 재귀 호출에 요구 되는 공간

■ 빅오 표기법

- 알고리즘의 공간복잡도 역시 빅오 표기법으로 표현 가능하며 계산법 역시 동일
- 단, 재귀 호출에 사용되는 스택 공간도 고려해야 함.
- 어떤 알고리즘이 N개의 입력 데이터에 대하여 $N \times N$ 크기의 2차원 배열과 N 크기의 1차원 배열이 필요하다면,
- 이때 이 알고리즘의 공간복잡도는 $O(N^2)$ 이다.



STL

(Standard Template Library)

STL

Standard Template Library

1. Stack (잘 안씀)
2. Vector(Stack의 상위 호환)
3. Queue
4. Deque (Double Ended Queue)

vector 컨테이너

■ vector의 특징

- 크기를 바꿀 수 있는 순차 컨테이너
- 어떤 자료형도 저장 가능
- 특정 위치의 원소에 빠르게 접근 가능
- 벡터에 원소가 삽입되고 삭제됨에 따라 자동으로 크기 조절됨
- `#include <vector>` 필요
- 원소의 추가/삭제
 - 벡터의 뒤쪽 끝에서만 삽입, 삭제 가능
 - 삽입: `push_back(x)`
 - 삭제: `push_pop();`

■ vector의 선언

```
vector<자료형> 변수명;  
vector<자료형> 변수명 = { 초기값 };
```

```
vector<int> v1;    // int를 담는 벡터  
vector<double> v2; //double을 담는 벡터
```

vector 컨테이너

■ vector의 사용 예시

```
#include <stdio.h>
#include <vector>
using namespace std;
```

```
int main() {
    vector<int> v = {2, 4, 5};
    v.push_back(6);
    v.pop_back();
    v[1] = 3;
```

```
    printf("%d\n", v[2]);
    for(int x: v) printf("%d ", x);
    v.reserve(8);
    v.resize(5, 0);
    printf("\n%d\n", v.capacity());
    printf("%d\n", v.size());
```

```
}
```

2	4	5
---	---	---

2	4	5	6
---	---	---	---

2	4	5	
---	---	---	--

2	3	5
---	---	---

prints 5

prints 2 3 5

2	3	5					
---	---	---	--	--	--	--	--

2	3	5	0	0	0		
---	---	---	---	---	---	--	--

prints 8

prints 5

// 벡터 v 선언 및 초기화

// 맨 마지막에 6 삽입

// 맨 마지막 원소 제거

// 1번 인덱스에 3 삽입

// 2번 인덱스 원소 출력

// 벡터 순회하면서 출력

// 벡터에 할당된 메모리 8칸으로 조정

// 벡터의 크기 5로 조절하고 빈 공간 0으로 채움

// 벡터가 차지하는 공간 출력

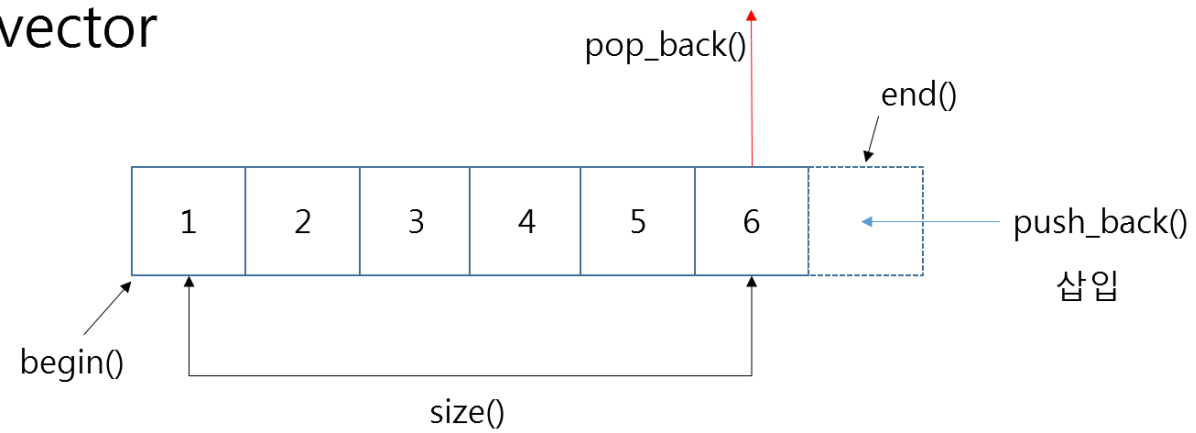
// 벡터의 크기 출력

vector 컨테이너

■ vector의 메소드

Member Function	Description
at(position)	at의 위치에 있는 값 리턴
capacity()	vector의 크기 리턴
clear()	Vector의 모든 값 삭제
empty()	Vector가 비었으면 true 리턴
pop_back()	Vector의 마지막 값 리턴
push_back(value)	Vector의 마지막에 값 저장
reverse()	Vector의 모든 값의 위치 반전
resize(n)	Vector의 크기 조정
resize(n, value)	Vector의 크기, 초기값 조정
size()	Vector의 수의 개수 리턴
swap(vector)	Vector의 내용 교환

vector



vector 컨테이너

■ 1차원 벡터의 순회

```
#include <stdio.h>
#include <vector>
using namespace std;

int main() {
    vector<int> v = { 6,2,9,7 };
    //방법1
    for(int i=0; i<v.size(); i++) {
        printf("%d ", v[i]);
    }
    puts("");
    //방법2
    for(int i : v) {
        printf("%d ", i);
    }
}
```

■ 2차원 벡터의 순회

```
#include <stdio.h>
#include <vector>
using namespace std;

int main() {
    vector<vector<int>> v =
        { {3, 1}, {2, 1, 5}, {6} };

    for(int i=0; i<v.size(); i++) {
        for(int j : v[i])
            printf("%d ", j);
        puts("");
    }
}
```

vector 컨테이너

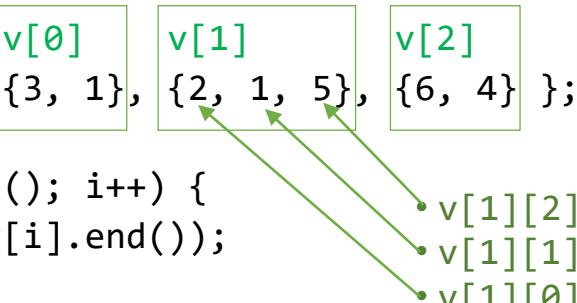
■ 2차원 벡터의 순회와 정렬

```
#include <stdio.h>
#include <vector>
#include <algorithm>

using namespace std;
int main(void) {
    // v[0] v[1] v[2]
    vector<vector<int>> v = { {3, 1}, {2, 1, 5}, {6, 4} };

    for (int i = 0; i < v.size(); i++) {
        sort(v[i].begin(), v[i].end());
    }

    for (int i = 0; i < v.size(); i++) {
        for (int j = 0; j < v[i].size(); j++) {
            printf("%d ", v[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```



D:\MyProjects\Test\TestWbin\Debug\Test.exe

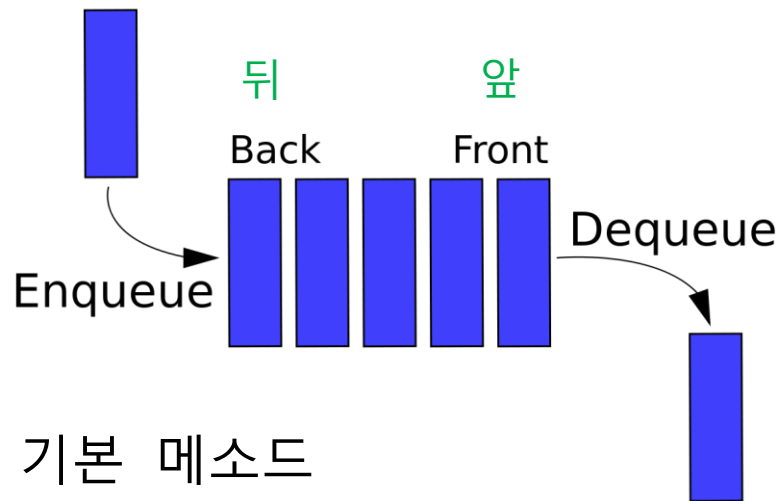
```
1 3
1 2 5
4 6
```

Process returned 0 (0x0) execution time : 0.040 s
Press any key to continue.

queue

■ Queue

- 대표적인 FIFO(First In First Out) 구조



- 기본 메소드
 - push, pop, empty, front, back

■ 선언과 사용 예

```
#include <stdio.h>
#include <queue>
using namespace std;

int main(void) {
    queue<int> q;
    q.push(1);    q.push(2);
    q.push(10);   q.push(20);

    while(!q.empty()) {
        printf("%d\n", q.front());
        q.pop();  // 원소 제거
    }
}
```

queue

■ Queue의 순회

```
#include <stdio.h>
#include <queue>
using namespace std;

// 큐의 다음 내용물을 보려면 pop()해야 하기 때문에
// 순회가 불가능하므로 인수로 사본을 받아 살펴본다
void output_Q(queue<int> Q) {
    printf("Q:");
    while(!Q.empty()) {
        printf("%2d ", Q.front());
        Q.pop();
    }
    printf("], ");
}
```

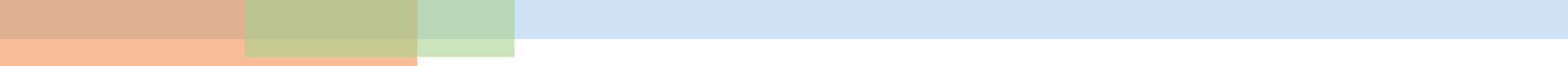
Q	1	3	5	7	9
---	---	---	---	---	---

```
int main(void) {
    queue<int> q;

    // Q에 1부터 10사이 홀수를 채운다.
    for(int i=1; i<10; i+=2) {
        q.push(i);
    }

    output_Q(q);
}
```

q	1	3	5	7	9
---	---	---	---	---	---



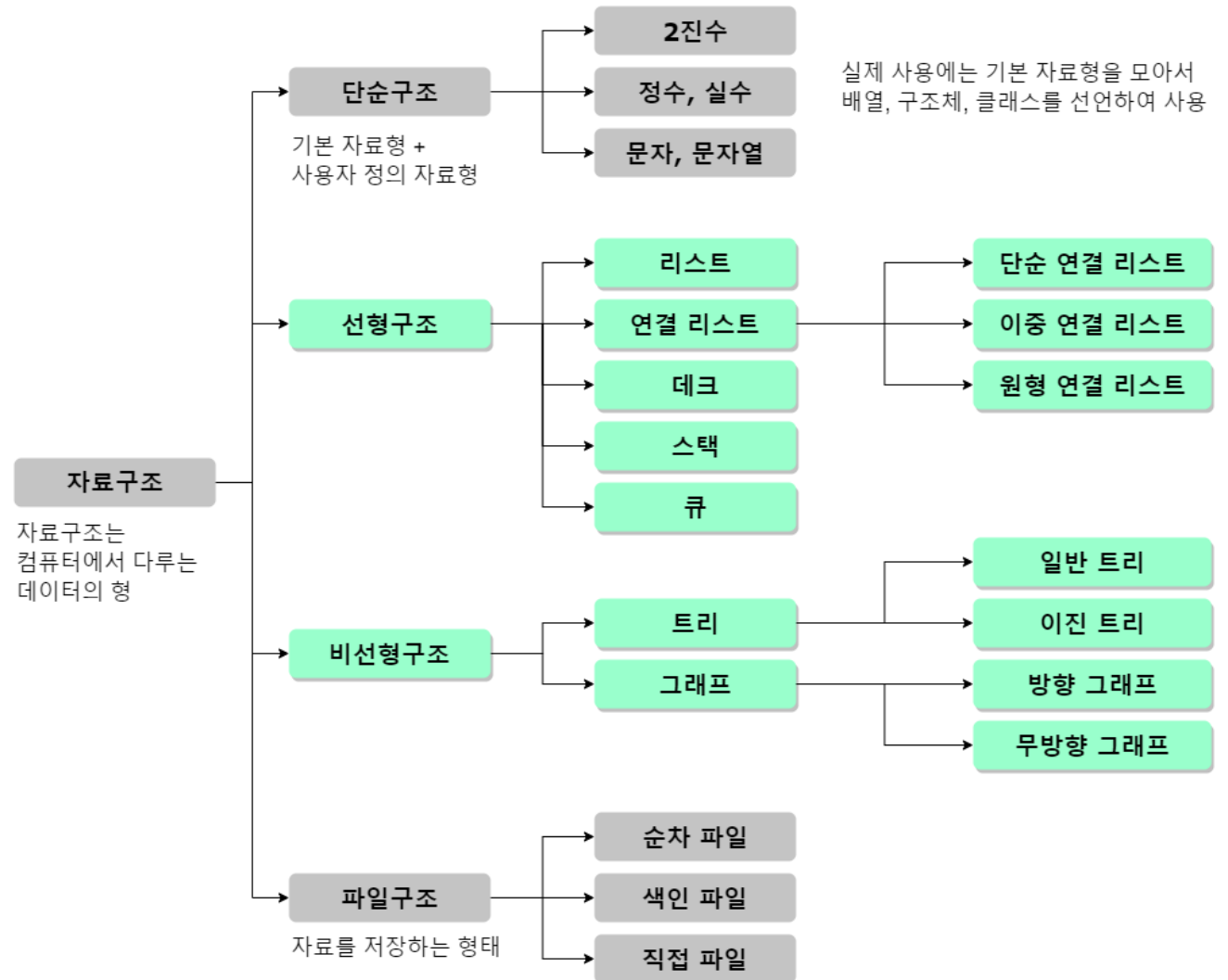
자료구조

선형구조와 비선형구조

자료구조

■ 자료구조(data structure)

- 전산학에서 자료를 효율적으로 이용할 수 있도록 컴퓨터에 저장하는 방법이다.
- 신중히 선택한 자료구조는 보다 효율적인 알고리즘을 사용할 수 있게 한다.



선형구조

■ 선형구조란?

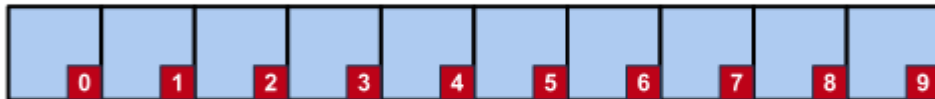
- 자료를 구성하는 데이터를 순차적으로 나열시킨 형태를 의미

■ 탐색법

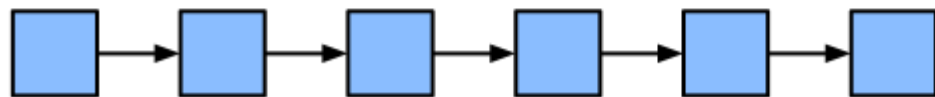
- 순차탐색
- 이분탐색

Array & Linked List

Access $A[k]$ in $O(1)$ time!

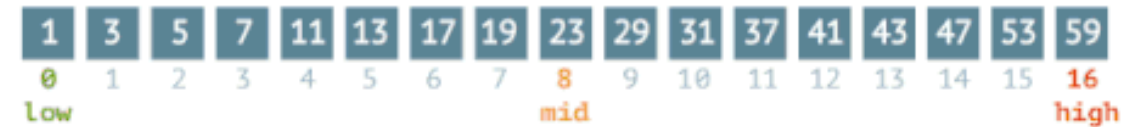


Access $L[k]$ in $O(n)$ time!



Binary search

steps: 0



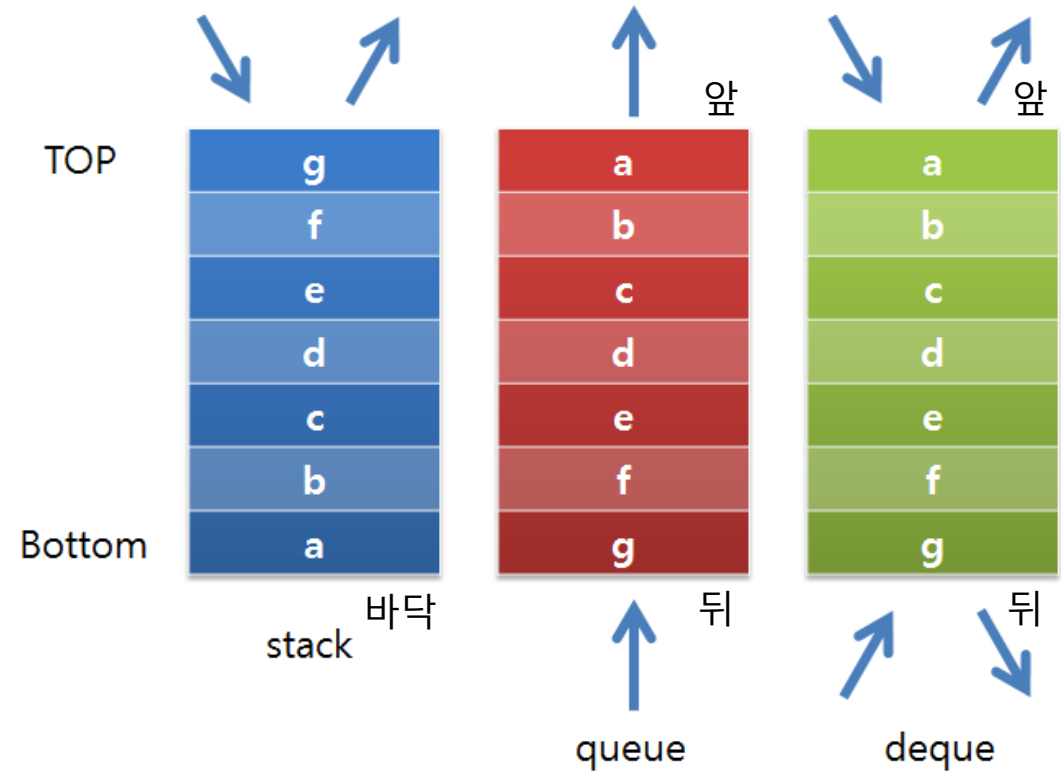
Sequential search

steps: 0



선형구조

- 배열
 - 고정 배열, 동적 배열(vector)
- 리스트
 - 연결리스트, 이중연결, 원형연결 리스트
- 스택
 - 후입선출(Last In First Out)
- 큐
 - 선입선출(First In First Out)
- 데크
 - Double Ended Queue

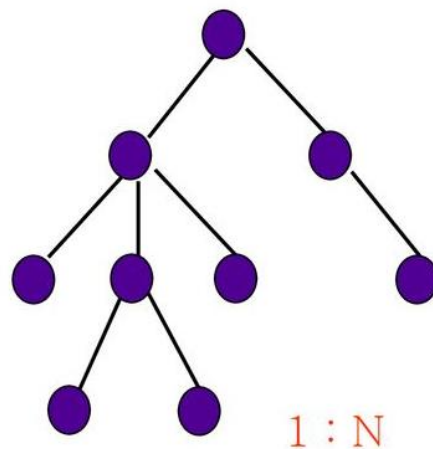


비선형구조

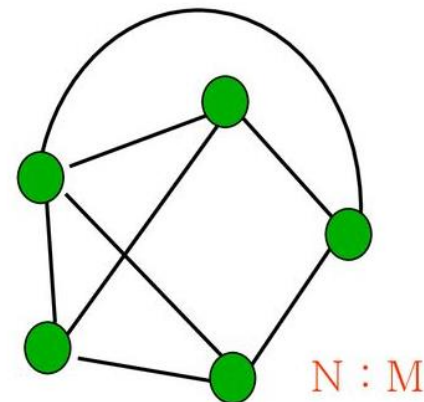
- 비선형구조란 i 번째 원소를 탐색한 다음 그 원소와 연결된 다른 원소를 탐색하려고 할 때, 여러 개의 원소가 존재하는 탐색구조
- 트리나 그래프로 구성된 경우
- 선형과 달리 자료가 순차적이지 않으므로 단순히 반복문을 이용하여 탐색하기 어려움
- 비선형구조는 스택이나 큐와 같은 자료구조를 활용하여 탐색
- 비선형구조 탐색법
 - 깊이우선탐색(DFS, depth first search)
 - 너비우선탐색(BFS, breadth first search)

비선형 구조

트리



그래프



- 그래프 중에 회로가 없는 그래프를 트리라고 한다.

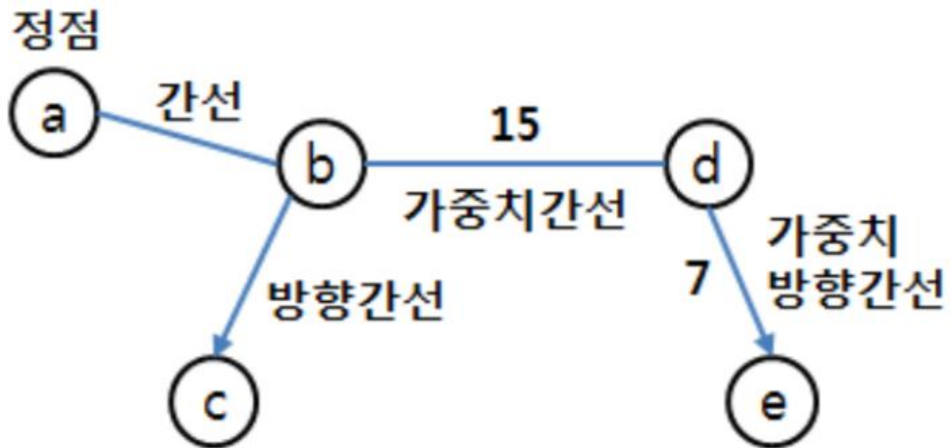
비선형구조 - 그래프

■ 정점(vertex)

- 노드(node)라 부르기도 한다

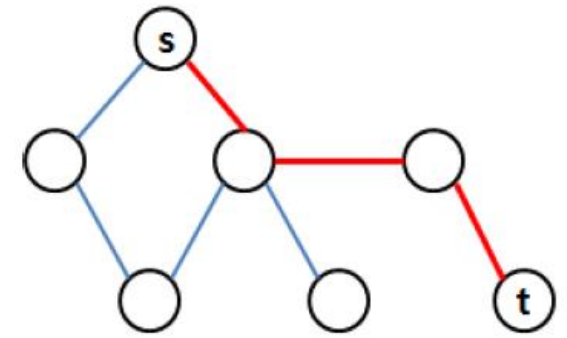
■ 간선(edge)

- 일반간선, 가중치 간선
- 방향간선, 양방향간선, 무방향간선



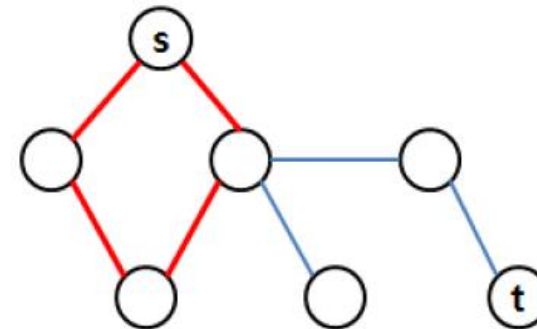
■ 경로(path)

- 임의의 정점 s 에서 임의의 정점 t 로 이동할 때, s 에서 t 로 이동하는데 사용한 정점들을 연결하고 있는 간선들의 순서로된 집합



■ 회로(cycle)

- 그래프에서 임의의 정점 s 에서 같은 정점 s 로의 경로들



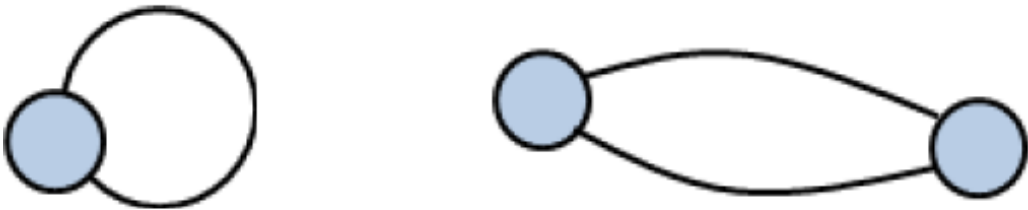
비선형구조 - 그래프

■ 자기간선(loop)

- 임의의 정점에서 자기 자신으로 연결하고 있는 간선

■ 다중간선(multi edge)

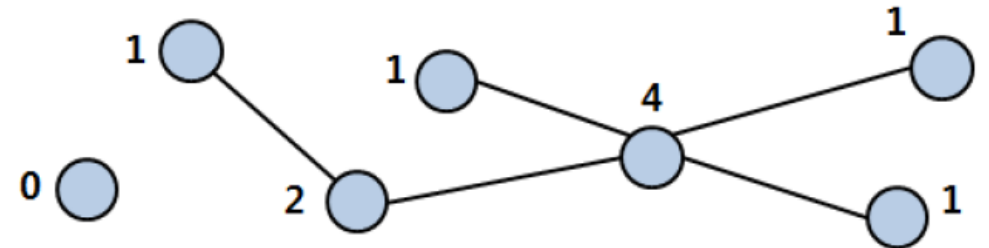
- 임의의 정점에서 다른 점점으로 연결된 간선의 수가 2개 이상일 경우



왼쪽은 자기간선 오른쪽은 다중간선을 나타낸다.

■ 그래프의 차수

- 그래프의 임의의 한 정점에서 다른 정점으로 연결된 간선의 수



각 정점에서의 차수

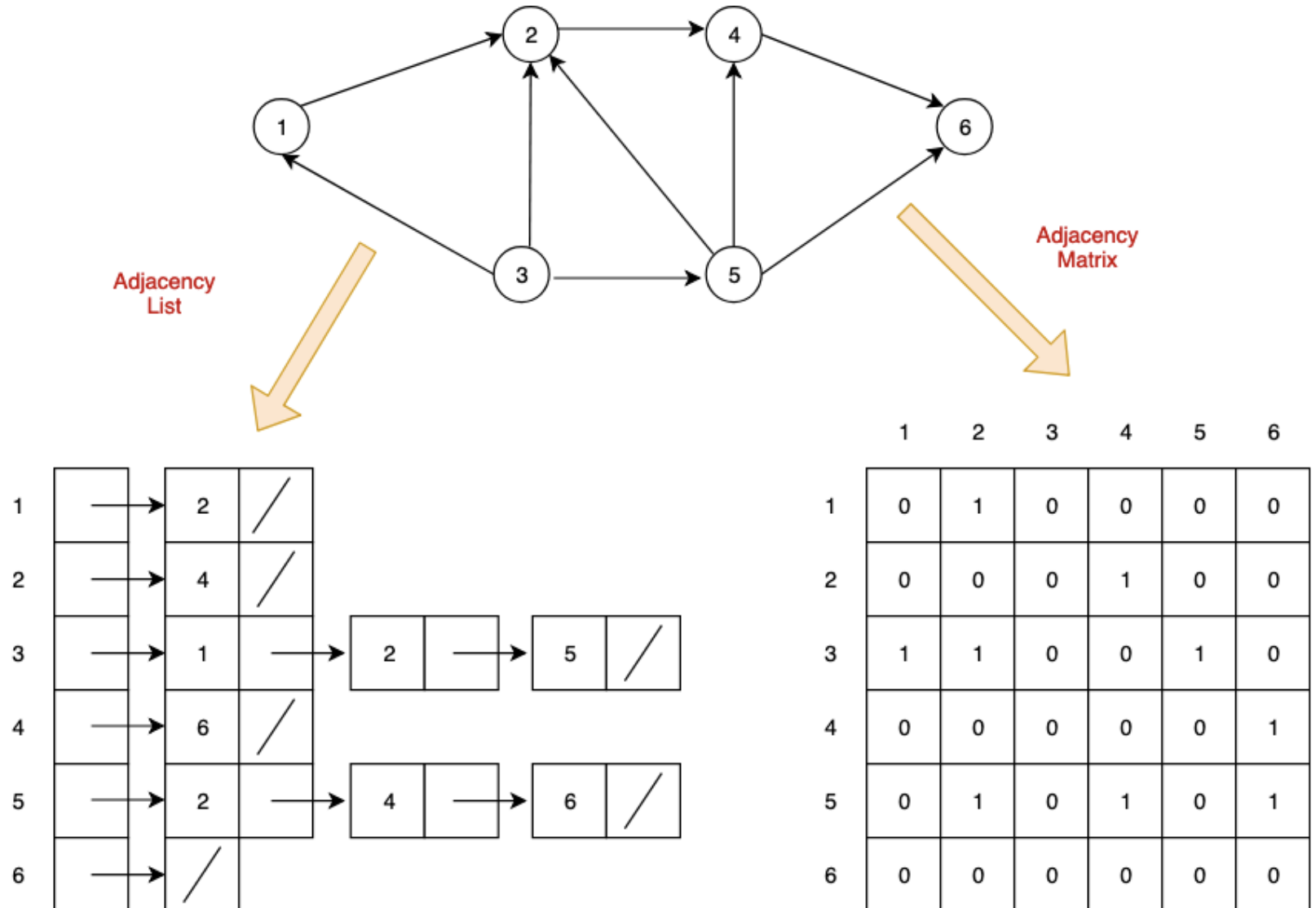
비선형구조 - 그래프

■ 그래프의 자료 구현

- 인접리스트
(adjacency list)

- 인접행렬
(adjacency matrix)

- 기타방법 ...



선형 탐색

feat. 탐색공간의 수학적 배제

탐색공간의 배제

■ 필요성

- 전체 탐색으로 대부분의 경우 해를 구할 수 있음
- 하지만 실행 시간이 너무 길어 제한 시간 내에 문제를 해결하기 힘든 경우가 많음
- 전체탐색에서 불필요한 탐색 공간을 탐색하지 않음으로써 알고리즘의 효율 향상 가능
- 모든 공간을 탐색할 것이 아니라 일정한 조건을 두어 탐색에서 제외

■ 수학적 배제

- 수학적으로 탐색할 필요가 없음이 증명된 공간을 탐색에서 제외

■ 경험적 배제(가지치기)

- 일정 조건을 만족하는 경우 탐색에서 배제하는데 이 조건은
- 이전에 탐색한 정보를 이용하며,
- 배제 조건은 계속 갱신될 수 있음

약수의 합

■ 문제

한 정수 n 을 입력 받는다.

1부터 n 의 자연수들 중 n 약수의 합을 구하는 프로그램을 작성하시오.

예를 들어 n 이 10이라면,

10의 약수는 1, 2, 5, 10이므로 구하고자 하는 값은 $1 + 2 + 5 + 10$ 을 더한 18이 된다.

■ 입력

첫 번째 줄에 정수 n 이 입력된다.

(단, $1 \leq n \leq 10,000,000,000(100억)$)

■ 출력

n 의 약수의 합을 출력한다.

입력 예	출력 예
10	18

탐색공간이 매우
넓기 때문에
일반적인 방법으로는
시간 제한에 걸리게
된다.

약수의 합

■ 단순 풀이

```
#include <stdio.h>
long long n;
long long solve() {
    long long ans=0;

    for(long long i=1; i<=n; i++) {
        //n이 i로 나누어 떨어지면 i는 n의 약수이다
        if(n%i==0)
            ans+=i;
    }
    return ans;
}

int main() {
    scanf("%lld", &n);
    printf("%lld\n", solve());
    return 0;
}
```

■ 평가

- 이 소스코드는 1부터 n까지의 모든 원소들을 탐색하여, 탐색 대상인 수 i가 n의 약수라면 취하는 방식으로 진행된다.
- 따라서 계산량은 $O(n)$ 이다.
- 이번 문제는 n의 최댓값이 100억이므로 이 방법으로는 너무 많은 시간이 걸린다.
- 따라서 탐색영역을 배제해야 할 필요가 있다.

약수의 합

■ 고찰

1) 배제를 위한 수학적 아이디어 1

모든 자연수 n 에 대하여 1 과 n 은 항상 n 의 약수이다.

ex) 10의 약수

1, 2, 5, 10

ex) 16의 약수

1, 2, 4, 8, 16

```
for(int i=2; i<n; i++) {  
    if(n % i==0)  
        ans+=i;  
}
```

■ 고찰

2) 배제를 위한 수학적 아이디어 2

모든 자연수 n 에 대하여,
2이상 n 미만의 자연수들 중 가장 큰
 n 의 약수는 $n/2$ 를 넘지 않는다.

ex) 10의 약수

1, 2, 5, 10

ex) 16의 약수

1, 2, 4, 8, 16

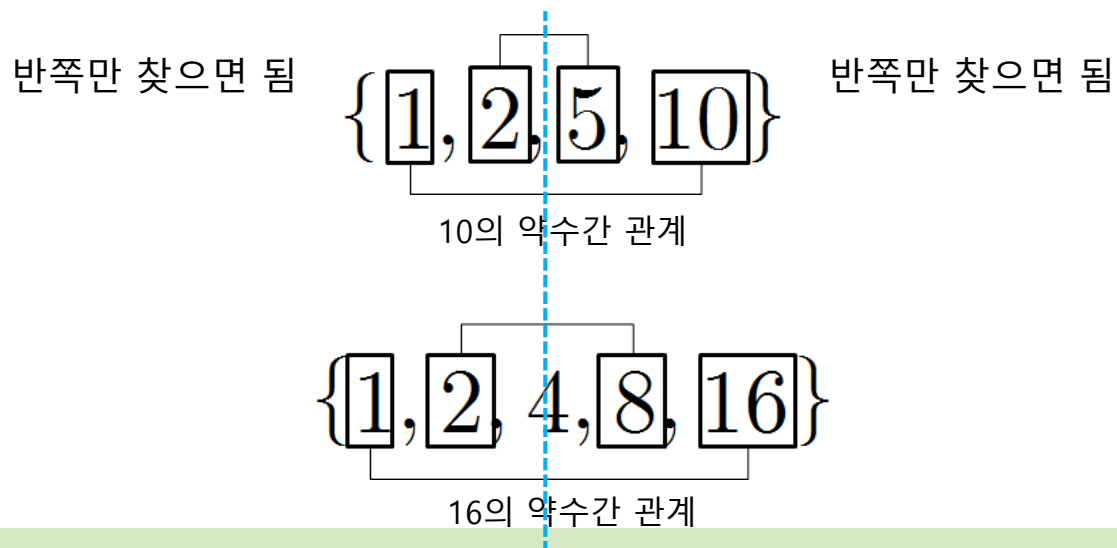
```
for(int i=2; i<=n/2; i++) {  
    if(n % i==0)  
        ans+=i;  
}
```

약수의 합

■ 고찰

3) 배제를 위한 수학적 아이디어 3

임의의 자연수 n 의 약수들 중 두 약수의 곱은, n 이 되는 약수 a 와 약수 b 가 반드시 존재한다. 단, n 이 완전제곱수 일 경우에는 약수 a 와 약수 b 가 같을 수 있다.



약수의 개수를 c 개라고 하고, d 를 n 의 약수 중 i 번째 약수라 하면,

$$n = \underline{d_k} \times \underline{d_{c-k+1}}$$

완전 제곱수일 경우 우변 두 항이 동일, 최악의 경우 n 부터 $\sqrt{n}$ 까지만 검색하면 나머지 약수를 모두 찾아낼 수 있음

```
#include <math.h>
```

```
:
```

```
for(int i=1; i<=sqrt(n); i++) {  
    if(n % i==0)  
        ans+=i;  
}
```

```
for(int i=1; i*i<=n; i++) {  
    if(n % i==0)  
        ans+=i;  
}
```

약수의 합

■ 고찰

3) 배제를 위한 수학적 아이디어 3 구현

ex) 100의 약수 모두 구하기

① $1 \sim \sqrt{100} = 10$ 까지 조사

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

② 약수 집합a { 1, 2, 5, 10}으로부터

약수 집합b {100, 50, 20, 10}유도

③ 약수 합집합 {1,2,5,10,20,50,100}

■ 수학적 배제 적용

```
#include <stdio.h>
long long int n;
long long int solve() {
    long long int i, ans = 0;

    return ans;
}
int main() {
    scanf("%lld", &n);
    printf("%lld\n", solve());
    return 0;
}
```

N번째 소수 찾기

■ 문제

한 정수 n 을 입력 받는다.

n 번째로 큰 소수를 구하여 출력한다.

예를 들어 n 이 5라면,

자연수들 중 소수는 2, 3, 5, 7, 11, 13, ...

이므로 구하고자 하는 5번째 소수는 11이 된다.

■ 입력

첫 번째 줄에 정수 n 이 입력된다.

(단, $1 \leq n \leq 100,000$)

■ 출력

n 번째 소수를 출력한다.

입력 예	출력 예
5	11

입력 예	출력 예
77	389

N번째 소수 찾기

■ 단순 풀이

```
#include <stdio.h>

bool is_prime(int k) {
    int cnt = 0;
    for(int i=1; i<=k; i++)
        if(k%i==0) // k의 약수의 갯수 cnt를 구한다
            cnt++;
```

임의의 자연수 k가 소수라면 k의 약수는 1과 k만 존재한다. (약수가 2개 뿐임)

```
    if(cnt==2)
        return true;
    else
        return false;
}
```

```
int main() {
    int nth; // 몇 번째
    scanf("%d", &nth);

    int prime_cnt=0;
    int n=2;
    while(true) {
        if(is_prime(n)) //소수이면 cnt증가
            prime_cnt++;
        // 찾고자 하는 번째 이면
        if(prime_cnt == nth)
            break;
        n++;
    }
    printf("%d", n);
}
```


N번째 소수 찾기

■ 고찰

1) 배제를 위한 수학적 아이디어 1

약수를 두 개 이상 발견하면 바로 탈출

```
bool is_prime(int k) {
    int cnt = 0;
    for(int i=1; i<=k; i++) {
        if(k%i==0) cnt++;
        if(cnt > 2)
            break;
    }
    return (cnt==2);
}
```

2) 배제를 위한 수학적 아이디어 2

임의의 자연수 k 가 소수라면 구간 $[2, k-1]$ 에서 약수는 존재하지 않는다.

```
bool is_prime(int k) {
    for(int i=2; i<k; i++) {
        //2~ k-1사이 숫자로 나누어 떨어지면,
        // 즉, 약수가 존재하면
        if(k%i==0)
            return false;
    }
    return true;
}
```

N번째 소수 찾기

■ 고찰

3) 배제를 위한 수학적 아이디어 3

k의 약수를 구하기 위해서는 ____ 까지만 검사하면 된다.

```
bool is_prime(int k) {  
    for(int i=2; ?; i++) {  
        if(k%i==0)  
            return false;  
    }  
    return true;  
}
```

```
#include <stdio.h>
```

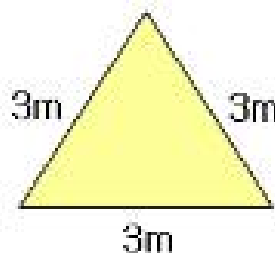
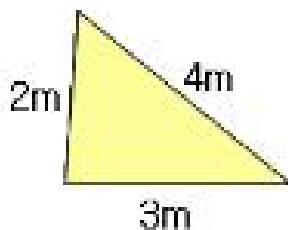
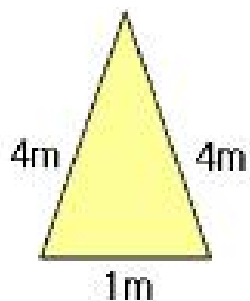
```
bool is_prime(int k) {  
    for(int i=2; i*i<=k; i++) {  
        if(k%i==0)  
            return false;  
    }  
    return true;  
}
```

```
int main() {  
    int nth;  
    scanf("%d", &nth);  
  
    int prime_cnt=0;  
    int n=2;  
    while(true) {  
        if(is_prime(n))  
            prime_cnt++;  
        if(prime_cnt == nth)  
            break;  
        n++;  
    }  
    printf("%d\n\n", n);  
}
```

삼각화단 만들기

주어진 화단 둘레의 길이를 이용하여 삼각형 모양의 화단을 만들려고 한다. 이 때 만들어진 삼각형 화단 둘레의 길이는 반드시 주어진 화단 둘레의 길이와 같아야 한다. 또한, 화단 둘레의 길이와 각 변의 길이는 자연수이다. 예를 들어, 만들고자 하는 화단 둘레의 길이가 9m라고 하면,

- 한 변의 길이가 1m, 두 변의 길이가 4m인 화단
- 한 변의 길이가 2m, 다른 변의 길이가 3m, 나머지 변의 길이가 4m인 화단
- 세 변의 길이가 모두 3m인 3가지 경우의 화단을 만들 수 있다.



화단 둘레의 길이를 입력 받아서 만들 수 있는 서로 다른 화단의 수를 구하는 프로그램을 작성하시오.

- **입력**

화단의 길이 n 이 주어진다. ($1 \leq n \leq 50,000$)

- **출력**

입력받은 n 으로 만들 수 있는 서로 다른 화단의 수를 출력한다.

입력 예	출력 예
9	3

- **주의**

2,3,4 화단과 3,2,4 화단 2,4,3 화단은 모두 같은 모양의 화단임.

삼각화단 만들기

■ 단순 풀이 (오답)

```
#include <stdio.h>

int main(void) {
    int n;
    int cnt=0;
    scanf("%d", &n);
    for(int a=1; a<=n; a++)
        for(int b=1; b<=n; b++)
            for(int c=1; c<=n; c++) {
                if(a+b+c==n) {
                    printf("[%d %d %d]\t", a, b, c);
                    cnt++;
                    // 5개 출력할 때마다 줄 내림
                    if(cnt%5 == 0) puts("");
                }
            }
    printf("\nfound %d\n", cnt);
}
```

■ 평가

- $O(n^3)$ 의 시간 소모
- 중복된 삼각형이 포함됨
- 삼각형 만들기가 불가능한 길이기도 포함

9

[1 1 7]	[1 2 6]	[1 3 5]	[1 4 4]
[1 5 3]	[1 6 2]	[1 7 1]	[2 1 6]
[2 2 5]	[2 3 4]	[2 4 3]	[2 5 2]
[2 6 1]	[3 1 5]	[3 2 4]	[3 3 3]
[3 4 2]	[3 5 1]	[4 1 4]	[4 2 3]
[4 3 2]	[4 4 1]	[5 1 3]	[5 2 2]
[5 3 1]	[6 1 2]	[6 2 1]	[7 1 1]

found 28

삼각화단 만들기

■ 풀이

- 동일한 길이 쌍 제거 조건

$$a \leq b \leq c$$

- 삼각형의 조건

- $a + b > c$

- $a + b + c = n$

■ 정답 알고리즘

```
#include <stdio.h>
int n;
int solve() {
    int cnt = 0;
    scanf("%d", &n);

    for(int a=1; a<=n; a++)
        for(int b=a; b<=n; b++)
            for(int c=b; c<=n; c++ )
                if(a+b+c==n && a+b>c)
                    cnt++;

    return cnt;
}

int main() {
    printf("%d\n", solve());
}
```

삼각화단 만들기

■ 고찰

1) 배제를 위한 수학적 아이디어 1

둘레 길이가 n 인 삼각형의 a , b 길이가 정해지면 c 변은 $n-(a+b)$ 계산으로 구할 수 있다.

```
for(int a=1; a<=n; a++)
    for(int b=a; b<=n; b++) {
        int c=n-(a+b);    // a+b+c=n 조건만족
        if(b<=c && a+b>c) // a<=b는 이미 만족
            cnt++;
    }
```

- 공간복잡도가 $O(n^3)$ 에서 $O(n^2)$ 으로 줄어든다.

2) 배제를 위한 수학적 아이디어 2

둘레가 n 인 삼각형의 각 변의 길이를 오름차순으로 정렬한 결과를 a , b , c 라고 할 때, 다음 조건을 만족한다.

$$\left\lceil \frac{n}{3} \right\rceil \leq c < \left\lfloor \frac{n}{2} \right\rfloor, 1 \leq a \leq \left\lfloor \frac{n}{3} \right\rfloor$$

```
for(int c=n/3; c<=n/2; c++)
    for(int a=1; a<=n/3; a++) {
        int b=n-(a+c);
        if(a+b>c && (a<=b && b<=c))
            cnt++;
    }
```

삼각화단 만들기

■ 실행시간 비교

1) 배제를 위한 수학적 아이디어 1

```
time_space_table:
/1030/sample.in:AC mem=0k time=3ms
/1030/test01.in:AC mem=0k time=2ms
/1030/test02.in:AC mem=0k time=2ms
/1030/test03.in:AC mem=0k time=3ms
/1030/test04.in:AC mem=0k time=3ms
/1030/test05.in:AC mem=0k time=11ms
/1030/test06.in:AC mem=0k time=38ms
/1030/test07.in:AC mem=0k time=149ms
/1030/test08.in:AC mem=0k time=355ms
/1030/test09.in:AC mem=0k time=586ms
/1030/test10.in:AC mem=0k time=925ms
```

2) 배제를 위한 수학적 아이디어 2

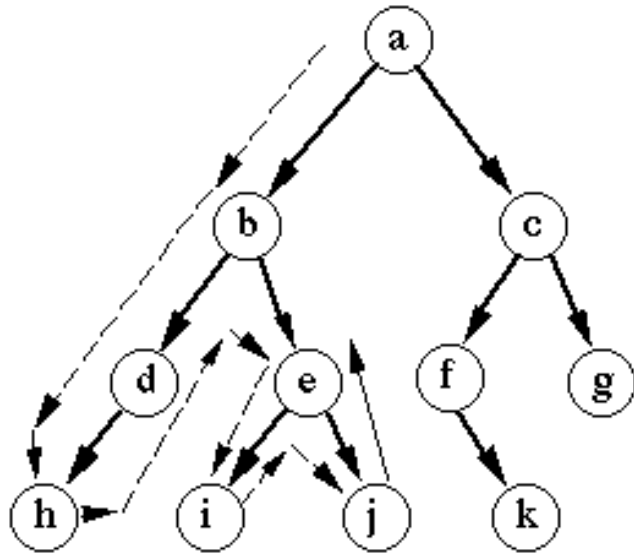
```
time_space_table:
/1030/sample.in:AC mem=0k time=2ms
/1030/test01.in:AC mem=0k time=2ms
/1030/test02.in:AC mem=0k time=2ms
/1030/test03.in:AC mem=0k time=3ms
/1030/test04.in:AC mem=0k time=3ms
/1030/test05.in:AC mem=0k time=4ms
/1030/test06.in:AC mem=0k time=7ms
/1030/test07.in:AC mem=0k time=19ms
/1030/test08.in:AC mem=0k time=40ms
/1030/test09.in:AC mem=0k time=68ms
/1030/test10.in:AC mem=0k time=107ms
```

비선형 탐색

비선형구조의 전체탐색(DFS vs BFS)

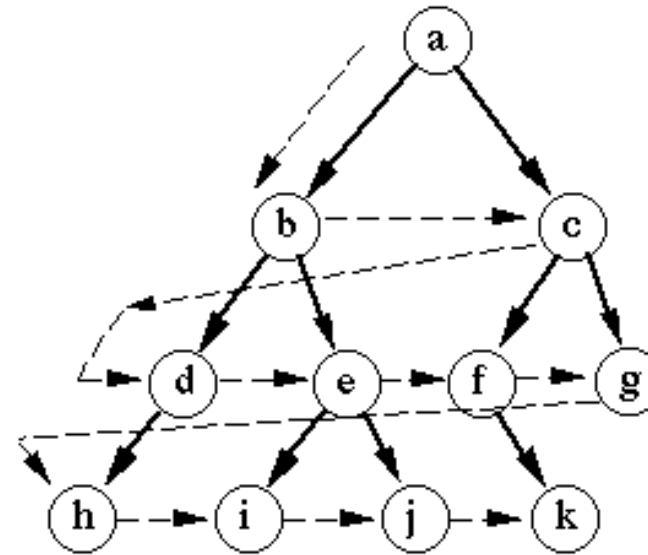
탐색

■ 깊이우선탐색(DFS)



Depth-first search

■ 너비우선탐색(BFS)



Breadth-first search

DFS(깊이우선탐색)	BFS(너비우선탐색)
현재 정점에서 갈 수 있는 점들까지 들어가면서 탐색	현재 정점에 연결된 가까운 점들부터 탐색
스택 또는 재귀함수로 구현	큐를 이용해서 구현

탐색

■ 깊이우선탐색(DFS)

- 최대한 깊이 내려간 뒤, 더이상 깊이 갈 곳이 없을 경우 옆으로 이동
- 루트 노드(혹은 다른 임의의 노드)에서 시작해서 다음 분기(branch)로 넘어가기 전에 해당 분기를 완벽하게 탐색하는 방식

■ 평가

1. 깊이 우선 탐색(DFS)이 너비 우선 탐색(BFS)보다 좀 더 간단함
2. 검색 속도 자체는 너비 우선 탐색(BFS)에 비해서 느림

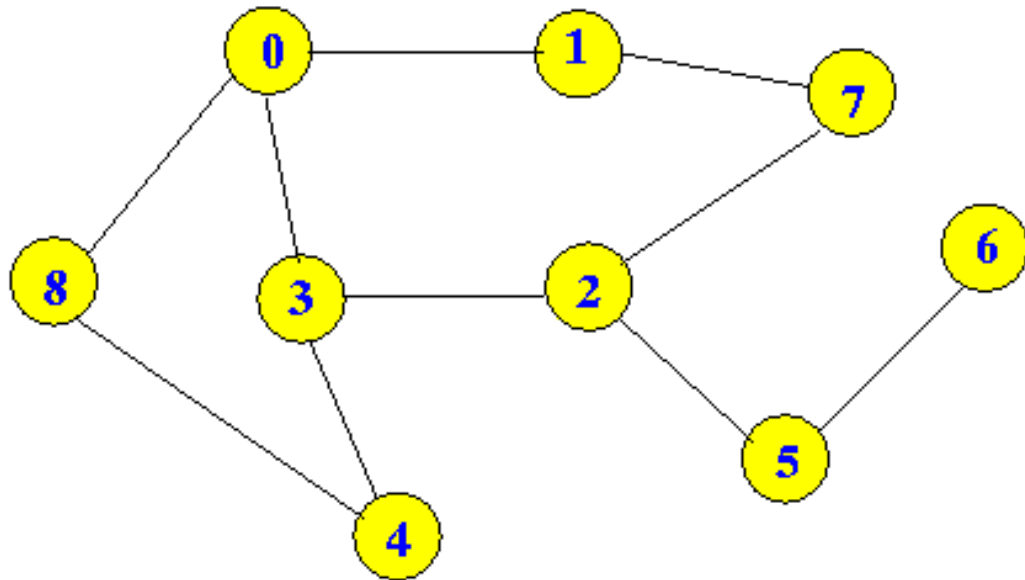
■ 너비우선탐색(BFS)

- 최대한 넓게 이동한 다음, 더 이상 갈 수 없을 때 아래로 이동
- 루트 노드(혹은 다른 임의의 노드)에서 시작해서 인접한 노드를 먼저 탐색하는 방법
- 시작 정점으로부터 가까운 정점을 먼저 방문하고 멀리 떨어져 있는 정점을 나중에 방문하는 순회 방법

깊이우선탐색(DFS)

■ 그래프의 순회

트리와 달리 그래프는 사이클이 존재함
방문정보를 유지하여 재방문을 막아야 함



■ 깊이우선탐색 알고리즘

: go deep(before going wide)

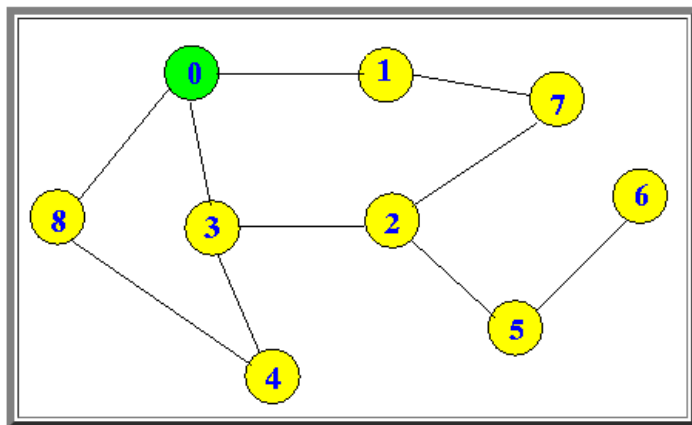
```
def dfs(k):
```

- 1) 정점 k를 처리하고 방문한 것으로 표시
- 2) 정점 k와 연결된 모든 정점에 대하여 방문한적이 없으면 그 정점에서 dfs, 완료되면 되돌아오기 (백트랙)

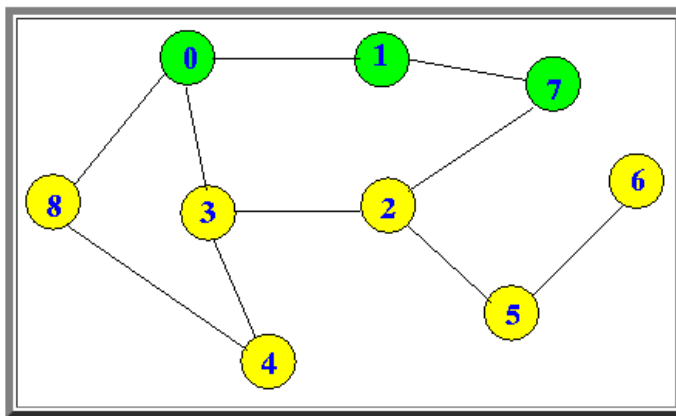
깊이우선탐색(DFS)

■ 탐색순서

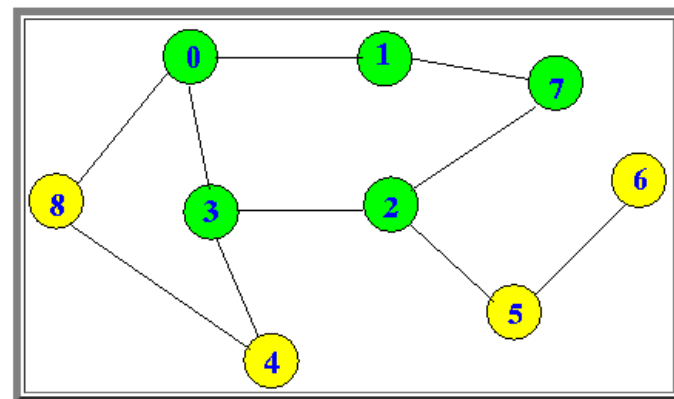
① dfs(0):



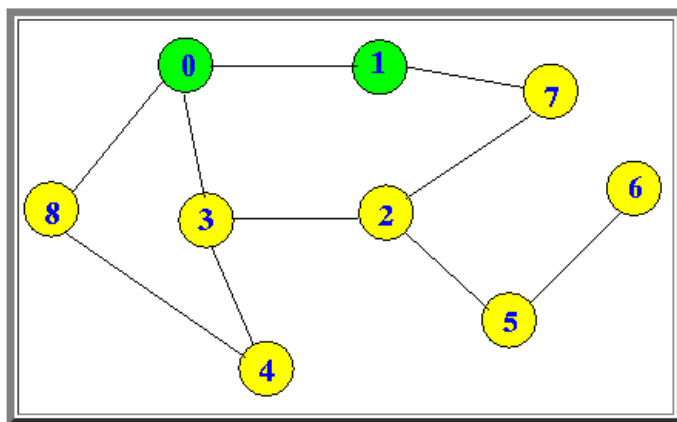
■ dfs(1) → dfs(0) (because node 0 is "visited"); dfs(1) → dfs(7)



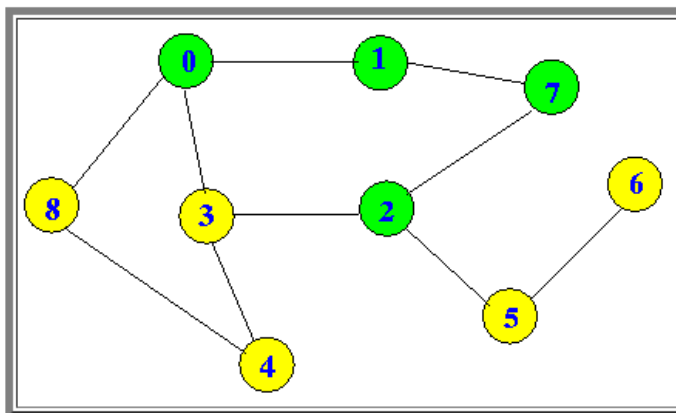
■ dfs(2) → dfs(3)



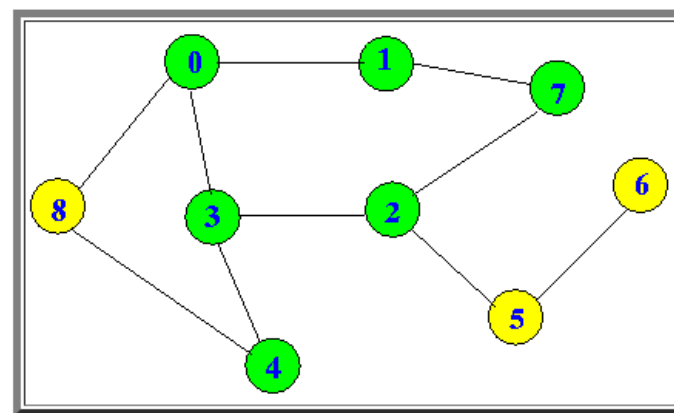
② dfs(0) → dfs(1)



■ dfs(7) → dfs(1); dfs(7) → dfs(2)



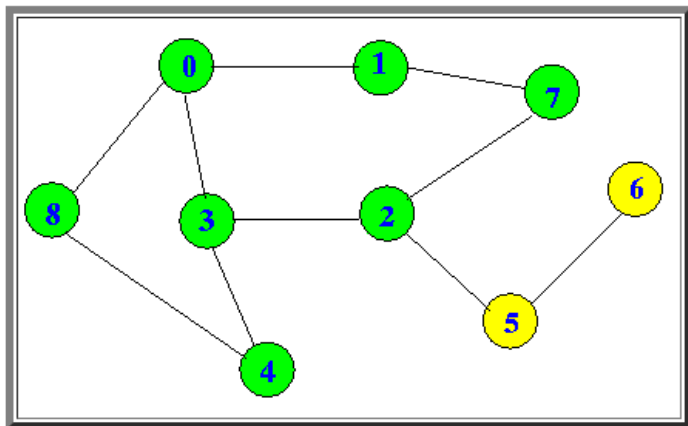
■ dfs(3) → dfs(0); dfs(3) → dfs(2); dfs(3) → dfs(4)



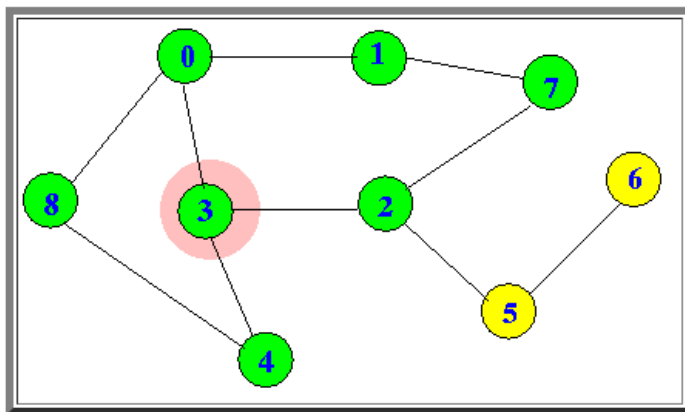
깊이우선탐색(DFS)

■ 탐색순서

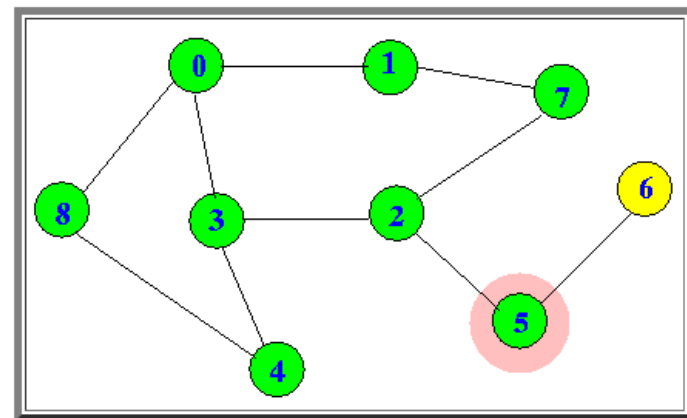
- $\text{dfs}(4) \rightarrow \text{dfs}(3)$; $\text{dfs}(4) \rightarrow \text{dfs}(8)$



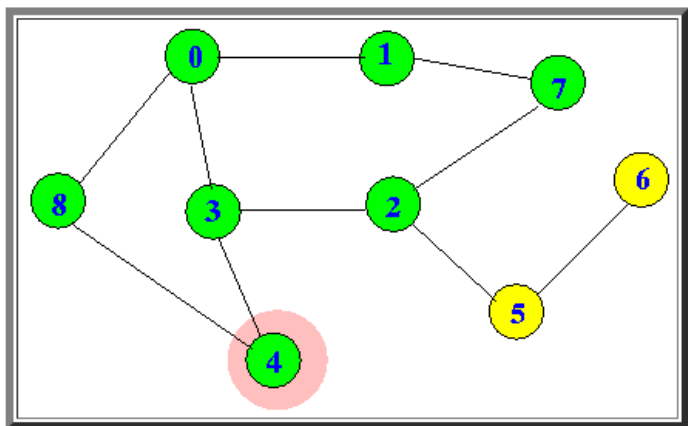
- return to $\text{dfs}(3)$



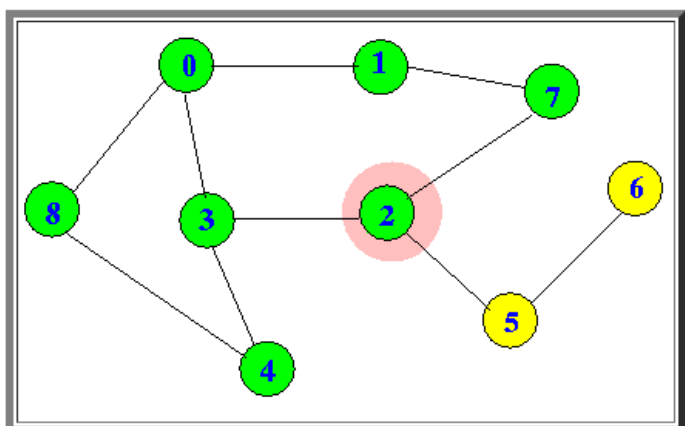
- $\text{dfs}(2) \rightarrow \text{dfs}(3)$; $\text{dfs}(2) \rightarrow \text{dfs}(5)$



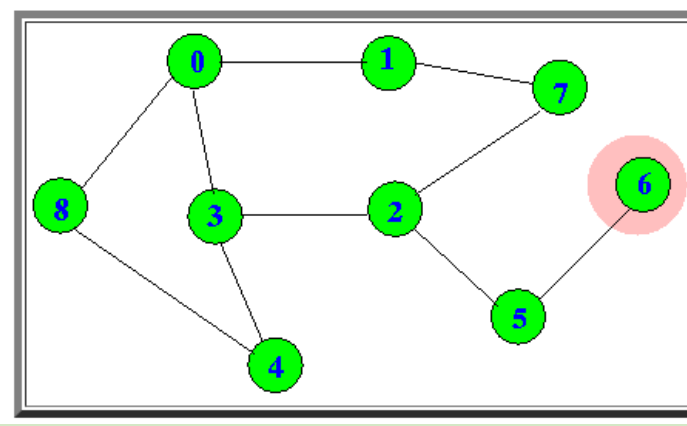
- $\text{dfs}(8) \rightarrow \text{dfs}(0)$; $\text{dfs}(8) \rightarrow \text{dfs}(4)$; return to $\text{dfs}(4)$



- return to $\text{dfs}(2)$



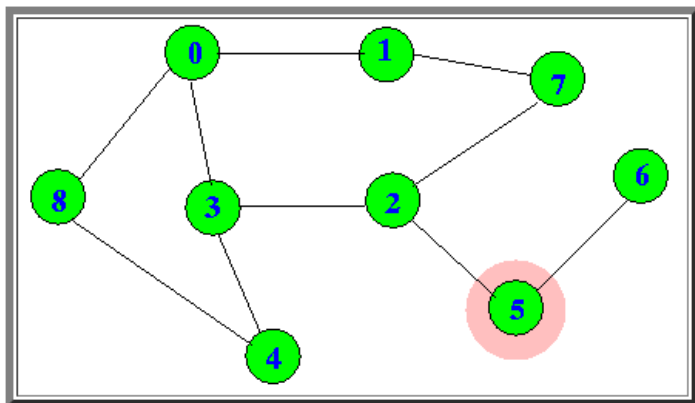
- $\text{dfs}(5) \rightarrow \text{dfs}(2)$; $\text{dfs}(5) \rightarrow \text{dfs}(6)$



깊이우선탐색(DFS)

■ 탐색순서

- $\text{dfs}(6) \rightarrow \text{dfs}(5)$; return to $\text{dfs}(5)$



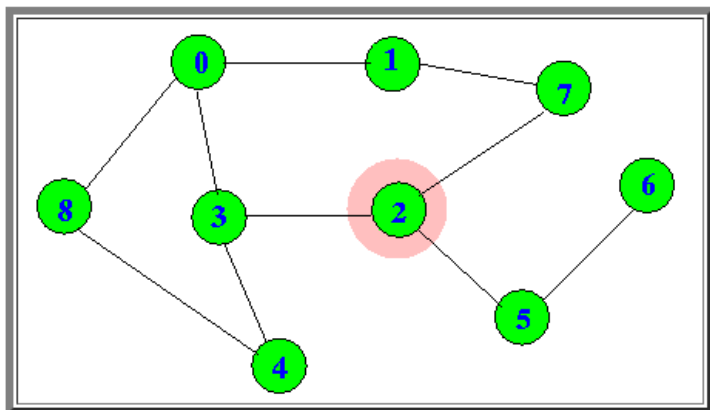
- return to $\text{dfs}(7)$

- return to $\text{dfs}(1)$

- return to $\text{dfs}(0)$

DONE

- return to $\text{dfs}(2)$



깊이우선탐색(DFS)

■ vector를 인접리스트로 활용

그래프	데이터
	8 10
	0 1
	0 3
	0 8
	1 7
	2 3
	2 5
	3 4
	2 7
	4 8
	5 6

```

int n, m;
vector<vector<int>> G;
vector<bool> visited;

void input_G() {
    scanf("%d %d", &n, &m);
    // 정점이 0부터 시작하므로 n+1
    G.resize(n+1); // 공간확보
    visited.assign(n+1, 0);

    for(int i=0; i<m; i++) {
        int a, b;
        scanf("%d %d", &a, &b);
        // 양방향 연결이므로 a->b, a<-b
        G[a].push_back(b);
        G[b].push_back(a);
    }
}

int main() {
    input_G();
}

```

```

idx  [0][1][2]
G[0]: 1  3  8
G[1]: 0  7
G[2]: 3  5  7
G[3]: 0  2  4
G[4]: 3  8
G[5]: 2  6
G[6]: 5
G[7]: 1  2
G[8]: 0  4

visited[0]: 0
visited[1]: 0
visited[2]: 0
visited[3]: 0
visited[4]: 0
visited[5]: 0
visited[6]: 0
visited[7]: 0
visited[8]: 0

```

깊이우선탐색(DFS)

■ DFS 알고리즘 구현 (인접리스트)

- DFS 알고리즘

def dfs(k):

- 1) 정점 k를 처리하고 방문한 것으로 표시
- 2) 정점 k와 연결된 모든 정점에 대하여 방문한 적이 없으면 그 정점에서 dfs, dfs 완료되면 되돌아오기(백트랙)

dfs함수가 종료되면
호출 위치로 알아서
되돌아오므로 딱히
백트랙을 구현할
필요는 없음.

```
int n, m;
vector<vector<int>> G;
vector<bool> visited;
```

// 정점 k에서 dfs

```
void dfs(int k) {
```

// 정점 k를 방문하였음을 출력

```
printf("dfs(%d) started.\n", k);
```

// 나중에 다시오지 않기 위해 방문을 표시하고,
?

// 정점 k와 연결된 모든 정점에 대하여

```
for (int i=0; i<G[k].size(); i++) {
```

// k와 연결된 i번째 정점에 방문한 적 없으면,

```
if ( ? ) {
```

// 그 정점에서 다시 dfs 시작

```
?
```

// dfs 완료하고 돌아왔다고 메시지 출력

```
printf("return to dfs(%d).\n", k);
```

```
}
```

```
}
```

return; // 연결된 모든 정점을 방문완료하여 되돌아가기

```
}
```

idx	[0]	[1]	[2]
G[0]:	1	3	8
G[1]:	0	7	
G[2]:	3	5	7
G[3]:	0	2	4
G[4]:	3	8	
G[5]:	2	6	
G[6]:	5		
G[7]:	1	2	
G[8]:	0	4	

visited[0]:	0
visited[1]:	0
visited[2]:	0
visited[3]:	0
visited[4]:	0
visited[5]:	0
visited[6]:	0
visited[7]:	0
visited[8]:	0


```
#include <stdio.h>
#include <vector>
using namespace std;
int n, m;
vector<vector<int>> G;
vector<bool> visited;

void input_G() {
    scanf("%d %d", &n, &m);
    G.resize(n+1); // 정점이 0부터 시작하므로 n+1
    visited.assign(n+1, 0);
    for(int i=0; i<m; i++) {
        int a, b;
        scanf("%d %d", &a, &b);
        G[a].push_back(b);
        G[b].push_back(a);
    }
}

void output_G() {
    printf("\n");
    for(int a=0; a<G.size(); a++) {
        printf("%2d:", a);
        for(int i : G[a])
            printf("%3d", i);
        printf("\n");
    }
    printf("\n");
}
```

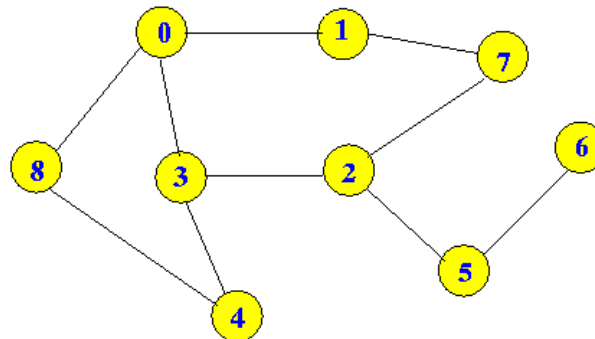
```
// 정점 k에서 dfs
void dfs(int k) {
    // 정점 k를 방문하였음을 출력
    printf("dfs(%d) started.\n", k);
    // 나중에 다시오지 않기 위해 방문을 표시하고,

    // 정점 k와 연결된 모든 정점에 대하여
    for (int i=0; i<          ; i++) {
        // 정점k의 i번째 정점에 방문한 적이 없으면,
        if (          ) {
            // 그 정점에서 다시 dfs 시작

            // dfs 완료하고 돌아왔다고 메시지 출력
            printf("return to dfs(%d).\n", k);

        }
    }
    return;
}

int main() {
    input_G();
    output_G();
    dfs(0);
}
```



```
0:  1  3  8
1:  0  7
2:  3  5  7
3:  0  2  4
4:  3  8
5:  2  6
6:  5
7:  1  2
8:  0  4
```

```
dfs(0) started.
dfs(1) started.
dfs(7) started.
dfs(2) started.
dfs(3) started.
dfs(4) started.
dfs(8) started.
return to dfs(4).
return to dfs(3).
return to dfs(2).
dfs(5) started.
dfs(6) started.
return to dfs(5).
return to dfs(2).
return to dfs(7).
return to dfs(1).
return to dfs(0).
```

너비우선탐색(BFS)

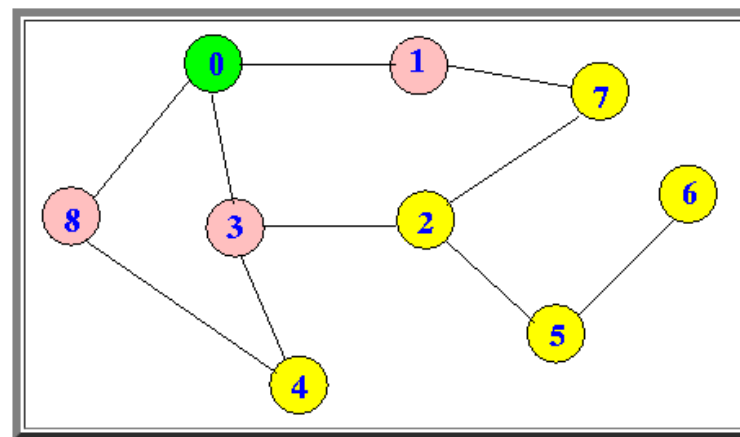
■ 너비우선탐색 알고리즘

: Breadth First Search

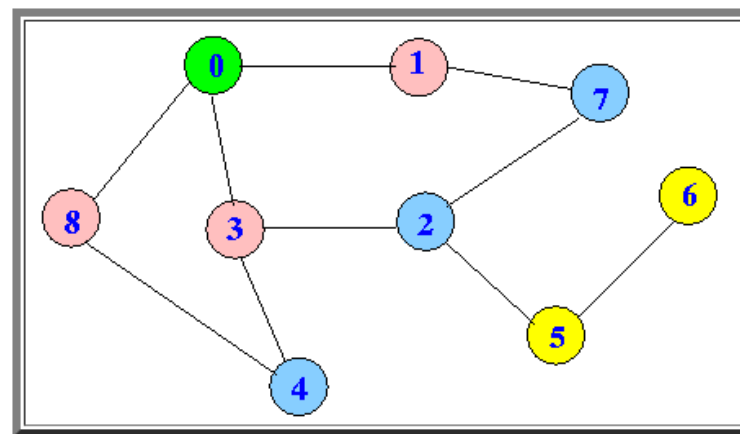
1. 정점 k 를 처리하고 큐에 삽입, k 를 방문한 것으로 표시
2. 큐가 빌 때까지 다음을 반복:
 - 1) 큐에서 첫 번째 항목 삭제
 - 2) 삭제된 항목과 이웃하는 모든 정점에 대하여 방문하지 않은 정점이라면,
 - ① 그 정점을 처리하고 큐에 삽입
 - ② 그 정점에 방문을 표시

■ 너비 우선의 의미

- Visit **all the neighbors** first:

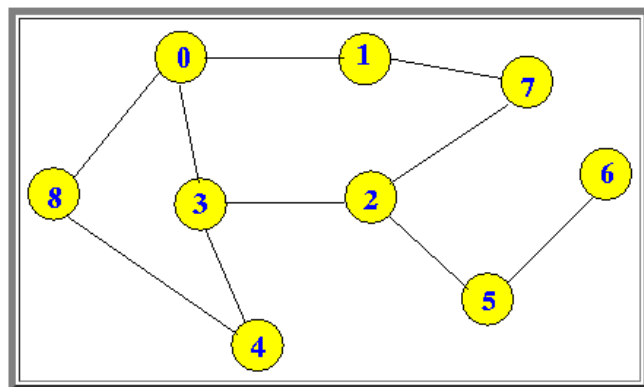


- Only then visit the **neighbors' neighbors**:

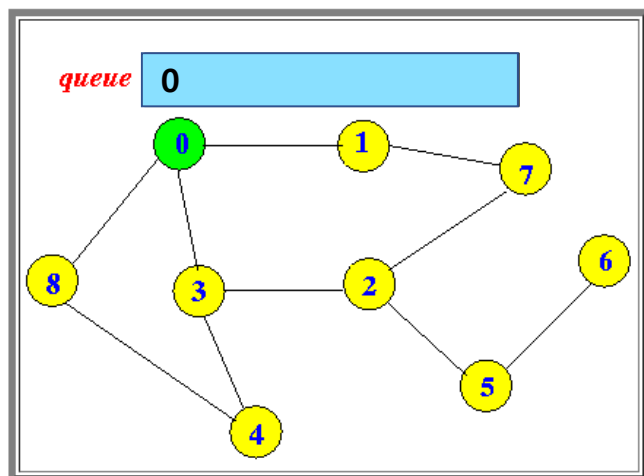


1. 정점 k 를 처리하고 큐에 삽입, k 를 방문한 것으로 표시
2. 큐가 빌 때까지 다음을 반복:
 - 1) 큐에서 첫 번째 항목 삭제
 - 2) 삭제된 항목과 이웃하는 모든 정점에 대하여 방문하지 않은 정점이라면,
 - ① 그 정점을 처리하고 큐에 삽입
 - ② 그 정점에 방문을 표시

① Graph:



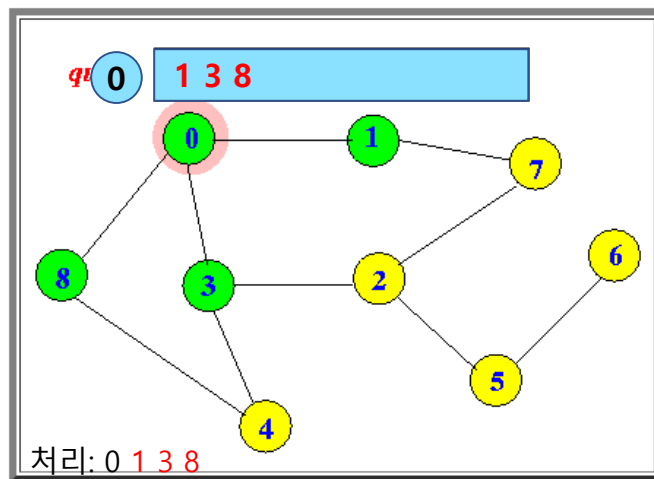
② Initial state:



처리: 0

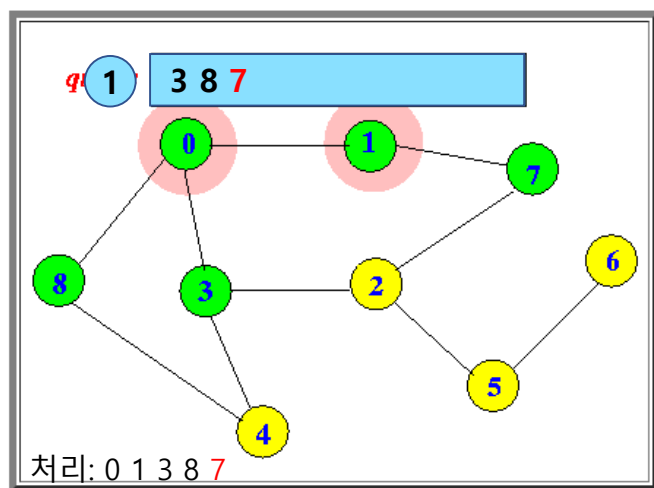
너비우선탐색(BFS)

■ State after processing 0



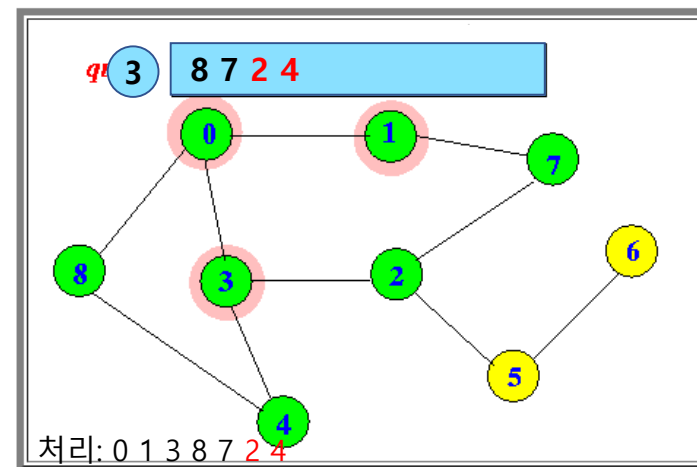
처리: 0 1 3 8

■ State after processing 1



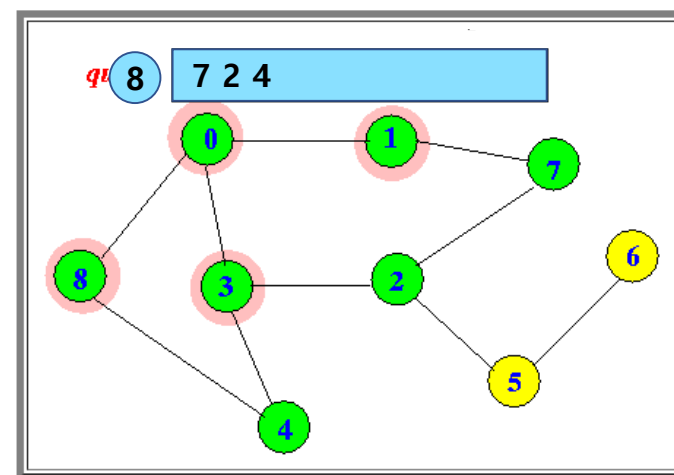
처리: 0 1 3 8 7

■ State after processing 3



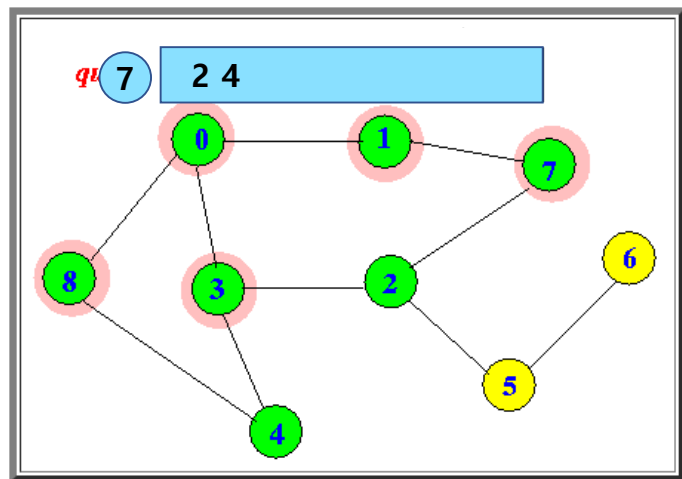
처리: 0 1 3 8 7 2 4

■ State after processing 8



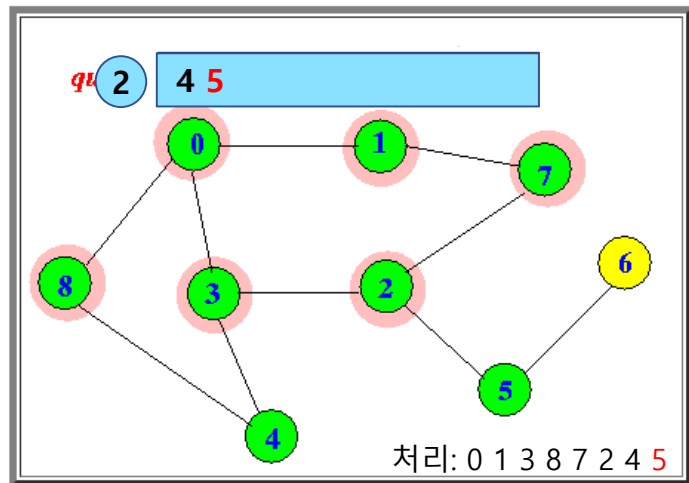
너비우선탐색(BFS)

① State after processing 7

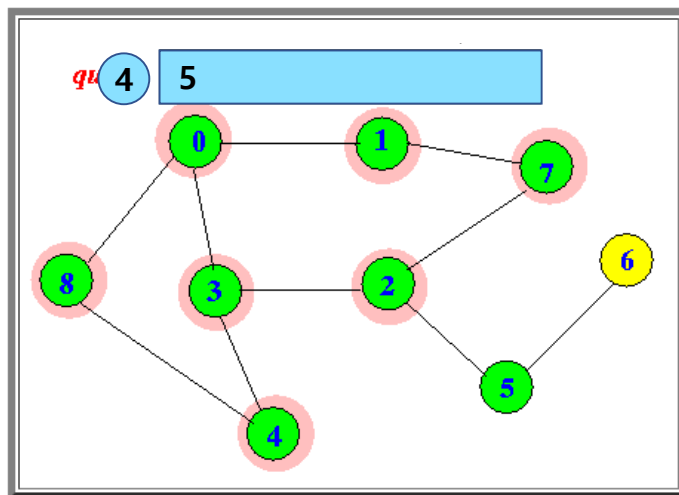


②

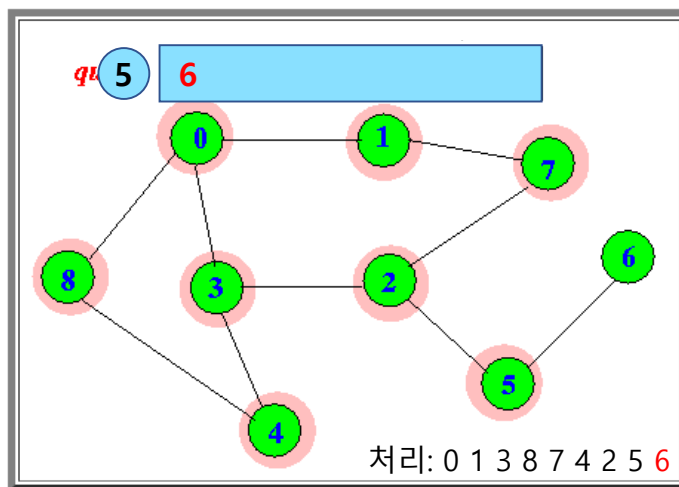
State after processing 2



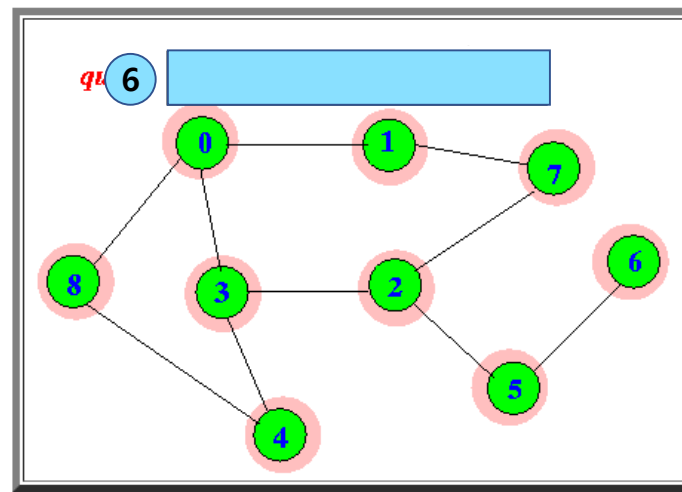
State after processing 4



State after processing 5



State after processing 6



DONE

(The queue has become empty)

너비우선탐색

■ 너비우선탐색 알고리즘

: Breadth First Search

1. 정점 k를 처리하고 큐에 삽입, k를 방문한 것으로 표시
2. 큐가 빌 때까지 다음을 반복:
 - 1) 큐에서 첫 번째 항목 삭제
 - 2) 삭제된 항목과 이웃하는 모든 정점에 대하여 방문하지 않은 정점이라면,
 - ① 그 정점을 처리하고 큐에 삽입
 - ② 그 정점에 방문을 표시

```
int n, m;
vector<vector<int>> G;
vector<bool> visited;

// 정점 k에서 bfs 시작
void bfs(int k) {
    queue<int> Q;    // 방금 방문한 이웃의 정점을 넣어 놓는 큐

    printf("bfs: (%d)\n", k);
    Q.push(k);      // 시작 정점을 Q에 삽입
    visited[k]=1;    // 시작 정점 방문했다고 표시

    while(!Q.empty()) {    // 큐에 내용물 있으면 계속반복
        int cur=Q.front(); // 큐의 첫번째 원소
        Q.pop();           // 빼냄(삭제)
        printf("%d deleted.\nbfs: ", cur);

        // 방금 전에 삭제한 정점과 이웃하는 모든 정점에 대하여
        for(int i=0; i<G[cur].size(); i++) {
            // 방문한 적이 없으면
            if(!visited[G[cur][i]]) {
                printf("(%d) ", G[cur][i]);
                Q.push(G[cur][i]);
                visited[G[cur][i]]=1;
            }
        }
        printf("\n");
    }
}
```

idx	[0]	[1]	[2]
G[0]:	1	3	8
G[1]:	0	7	
G[2]:	3	5	7
G[3]:	0	2	4
G[4]:	3	8	
G[5]:	2	6	
G[6]:	5		
G[7]:	1	2	
G[8]:	0	4	

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

int n, m;
vector<vector<int>> G;
vector<bool> visited;

// 현재 큐의 모습을 출력
void output_Q(queue<int> Q) {
    printf("Q: [");
    while(!Q.empty()) {
        int cur=Q.front();
        printf("%3d", cur);
        Q.pop();
    }
    printf("], ");
}
```

1. 정점 k를 처리하고 큐에 삽입, k를 방문한 것으로 표시
2. 큐가 빌 때까지 다음을 반복:
 - 1) 큐에서 첫 번째 항목 삭제
 - 2) 삭제된 항목과 이웃하는 모든 정점에 대하여 방문하지 않은 정점이라면,
 - ① 그 정점을 처리하고 큐에 삽입
 - ② 그 정점에 방문을 표시

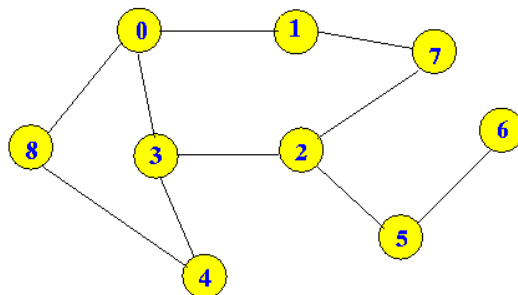
```
// 정점 k에서 bfs
void bfs(int k) {
    queue<int> Q;

    printf("bfs: (%d)\n", k);
    Q.push(k);          // 시작 정점을 Q에 삽입
    visited[k]=1;       // 시작 정점 방문했다고 표시

    while(!Q.empty()) {
        output_Q(Q);

        int cur=          // 큐의 첫번째 원소
        Q.  ();           // 빼냄(삭제)
        printf("%d deleted.\nbfs: ", cur);

        // 삭제한 정점과 이웃하는 모든 정점에 대하여
        for(int i=0; i<          ; i++) {
            // 방문한 적이 없으면
            if(!visited[          ]) {
                printf("(%)d ", G[cur][i]);
                Q.push(          );
                visited[          ]=1;
            }
        }
        printf("\n");
    }
}
```



```
void output_G() {
    printf("\n");
    for(int a=0; a<=n; a++) {
        printf("%2d:", a);
        for(int i : G[a])
            printf("%3d", i);
        printf("\n");
    }
    printf("\n");
}

void input_G() {
    scanf("%d %d", &n, &m);
    // 정점이 0부터 시작하므로 n+1
    G.resize(n+1);
    visited.assign(n+1, 0);

    for(int i=0; i<m; i++) {
        int a, b;
        scanf("%d %d", &a, &b);
        G[a].push_back(b);
        G[b].push_back(a);
    }
}

int main() {
    input_G();
    output_G();
    bfs(0);
}
```

```
0:  1  3  8
1:  0  7
2:  3  5  7
3:  0  2  4
4:  3  8
5:  2  6
6:  5
7:  1  2
8:  0  4
```

```
bfs: (0)
Q: [ 0], 0 deleted.
bfs: (1) (3) (8)
Q: [ 1 3 8], 1 deleted.
bfs: (7)
Q: [ 3 8 7], 3 deleted.
bfs: (2) (4)
Q: [ 8 7 2 4], 8 deleted.
bfs:
Q: [ 7 2 4], 7 deleted.
bfs:
Q: [ 2 4], 2 deleted.
bfs: (5)
Q: [ 4 5], 4 deleted.
bfs:
Q: [ 5], 5 deleted.
bfs: (6)
Q: [ 6], 6 deleted.
bfs:
```

미로 탈출 (BFS vs DFS)

<https://seanperfecto.github.io/BFS-DFS-Pathfinder/>

BFS/DFS Pathfinder (by Sean Perfecto) MAZE 1 | MAZE 2 | MAZE 3

Breadth-First Search

Depth-First Search

▶ ↺

미로 탈출

■ 문제

정진이는 벽과 길로 만들어진 N 행 M 열(세로 N 칸, 가로 M 칸) 크기의 미로에 갇혀 있다.

미로에서는 한 번에 한 칸씩만 이동할 수 있다. 벽은 0으로, 길은 0아닌 수로 표시된다.

정진이가 미로에서 탈출하기 위해 이동하여야 하는 최소 칸수를 구하시오. 시작 칸과 마지막 칸도 이동거리에 포함시킨다.

입력 예	출력 예
7 8 1 0 1 0 0 0 0 1 1 0 1 0 0 1 1 1 1 1 8 1 1 1 0 1 0 1 0 1 0 0 0 1 1 1 0 1 0 1 1 1 1 0 0 1 0 1 0 9 0 0 0 1 1 1 0 1	11

■ 입력

첫 번째 줄에 두 정수 N, M 이 주어진다.

($4 \leq N, M \leq 200$)

다음 N 개의 줄에는 각각 M 개의 정수로 미로의 정보가 주어진다. 숫자는 각각 다음을 의미한다.

(0: 벽, 1: 길, 8: 시작위치, 9: 도착위치)

■ 출력

첫 번째 줄에 최소 이동 칸의 개수를 출력한다.

미로 탈출

■ 지도 데이터

7 8

1 0 1 0 0 0 0 1

1 0 1 0 0 1 1 1

1 1 8 1 1 1 0 1

0 1 0 1 0 0 0 1

1 1 0 1 0 1 1 1

1 0 0 1 0 1 0 9

0 0 0 1 1 1 0 1

■ 그래프로 해석

- 지도데이터를 그래프로보고 탐색기법 적용



미로 탈출 (초기설계)

```
#include <stdio.h>
#include <algorithm>
#include <queue>
#define MAX_INT 0x7fffffff
using namespace std;

typedef struct {
    int a, b;    // a행 b열
} vertex;

int n, m;        // n행 m열
int M[201][201]; // 지도 정보
int Sa, Sb;      //출발지 a행 b열
int Ga, Gb;      //목적지 a행 b열

// 이동할 네 가지 방향 정의, 아래와 같이 적으면
// 아래, 오른쪽, 위쪽, 왼쪽 순서로 탐색하게 됨.
int da[] = {1, 0, -1, 0}; // 행(세로) 방향
int db[] = {0, 1, 0, -1}; // 열(가로) 방향

bool safe(int a, int b) { // a행 b열
    return (0<=a && a<n) && (0<=b && b<m);
}
```

```
void input() {
    // (n행 m열)
    scanf("%d %d", &n, &m);
    // 2차원 리스트의 맵 정보 입력 받기
    for(int a=0; a<n; a++) {
        for(int b=0; b<m; b++) {
            int c;
            scanf("%d", &c);

            if(c==8) // 출발지 설정
                Sa=a, Sb=b;
            else if(c==9) // 목적지 설정
                Ga=a, Gb=b;

            // 모든 길을 -1로 변경
            // a행 b열에 삽입
            if(c>=1)
                M[a][b]=-1;
            else
                M[a][b]=0;
        }
    }
}
```

```
// 현재 맵에서 탐색 상태를 출력
void output(const char* title, int dist) {
    // 제목출력
    printf("\n[%s] (%d)\n", title, dist);

    for(int a = 0; a < n; a++) {
        for(int b = 0; b < m; b++) {
            printf("%3d", M[a][b]);
        }
        printf("\n");
    }

    void dfs(int a, int b, int d) {
    }

    int main(void) {
        input();
        output("initial state",-1);
        // dfs 또는 bfs 하나의 함수만 호출할 것!
        dfs(Sa, Sb, 1); // Sa행 Sb열에서 DFS시작
        //bfs(Sa, Sb, 1); //Sa행 Sb열에서 BFS시작
        output("last state", -1);
        return 0;
    }
}
```

미로 탈출 - DFS로 구현

■ DFS 알고리즘

```
def dfs(k):
```

- 1) 정점 k를 처리하고 방문한 것으로 표시
- 2) k와 연결된 모든 정점에 대하여
방문한적이 없으면 그 정점에서 dfs,
방문이 완료되면 되돌아 오기

■ 초기맵 상태

Sa, Sb = (2, 2)

Ga, Gb = (5, 7)

	0	1	2	3	4	5	6	7
0	-1	0	-1	0	0	0	0	-1
1	-1	0	-1	0	0	-1	-1	-1
2	-1	-1	-1	-1	-1	-1	0	-1
3	0	-1	0	-1	0	0	0	-1
4	-1	-1	0	-1	0	-1	-1	-1
5	-1	0	0	-1	0	-1	0	-1
6	0	0	0	-1	-1	-1	0	-1

■ DFS 구현

```
// a행, b열에서 dfs, 현재까지 계산한 거리는 d
```

```
void dfs(int a, int b, int d) {
```

```
    ? //① a행 b열에 거리 기록
```

```
    //② 방문한 것으로 표시 이거 안해도 됨?
```

```
    //④ 목적지에 도착하면 맵상태와 도달거리 출력
```

```
    ?
```

```
    //③ 4방향으로 dfs
```

```
    // da, db배열을 이용하여 for문으로 간략화 가능
```

```
    // ↓
```

```
    // →
```

```
    // ↑
```

```
    // ←
```

```
}
```

미로 탈출 - DFS로 구현

■ 결과 고찰

```

7 8
1 0 1 0 0 0 0 1
1 0 1 0 0 1 1 1
1 1 8 1 1 1 0 1
0 1 0 1 0 0 0 1
1 1 0 1 0 1 1 1
1 0 0 1 0 1 0 9
0 0 0 1 1 1 0 1
    
```

```

[initial state] (-1)
-1 0 -1 0 0 0 0 -1
-1 0 -1 0 0 -1 -1 -1
-1 -1 -1 -1 -1 -1 0 -1
0 -1 0 -1 0 0 0 -1
-1 -1 0 -1 0 -1 -1 -1
-1 0 0 -1 0 -1 0 -1
0 0 0 -1 -1 -1 0 -1
    
```

```

[DFS success] (13)
-1 0 -1 0 0 0 0 -1
-1 0 -1 0 0 -1 -1 -1
-1 -1 1 2 -1 -1 0 -1
0 -1 0 3 0 0 0 -1
-1 -1 0 4 0 10 11 12
-1 0 0 5 0 9 0 13
0 0 0 6 7 8 0 -1
    
```

```

[last state] (-1)
5 0 3 0 0 0 0 16
4 0 2 0 0 17 16 15
3 2 1 2 19 18 0 14
0 3 0 3 0 0 0 13
5 4 0 4 0 10 11 12
6 0 0 5 0 9 0 13
0 0 0 6 7 8 0 -1
    
```

- 엥? 최적해가 아는데...
- 왜 최적해를 못 찾지?
- 원래 전체탐색이 진행되어야 하는데...

■ DFS 구현

```

// a행, b열에서 dfs, 현재까지 계산한 거리는 d
void dfs(int a, int b, int d) {
    M[a][b] = d; //① a행 b열에 거리 기록
    //② 방문한 것으로 표시 이거 안해도 됨?
    //④ 목적지에 도착하면 맵상태와 도달거리 출력
    if(a==Ga && b==Gb) {
        output("DFS success", d);
        return;
    }

    //③ 4방향으로 dfs
    // da, db배열을 이용하여 for문으로 간략화 가능
    if(safe(a+1, b) && M[a+1][b]==-1) // ↓
        dfs(a+1, b, d+1);
    if(safe(a, b+1) && M[a][b+1]==-1) // →
        dfs(a, b+1, d+1);
    if(safe(a-1, b) && M[a-1][b]==-1) // ↑
        dfs(a-1, b, d+1);
    if(safe(a, b-1) && M[a][b-1]==-1) // ←
        dfs(a, b-1, d+1);
}
    
```

미로 탈출 - DFS로 구현

■ 이전 버전

```
// a행, b열에서 dfs, 현재까지 계산한 거리는 d
void dfs(int a, int b, int d) {
    M[a][b] = d; //① a행 b열에 거리 기록
    //② 방문한 것으로 표시 이거 안해도 됨?
    //④ 목적지에 도착하면 맵상태와 도달거리 출력
    if(a==Ga && b==Gb) {
        output("DFS success", d);
        return;
    }

    //③ 4방향으로 dfs
    // da, db배열을 이용하여 for문으로 간략화 가능
    if(safe(a+1, b) && M[a+1][b]==-1) // ↓
        dfs(a+1, b, d+1);
    if(safe(a, b+1) && M[a][b+1]==-1) // →
        dfs(a, b+1, d+1);
    if(safe(a-1, b) && M[a-1][b]==-1) // ↑
        dfs(a-1, b, d+1);
    if(safe(a, b-1) && M[a][b-1]==-1) // ←
        dfs(a, b-1, d+1);
}
```

■ 방향 탐색 배열이용 업데이트

```
// 이동할 네 가지 방향 정의, 아래와 같이 적으면
// 아래, 오른쪽, 위쪽, 왼쪽 순서로 탐색하게 됨.
int da[] = {1, 0, -1, 0}; // 행(세로) 방향
int db[] = {0, 1, 0, -1}; // 열(가로) 방향

// a행, b열에서 dfs, 현재까지 계산한 거리는 d
void dfs(int a, int b, int d) {
    M[a][b] = d;
    if(a==Ga && b==Gb) {
        output("DFS success", d);
        return;
    }

    //③ 4방향으로 dfs
    for(int i=0; i<4; i++) {
        int na = a+da[i], nb = b+db[i];
        if(safe(na, nb) && M[na][nb]==-1) {
            dfs(na, nb, d+1);
        }
    }
}
```

미로 탈출 - DFS로 구현

■ DFS 구현 업데이트

```
// 최소거리(최적해)를 저장하기 위한 변수 셋팅
?
// a행, b열에서 dfs, 현재까지 계산한 거리는 d
void dfs(int a, int b, int d) {
    M[a][b] = d;
    if(a==Ga && b==Gb) {
        // 더 짧은 방법을 찾으면 최소거리 업데이트
        ?
        output("DFS success", d);
        return;
    }
    //③ 4방향으로 dfs 방법 업데이트, 백트랙시 길 복원
    ?
}
```

■ 결과 확인

[initial state] (-1)

-1	0	-1	0	0	0	0	-1
-1	0	-1	0	0	-1	-1	-1
-1	-1	-1	-1	-1	-1	0	-1
0	-1	0	-1	0	0	0	-1
-1	-1	0	-1	0	-1	-1	-1
-1	0	0	-1	0	-1	0	-1
0	0	0	-1	-1	-1	0	-1

[DFS success] (11)

-1	0	-1	0	0	0	0	-1
-1	0	-1	0	0	5	6	7
-1	-1	1	2	3	4	0	8
0	-1	0	-1	0	0	0	9
-1	-1	0	-1	0	-1	-1	10
-1	0	0	-1	0	-1	0	11
0	0	0	-1	-1	-1	0	-1

[DFS success] (13)

-1	0	-1	0	0	0	0	-1
-1	0	-1	0	0	-1	-1	-1
-1	-1	1	2	-1	-1	0	-1
0	-1	0	3	0	0	0	-1
-1	-1	0	4	0	10	11	12
-1	0	0	5	0	9	0	13
0	0	0	6	7	8	0	-1

[last state] (-1)

-1	0	-1	0	0	0	0	-1
-1	0	-1	0	0	-1	-1	-1
-1	-1	1	-1	-1	-1	0	-1
0	-1	0	-1	0	0	0	-1
-1	-1	0	-1	0	-1	-1	-1
-1	0	0	-1	0	-1	0	-1
0	0	0	-1	-1	-1	0	-1

미로 탈출 - DFS 최종 구현

```

int min_dist=MAX_INT;
// a행, b열에서 dfs, 현재까지 계산한 거리는 d
void dfs(int a, int b, int d) {
    M[a][b] = d;

    // 목적지에 도달하면,
    if(a==Ga && b==Gb) {
        if(min_dist > d)
            min_dist=d;
        output("DFS search success", d);
        return;
    }

    int noway=0; // 현재 위치에서 이동 불가능 방향 개수
    for(int i=0; i<4; i++) {
        int na = a+da[i];
        int nb = b+db[i];

        if(safe(na, nb) && M[na][nb]==-1) {
            dfs(na, nb, d+1);
            M[na][nb]=-1; // 백트랙할 경우 길복원
        }
        else
            noway++; // 이동불가 카운터 증가

        if(noway==4) // 4방향 모두 길이 막혀있으면,
            output("DFS search fail", -1);
    }
}

```

[initial state] (-1)

```

-1 0 -1 0 0 0 0 -1
-1 0 -1 0 0 -1 -1 -1
-1 -1 -1 -1 -1 -1 0 -1
0 -1 0 -1 0 0 0 -1
-1 -1 0 -1 0 -1 -1 -1
-1 0 0 -1 0 -1 0 -1
0 0 0 -1 -1 -1 0 -1

```

[DFS search success] (13)

```

-1 0 -1 0 0 0 0 -1
-1 0 -1 0 0 -1 -1 -1
-1 -1 1 2 -1 -1 0 -1
0 -1 0 3 0 0 0 -1
-1 -1 0 4 0 10 11 12
-1 0 0 5 0 9 0 13
0 0 0 6 7 8 0 -1

```

[DFS search fail] (-1)

```

-1 0 -1 0 0 0 0 16
-1 0 -1 0 0 -1 -1 15
-1 -1 1 2 -1 -1 0 14
0 -1 0 3 0 0 0 13
-1 -1 0 4 0 10 11 12
-1 0 0 5 0 9 0 -1
0 0 0 6 7 8 0 -1

```

[DFS search fail] (-1)

```

-1 0 -1 0 0 0 0 -1
-1 0 -1 0 0 17 16 15
-1 -1 1 2 19 18 0 14
0 -1 0 3 0 0 0 13
-1 -1 0 4 0 10 11 12
-1 0 0 5 0 9 0 -1
0 0 0 6 7 8 0 -1

```

[DFS search success] (11)

```

-1 0 -1 0 0 0 0 -1
-1 0 -1 0 0 5 6 7
-1 -1 1 2 3 4 0 8
0 -1 0 -1 0 0 0 9
-1 -1 0 -1 0 -1 -1 10
-1 0 0 -1 0 -1 0 11
0 0 0 -1 -1 -1 0 -1

```

[DFS search fail] (-1)

```

-1 0 -1 0 0 0 0 -1
-1 0 -1 0 0 5 6 7
-1 -1 1 2 3 4 0 8
0 -1 0 19 0 0 0 9
-1 -1 0 18 0 12 11 10
-1 0 0 17 0 13 0 -1
0 0 0 16 15 14 0 -1

```

[DFS search fail] (-1)

```

-1 0 -1 0 0 0 0 8
-1 0 -1 0 0 5 6 7
-1 -1 1 2 3 4 0 -1
0 -1 0 -1 0 0 0 -1
-1 -1 0 -1 0 -1 -1 -1
-1 0 0 -1 0 -1 0 -1
0 0 0 -1 -1 -1 0 -1

```

[DFS search fail] (-1)

```

-1 0 3 0 0 0 0 -1
-1 0 2 0 0 -1 -1 -1
-1 -1 1 -1 -1 -1 0 -1
0 -1 0 -1 0 0 0 -1
-1 -1 0 -1 0 -1 -1 -1
-1 0 0 -1 0 -1 0 -1
0 0 0 -1 -1 -1 0 -1

```

[DFS search fail] (-1)

```

-1 0 -1 0 0 0 0 -1
-1 0 -1 0 0 -1 -1 -1
-1 2 1 -1 -1 -1 0 -1
0 3 0 -1 0 0 0 -1
5 4 0 -1 0 -1 -1 -1
6 0 0 -1 0 -1 0 -1
0 0 0 -1 -1 -1 0 -1

```

[DFS search fail] (-1)

```

5 0 -1 0 0 0 0 -1
4 0 -1 0 0 -1 -1 -1
3 2 1 -1 -1 -1 0 -1
0 -1 0 -1 0 0 0 -1
-1 -1 0 -1 0 -1 -1 -1
-1 0 0 -1 0 -1 0 -1
0 0 0 -1 -1 -1 0 -1

```

[last state] (-1)

```

-1 0 -1 0 0 0 0 -1
-1 0 -1 0 0 -1 -1 -1
-1 -1 1 -1 -1 -1 0 -1
0 -1 0 -1 0 0 0 -1
-1 -1 0 -1 0 -1 -1 -1
-1 0 0 -1 0 -1 0 -1
0 0 0 -1 -1 -1 0 -1

```

미로 탈출 - BFS로 구현

■ BFS 알고리즘

1. 정점 k를 처리하고 큐에 삽입, k를 방문한 것으로 표시
2. 큐가 빌 때까지 다음을 반복:
 - 1) 큐에서 첫 번째 항목 삭제
 - 2) 삭제된 항목과 이웃하는 모든 정점에 대하여 방문하지 않은 정점이라면,
 - ① 그 정점을 처리하고 큐에 삽입
 - ② 그 정점에 방문을 표시

■ 초기 맵 상태

Sa, Sb = (2, 2)
Ga, Gb = (5, 7)

	0	1	2	3	4	5	6	7
0	-1	0	-1	0	0	0	0	-1
1	-1	0	-1	0	0	-1	-1	-1
2	-1	-1	-1	-1	-1	-1	0	-1
3	0	-1	0	-1	0	0	0	-1
4	-1	-1	0	-1	0	-1	-1	-1
5	-1	0	0	-1	0	-1	0	-1
6	0	0	0	-1	-1	-1	0	-1

```
// bfs(시작행a, 시작열b, 거리d)
void bfs(int sa, int sb, int d) {
    ? // 시작위치에 거리 d표시, 방문표시 안함?
    queue<vertex> q;
    ? // 큐에 현재 위치 삽입

    while(!q.empty()) { // 큐가 빌 때까지 반복
        ? // 큐의 첫 번째 정점 a행
        ? // 큐의 첫 번째 정점 b열
        ? // 큐에서 첫 번째 항목 삭제

        // 현재 위치에서 4가지 방향으로의 위치 확인
        for(int i = 0; i < 4; i++) {
            ? // na: next a
            ? // nb: next b
            if (safe(na, nb) && M[na][nb] == -1) {
                ? // 다음 위치에 거리 표시
                ? // 목적지에 도달하면 성공 출력하고 탈출

                ? // 다음 위치를 큐에 삽입
            }
        }
    }

    output("BFS search fail", -1); // 여기까지 오면 탐색 실패임
}
```


미로 탈출 - BFS로 구현

```
// bfs(시작행a, 시작열b, 거리d)
void bfs(int sa, int sb, int d) {
    M[sa][sb] = d;    // 시작위치에 거리 d표시
    queue<vertex> q;
    q.push({sa, sb}); // 큐에 현재 위치 삽입

    while(!q.empty()) { // 큐가 빌 때까지 반복
        int a = q.front().a; // 큐의 첫 번째 정점 a행
        int b = q.front().b; // 큐의 첫 번째 정점 b열
        q.pop();           // 큐에서 첫 번째 항목 삭제
        // 현재 위치에서 4가지 방향으로의 위치 확인
        for(int i=0; i<4; i++) {
            int na=a+da[i]; // na: next a
            int nb=b+db[i]; // nb: next b
            if (safe(na, nb) && M[na][nb] == -1) {
                M[na][nb] = M[a][b] + 1; // 다음 위치에 거리 표시
                // 목적지에 도달하면 성공 출력하고 탈출
                if(na==Ga && nb==Gb) {
                    output("BFS search success", M[na][nb]);
                    return;
                }
                q.push({na, nb}); // 다음 위치를 큐에 삽입
            }
        }
    }
    output("BFS search fail", -1); // 여기까지 오면 탐색 실패임
}
```

■ 결과 확인

```
7 8
1 0 1 0 0 0 0 1
1 0 1 0 0 1 1 1
1 1 8 1 1 1 0 1
0 1 0 1 0 0 0 1
1 1 0 1 0 1 1 1
1 0 0 1 0 1 0 9
0 0 0 1 1 1 0 1
```

[initial state] (-1)

```
-1 0 -1 0 0 0 0 -1
-1 0 -1 0 0 -1 -1 -1
-1 -1 -1 -1 -1 -1 0 -1
0 -1 0 -1 0 0 0 -1
-1 -1 0 -1 0 -1 -1 -1
-1 0 0 -1 0 -1 0 -1
0 0 0 -1 -1 -1 0 -1
```

[BFS search success] (11)

```
5 0 3 0 0 0 0 8
4 0 2 0 0 5 6 7
3 2 1 2 3 4 0 8
0 3 0 3 0 0 0 9
5 4 0 4 0 10 11 10
6 0 0 5 0 9 0 11
0 0 0 6 7 8 0 -1
```

[last state] (-1)

```
5 0 3 0 0 0 0 8
4 0 2 0 0 5 6 7
3 2 1 2 3 4 0 8
0 3 0 3 0 0 0 9
5 4 0 4 0 10 11 10
6 0 0 5 0 9 0 11
0 0 0 6 7 8 0 -1
```

미로 탈출 - BFS로 구현

■ 탐색 과정을 보여주는 업데이트

```
if (safe(na, nb) && M[na][nb] == -1) {  
    int new_d = M[a][b] + 1; // 새로운 거리 계산  
  
    // 새롭게 계산된 거리가 지금 거리보다 멀어지면,  
    if(new_d > d) { // 현재 맵상태 일단 출력하고  
        output("BFS searched new distance", -1);  
        d = new_d;  
    }  
    M[na][nb] = new_d; // 다음 위치에 거리 표시  
    // 목적지에 도달하면 성공 출력하고 탈출  
    if(na==Ga && nb==Gb) {  
        output("BFS search success", M[na][nb]);  
        return;  
    }  
    q.push({na, nb}); // 다음 위치를 큐에 삽입  
}
```

[BFS searched new distance]

-1	0	-1	0	0	0	0	-1
-1	0	-1	0	0	-1	-1	-1
-1	-1	1	-1	-1	-1	0	-1
0	-1	0	-1	0	0	0	-1
-1	-1	0	-1	0	-1	-1	-1
-1	0	0	-1	0	-1	0	-1
0	0	0	-1	-1	-1	0	-1

[BFS searched new distance]

5	0	3	0	0	0	0	-1
4	0	2	0	0	5	-1	-1
3	2	1	2	3	4	0	-1
0	3	0	3	0	0	0	-1
5	4	0	4	0	-1	-1	-1
-1	0	0	5	0	-1	0	-1
0	0	0	-1	-1	-1	0	-1

[BFS searched new distance]

-1	0	-1	0	0	0	0	-1
-1	0	2	0	0	-1	-1	-1
-1	2	1	2	-1	-1	0	-1
0	-1	0	-1	0	0	0	-1
-1	-1	0	-1	0	-1	-1	-1
-1	0	0	-1	0	-1	0	-1
0	0	0	-1	-1	-1	0	-1

[BFS searched new distance]

5	0	3	0	0	0	0	-1
4	0	2	0	0	5	6	-1
3	2	1	2	3	4	0	-1
0	3	0	3	0	0	0	-1
5	4	0	4	0	-1	-1	-1
6	0	0	5	0	-1	0	-1
0	0	0	6	-1	-1	0	-1

[BFS searched new distance]

-1	0	3	0	0	0	0	-1
-1	0	2	0	0	-1	-1	-1
3	2	1	2	3	-1	0	-1
0	3	0	3	0	0	0	-1
-1	-1	0	-1	0	-1	-1	-1
-1	0	0	-1	0	-1	0	-1
0	0	0	-1	-1	-1	0	-1

[BFS searched new distance]

5	0	3	0	0	0	0	-1
4	0	2	0	0	5	6	7
3	2	1	2	3	4	0	-1
0	3	0	3	0	0	0	-1
5	4	0	4	0	-1	-1	-1
6	0	0	5	0	-1	0	-1
0	0	0	6	7	-1	0	-1

[BFS searched new distance]

-1	0	3	0	0	0	0	-1
4	0	2	0	0	-1	-1	-1
3	2	1	2	3	4	0	-1
0	3	0	3	0	0	0	-1
-1	4	0	4	0	-1	-1	-1
-1	0	0	-1	0	-1	0	-1
0	0	0	-1	-1	-1	0	-1

[BFS searched new distance]

5	0	3	0	0	0	0	8
4	0	2	0	0	5	6	7
3	2	1	2	3	4	0	8
0	3	0	3	0	0	0	-1
5	4	0	4	0	-1	-1	-1
6	0	0	5	0	-1	0	-1
0	0	0	6	7	8	0	-1

미로 탈출 DFS + BFS 전체 소스코드

```
//written by akapo@naver.com
#include <stdio.h>
#include <algorithm>
#include <queue>
#define MAX_INT 0x7fffffff
using namespace std;

typedef struct {
    int a, b;
} vertex;

int n, m;
int M[201][201]; // 지도 정보
int Sa, Sb; //출발지 a행 b열
int Ga, Gb; //목적지 a행 b열

// 이동할 네 가지 방향 정의, 아래와 같이 적으면
// 아래, 오른쪽, 위쪽, 왼쪽 순서로 탐색하게 됨.
int da[] = {1, 0, -1, 0}; // 행 방향
int db[] = {0, 1, 0, -1}; // 열 방향

bool safe(int a, int b) { // a행 b열
    return (0<=a && a<n) && (0<=b && b<m);
}

void input() {
    // N, M을 공백을 기준으로 구분하여 입력 받기(n행 m열)
    scanf("%d %d", &n, &m);
    // 2차원 리스트의 맵 정보 입력 받기
    for(int a=0; a<n; a++) {
        for(int b=0; b<m; b++) {
            int c;
            scanf("%d", &c);

            if(c==8) Sa=a, Sb=b; // 출발지 설정
            else if(c==9) Ga=a, Gb=b; // 목적지 설정

            // 출발지 목적지 포함 모든 길을 -1로 변경함
            if(c>=1) M[a][b]=-1;
            else M[a][b]=0; // a행 b열에 삽입
        }
    }
}
```

```
// 현재 맵에서 탐색 상태를 출력
void output(const char* title, int dist) {
    if(dist > 0)
        printf("\n[%s] (%d)\n", title, dist);
    else
        printf("\n[%s]\n", title);

    for(int a=0; a<n; a++) {
        for(int b = 0; b < m; b++) {
            printf("%3d", M[a][b]);
        }
        printf("\n");
    }
}

// bfs(시작행a, 시작열b, 거리d)
void bfs(int sa, int sb, int d) {
    M[sa][sb] = d;
    queue<vertex> q;
    q.push({sa, sb});

    while(!q.empty()) { // 큐가 빌 때까지 반복하기
        int a = q.front().a;
        int b = q.front().b;
        q.pop();

        // 현재 위치에서 4가지 방향으로의 위치 확인
        for(int i=0; i<4; i++) {
            int na=a+da[i]; // na: next a
            int nb=b+db[i]; // nb: next b

            if (safe(na, nb) && M[na][nb] == -1) {
                M[na][nb] = M[a][b] + 1;

                if(na==Ga && nb==Gb) { // 목적지에 도달하면,
                    output("BFS search success", M[na][nb]);
                    return;
                }
                q.push({na, nb});
            }
        }
    }
    output("BFS search fail", -1);
}
```

```
int min_dist=MAX_INT;
// a행, b열에서 dfs, 현재까지 계산한 거리는 d
void dfs(int a, int b, int d) {
    M[a][b] = d;

    // 목적지에 도달하면,
    if(a==Ga && b==Gb) {
        if(min_dist > d)
            min_dist=d;
        output("DFS search success", d);
        return;
    }

    int noway=0; // 현재 위치에서 이동 불가능 방향 개수
    for(int i=0; i<4; i++) {
        int na = a+da[i];
        int nb = b+db[i];

        if(safe(na, nb) && M[na][nb]==-1) {
            dfs(na, nb, d+1);
            M[na][nb]=-1; // 백트랙할 경우 길복원
        }
        else
            noway++; // 이동불가 카운터 증가

        if(noway==4) // 4방향 모두 길이 막혀있으면,
            output("DFS search fail", -1);
    }
}

int main(void) {
    input();
    output("initial state",-1);
    // dfs 또는 bfs 하나의 함수만 호출할 것!
    // Sa행 Sb열에서 DFS시작
    dfs(Sa, Sb, 1);
    // Sa행 Sb열에서 BFS시작
    //bfs(Sa, Sb, 1);
    output("last state",-1);
    return 0;
}
```

질의 응답

