

Content

IV. 프로그램 제어

1. 연산자
2. 순서도
3. 제어문
4. 반복문
5. 비트연산
6. 시프트연산자
7. Bit 연산자
8. 변수 범위

연산자

- 주어진 식을 계산하여 결과를 얻는 과정을 연산이라고 하며, 연산을 수행 하는 기호
- 피연산자 : 연산자의 작업 대상(변수, 상식, 수식)

종류	연산자
대입 연산자	=
산술 연산자	+, -, *, /, &, ++, --
관계 연산자	<, >, <=, >=, ==, !=
논리 연산자	&&, , !
할당 연산자	+=, -=, *=, /=, %=
삼항 연산자	?
비트 연산자	&, , ~, ^, <<, >>

2. 대입 연산자

4. 프로그램 제어 4-1. 연산자

대입연산자

- = 로 나타내며 변수에 상수 값 또는 다른 변수 값을 대입



```
#include <stdio.h>

void main() {
    int data1, data2;
    data1 = 2;
    data2 = 3;
    printf("data1 : %d, data2 : %d\n", data1, data2);
}
```

```
C:\WINDOWS\system32\cmd.exe
data1 : 2, data2 : 3
Press any key to continue . . .
```

3. 산술 연산자

4. 프로그램 제어 4-1. 연산자

산술 연산자

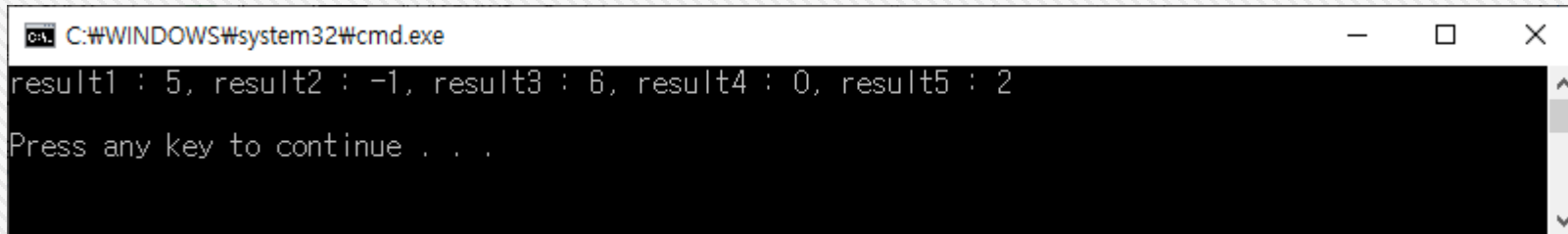
- 상수 또는 변수의 값을 이용 하여 +(더하기), -(빼기), *(곱하기), %(나누기) 함
- 나누기에서 몫과 나머지가 있는데 몫은 / 연산자, 나머지는 % 연산자 사용

```
#include <stdio.h>

void main() {
    int data1, data2;
    data1 = 2;
    data2 = 3;

    int result1 = data1 + data2;
    int result2 = data1 - data2;
    int result3 = data1 * data2;
    int result4 = data1 / data2; /* 2를 3으로 나눔 : 몫을 result4 에 저장 */
    int result5 = data1 % data2; /* 2를 3으로 나눔 : 나머지 result5 에 저장 */

    printf("result1 : %d, result2 : %d, result3 : %d, result4 : %d, result5 : %d\n"
           , result1, result2, result3, result4, result5);
}
```



C:\WINDOWS\system32\cmd.exe

result1 : 5, result2 : -1, result3 : 6, result4 : 0, result5 : 2

Press any key to continue . . .

3. 산술 연산자

4. 프로그램 제어

4-1. 연산자

산술 연산자 - 나머지 연산의 주의 사항

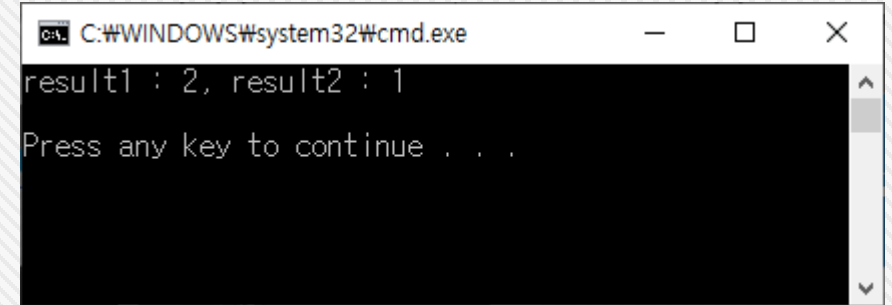
- C언어에서는 정수 끼리 나눗셈을 하면 정수

```
#include <stdio.h>

void main() {
    int data1, data2;
    data1 = 5;
    data2 = 2;

    int result1 = data1 / data2; /* 5를 2으로 나눔 : 몫 */
    int result2 = data1 % data2; /* 5를 2으로 나눔 : 나머지 */

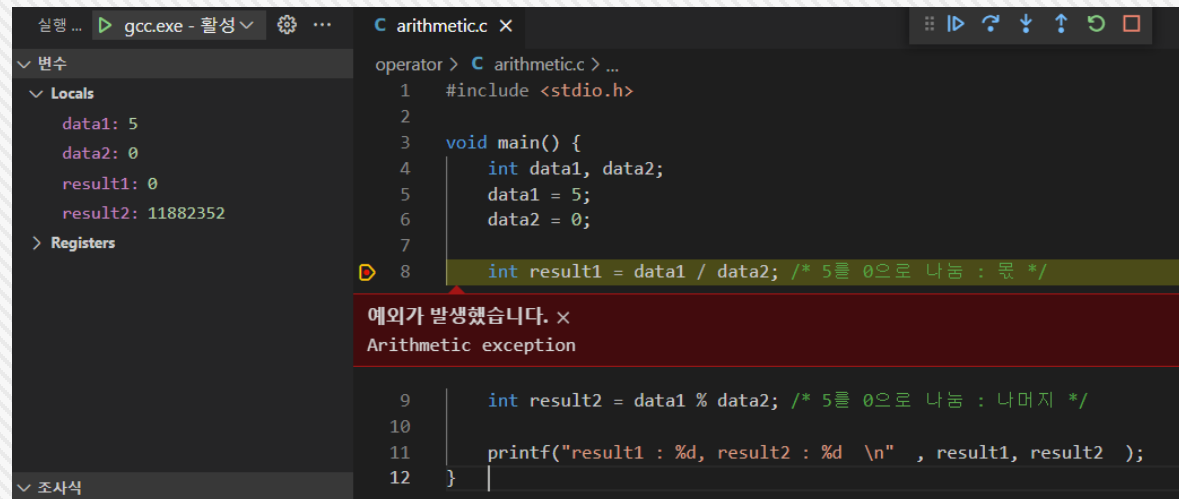
    printf("result1 : %d, result2 : %d \n" , result1, result2 );
}
```



```
C:\WINDOWS\system32\cmd.exe
result1 : 2, result2 : 1
Press any key to continue . . .
```

5/2 는 2.1

- 0으로 나눈다면 어떤 결과 일까 ?
 - : 컴파일시 에러 발생 하지 않음
 - : 실행 시 오류 발생



3. 산술 연산자

4. 프로그램 제어

4-1. 연산자

산술 연산자 - 실수

```
#include <stdio.h>

int main()
{
    float num1;
    float num2;

    num1 = 1.23f * 2.58f;    // 1.23을 2.58f를 곱
    num2 = 5.0f / 2.0f;      // 5.0에서 2.0을 나누기

    printf("%f\n", num1);    // 3.173400
    printf("%f\n", num2);    // 2.500000

    return 0;
}
```

```
#include <stdio.h>

int main()
{
    float num1;

    num1 = 5.0f / 0.0f;      // 5.0에서 0.0을 나누기
    printf("%f\n", num1);    // 1.#INF00

    return 0;
}
```

실수에서 0으로 나누면 무한대가 나온다.

0을 어떤 수로 나누면 어떻게 되나요?

- 0 / 10과 같이 0을 10으로 나누면 결과는 0입니다. 즉, $0 / 10 = 0$ 에서 10을 등호 오른쪽으로 보내면 $0 = 0 * 10$ 이 되므로 올바른 식.

나머지 연산은 정수에서만 사용 가능하고 실수에서는 사용할 수 없음

3. 산술 연산자

4. 프로그램 제어

4-1. 연산자

산술 연산자 - 실수에서 나머지 연산

- math.h 헤더 파일의 fmod, fmodf, fmodl 함수 사용
- fmod : double, fmodf : float, fmodl : long

```
#include <stdio.h>
#include <math.h>    // fmod 함수가 선언된 헤더 파일

int main()
{
    // 실수의 나머지 연산은 fmod, fmodf, fmodl 함수를 사용

    double num1 = 5.523;
    double num2 = 1.25;
    printf("%f\n", fmod(num1, num2));    // 0.523000

    float num3 = 5.523f;
    float num4 = 2.25f;
    printf("%f\n", fmodf(num3, num4));    // 1.023000

    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
0.523000
1.023000
Press any key to continue . . .
```

4. 증감 연산자

4. 프로그램 제어 4-1. 연산자

증감 연산자 - ++, --

- ++ : 순차적으로 1씩 증가, -- : 순차적으로 1씩 감소
- 전위형 : 연산을 먼저 수행 , 후위형 : 연산을 나중에 수행

```
operator > C math.c > main()
1  #include <stdio.h>
2
3  int main() {
4      int i = 10, numPostfix;
5      int j = 10, numprefix;
6      numPostfix = ++i;
7      numprefix = j++;
8
9      printf("PostFix : %d, PreFix : %d\n", numPostfix, numprefix);
10     return 0;
11 }
```

Locals

- i: 11
- numPostfix: 11
- j: 11
- numprefix: 10

Registers

```
C:\WINDOWS\system32\cmd.exe
PostFix : 11, PreFix : 10
Press any key to continue . . .
```


5. 관계 연산자

4. 프로그램 제어 4-1. 연산자

관계 연산자

- 두 수를 비교 하여 결과를 참(1), 거짓(0)으로 표시

```
#include <stdio.h>

int main() {
    int numa = 4, numb = 7;

    int result1 = numa > numb;
    int result2 = numa < numb;
    int result3 = numa == numb;
    int result4 = numa != numb;

    printf(" numa > numb : %d \n", result1);
    printf(" numa < numb : %d \n", result2);
    printf(" numa == numb : %d \n", result3);
    printf(" numa != numb : %d \n", result4);

    return 0;
}
```



C:\WINDOWS\system32\cmd.exe

```
numa > numb : 0
numa < numb : 1
numa == numb : 0
numa != numb : 1

Press any key to continue . . .
```

6. 논리 연산자

논리 연산자

- 두 값을 해당 하는 논리로 연산
- && : 두 값이 모두 참 이여야 결과 참
- || : 두 값 중 하나라도 참이면 결과 참
- ! : 값이 거짓 이면 참, 참 이면 거짓
- 0 이외는 모두 참(1)

A	B	A && B	A B	!A
0 (거짓)	0 (거짓)	0 (거짓)	1 (참)	1 (참)
0 (거짓)	1 (참)	0 (거짓)	1 (참)	1 (참)
1 (참)	0 (거짓)	0 (거짓)	1 (참)	0 (거짓)
1 (참)	1 (참)	1 (참)	0 (거짓)	0 (거짓)

```
#include <stdio.h>

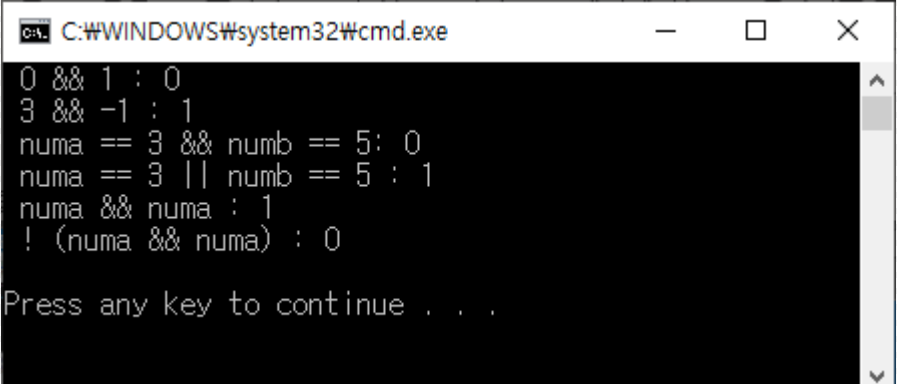
int main() {
    int numa = 10, numb = 5;

    printf(" 0 && 1 : %d \n", 0 && 1);
    printf(" 3 && -1 : %d \n", 3 && -1);

    printf(" numa == 3 && numb == 5: %d \n", numa == 3 && numb == 5);
    printf(" numa == 3 || numb == 5 : %d \n", numa == 3 || numb == 5);

    printf(" numa && numa : %d \n", numa && numa);
    printf(" ! (numa && numa) : %d \n", ! (numa && numa));

    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
0 && 1 : 0
3 && -1 : 1
numa == 3 && numb == 5: 0
numa == 3 || numb == 5 : 1
numa && numa : 1
! (numa && numa) : 0

Press any key to continue . . .
```

7. 할당 연산자

4. 프로그램 제어 4-1. 연산자

할당 연산자

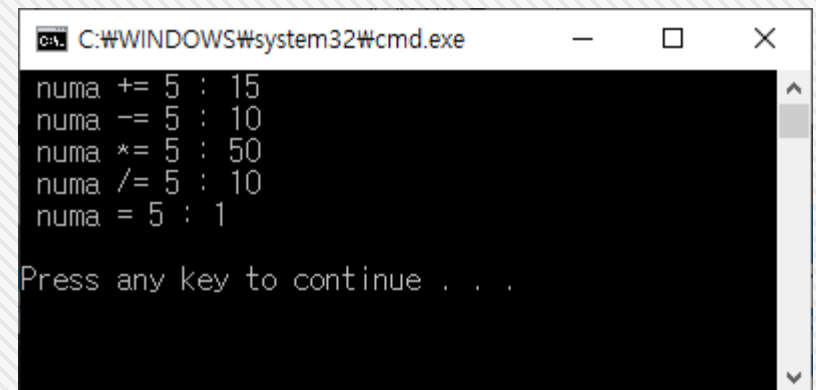
- 대입 연산자와 산술 연산자를 줄여서 사용

연산자	기능
+=	자신에 오른쪽 값을 더해 넣는다.
-=	자신에 오른쪽 값을 빼 넣는다.
*=	자신을 오른쪽 값으로 곱해 넣는다.
/=	자신을 오른쪽 값으로 나누어 몫을 넣는다.
%=	자신을 오른쪽 값으로 나누어 나머지를 넣는다.

```
#include <stdio.h>

int main() {
    int numa = 10, numb = 5;

    printf(" numa += 5 : %d \n", numa += 5);
    printf(" numa -= 5 : %d \n", numa -= 5);
    printf(" numa *= 5 : %d \n", numa *= 5);
    printf(" numa /= 5 : %d \n", numa /= 5);
    printf(" numa %= 5 : %d \n", numa %= 3);
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
numa += 5 : 15
numa -= 5 : 10
numa *= 5 : 50
numa /= 5 : 10
numa %= 5 : 1
Press any key to continue . . .
```

8. 삼항 연산자

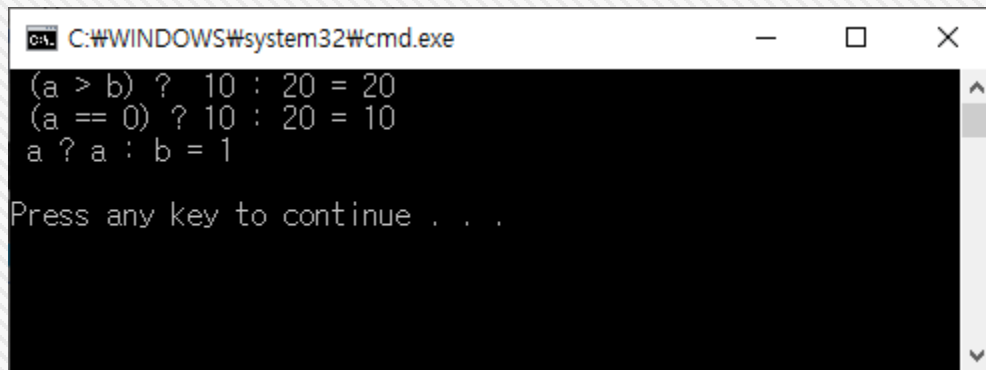
삼항 연산자

- 조건 ? 참일때의 값 : 거짓일때의 값;

```
#include <stdio.h>

int main() {
    int a = 0, b = 1, c = 0;

    printf(" (a > b) ? 10 : 20 = %d \n", (a > b) ? 10 : 20);
    printf(" (a == 0) ? 10 : 20 = %d \n", (a == 0) ? 10 : 20);
    printf(" a ? a : b = %d \n", a ? a : b);
    return 0;
}
```



C:\WINDOWS\system32\cmd.exe

```
(a > b) ? 10 : 20 = 20
(a == 0) ? 10 : 20 = 10
a ? a : b = 1
Press any key to continue . . .
```

수식 구성

- 논리 연산자의 특성을 이용한 조건 구성

data > 5

&&

data ++

data 가 3이면 data > 5 조건이 만족 하지 않으므로 data 값은 변경 없음
data 가 6이면 data > 5 조건이 만족 하여 data 값 증가 함.

```
#include <stdio.h>

int main() {
    int data = 3;

    printf( "data > 5 && data++ = %d\n", data > 5 && data++);
    printf( "data = %d\n", data);
    return 0;
}
```

data > 5 && data++ = 0
data = 0

```
#include <stdio.h>

int main() {
    int data = 7;

    printf( "data > 5 && data++ = %d\n", data > 5 && data++);
    printf( "data = %d\n", data);
    return 0;
}
```

data > 5 && data++ = 1
data = 8

data > 5

||

data ++

data > 5 조건이 만족 하지 않으므로 data 값은 증가함
data > 5 조건이 만족 하여 data 값은 변경 없음

```
#include <stdio.h>

int main() {
    int data = 3;

    printf( "data > 5 || data++ = %d\n", data > 5 || data++);
    printf( "data = %d\n", data);
    return 0;
}
```

data > 5 || data++ = 1
data = 4

```
#include <stdio.h>

int main() {
    int data = 3;

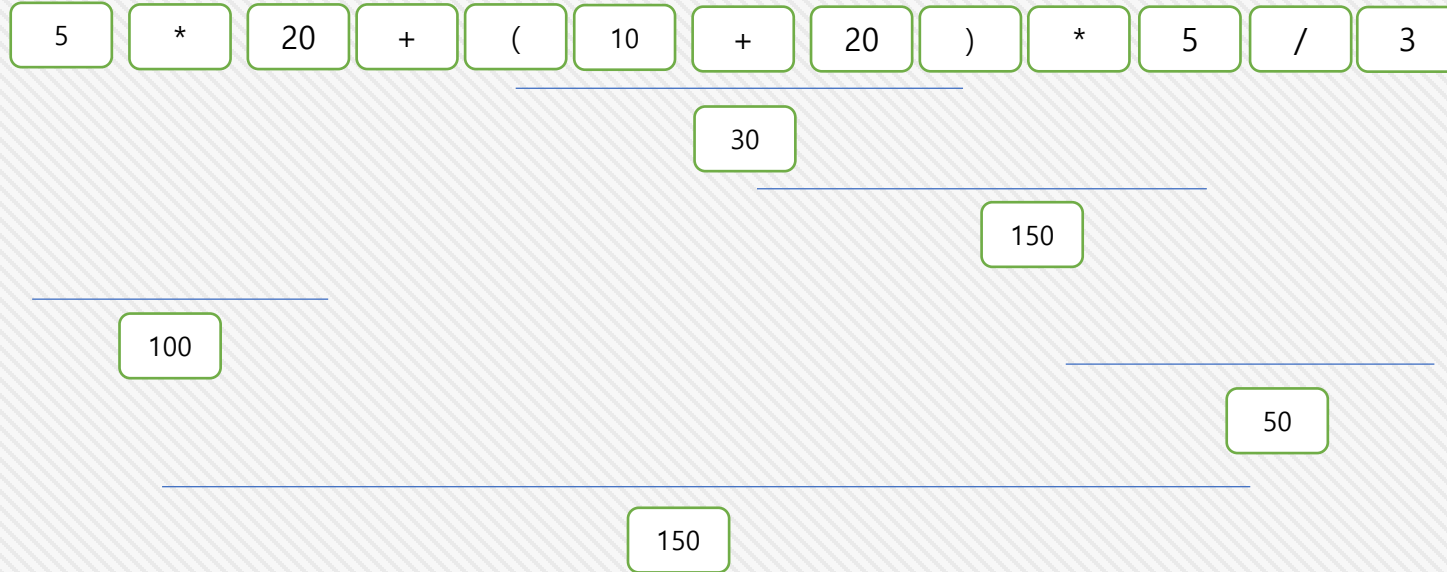
    printf( "data > 5 || data++ = %d\n", data > 5 || data++);
    printf( "data = %d\n", data);
    return 0;
}
```

data > 5 || data++ = 1
data = 7

9. 연산자 우선 순위

연산자 우선 순위

- 하나의 수식에 연산자를 여러 개 사용 하는 경우 어떤 연산자를 우선 실행 하는지 결정



```
#include <stdio.h>

int main() {
    printf( "%d\n", 5*20+(10+20)*5/3);
    return 0;
}
```

```
C:\WINDOWS\system32\cmd.exe
150
Press any key to continue . . .
```

9. 연산자 우선 순위

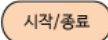

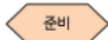

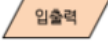

연산자 우선 순위

순위	종류	연산자	연산방향
1	괄호, 배열, 구조체	()(후위증가).[] [후위감소]->	->
2	단항 연산자	(자료형) *(간접) &(주소) !(not) ~(비트) ++ -- +(부호) -(부호) sizeof	<-
3	승제 연산자	* / %	->
4	차감 연산자	+ -	->
5	시프트연산자	<< >>	->
6	비교 연산자	< <= > >=	->
7	등가 연산자	== !=	->
8	비트 연산자 AND	&	->
9	비트 연산자 XOR	^	->
10	비트 연산자 OR		->
11	논리 연산자 AND	&&	->
12	논리 연산자 OR		->
13	조건 연산자	? :	<-
14	대입 연산자	= *= /= += -= %= <<= >>= &= ^= =	<-
15	나열 연산자	,	->

- 우선 순위가 같은 연산자는 연산 방향 우선

순서도

- 순서도(flowchart)란 어떠한 일을 처리하는 과정을 순서대로 간단한 기호와 도형으로 도식화한 것을 의미
- 프로그래밍 전반에 걸쳐 분석, 기획, 디자인, 설계 단계에서 사용

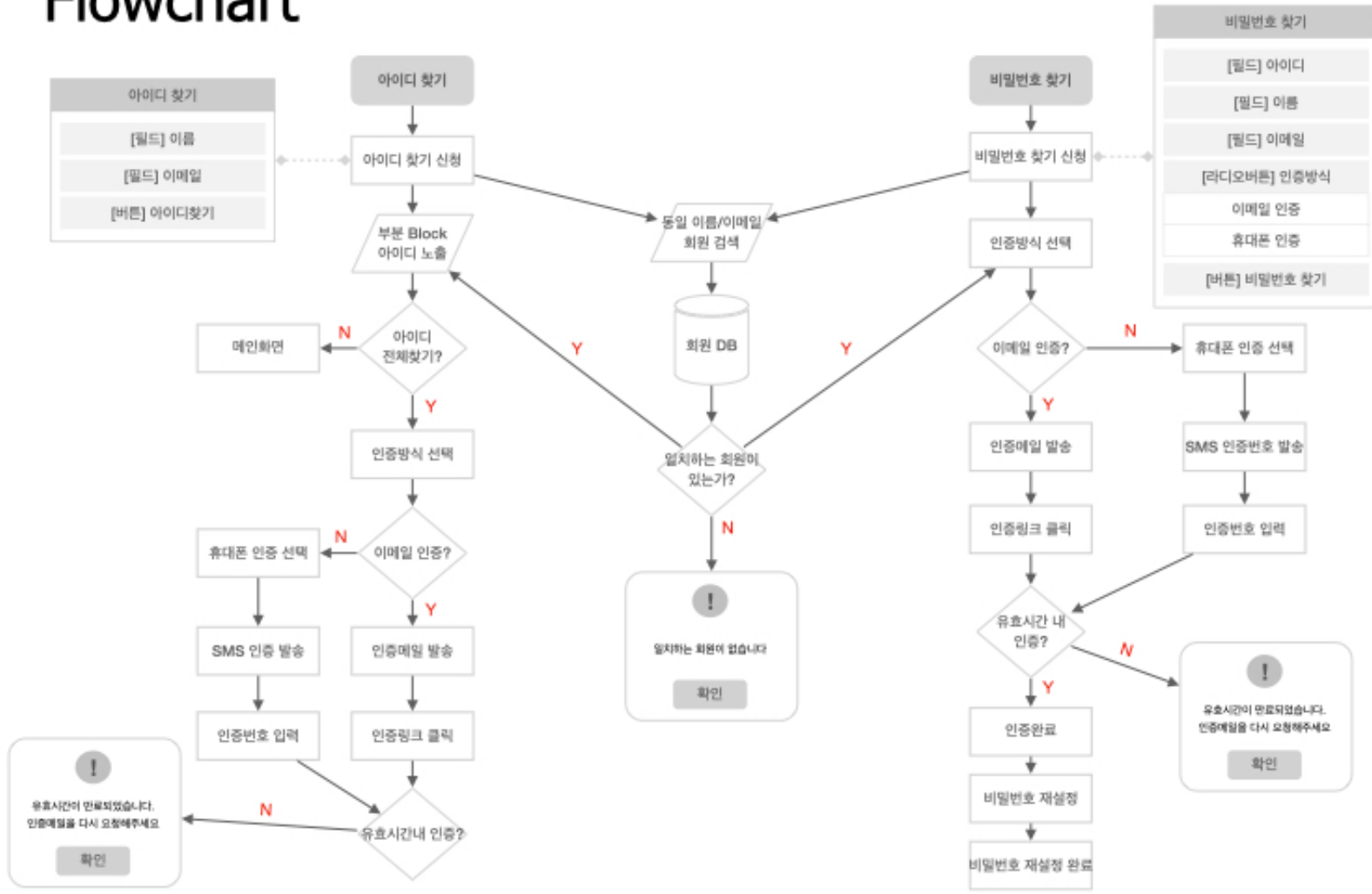
기호	명칭	설명
	단말	순서도의 시작과 끝을 나타냄.
	흐름선	순서도 기호 간의 연결 및 작업의 흐름을 표시함.
	준비	작업 단계 시작 전 해야 할 작업을 명시함.
	처리	처리해야 할 작업을 명시함.
	입출력	데이터의 입출력 시 사용함.
	의사 결정	비교 및 판단에 의한 논리적 분기를 나타냄.
	표시	화면으로 결과를 출력함.

기호	명칭	사용 용도	기호	명칭	사용 용도
	처리	각종 연산, 데이터 이동 등의 처리		터미널	순서도의 시작과 끝 표시
	연결자	흐름이 다른 곳과 연결되는 입출구를 나타냄		천공카드	천공카드의 입출력
	입출력	데이터의 입력과 출력		서류	서류를 매체로 하는 입출력 표시
	흐름선	처리의 흐름과 기호를 연결하는 기능		수동입력	콘솔에 의한 입력
	준비	기억장소, 초기값 등 작업의 준비 과정 나타냄		반복	조건을 만족하면 반복
	미리 정의된 처리	미리 정의된 처리로 옮길 때 사용		디스플레이	결과를 모니터로 나타냄

1. 순서도

4. 프로그램 제어 4-2. 순서도

Flowchart



- 프로그램의 흐름을 제어 할 수 있는 제어문에는 조건문과 반복문이 있다.
- 조건문 : 특정 조건을 부여 하여 조건에 만족 하는 경우와 불 만족하는 경우 문장을 수행함
- 반복문 : 특정 조건을 부여 하여 조건에 따라서 어떤 명령문들을 반복해서 수행함

조건문

- If 조건문
- If ~ else ~ 조건문
- 중첩된 조건문
- switch 조건문

반복문

- for 반복문
- while 반복문
- 중첩 반복문
- break, continue 제어문

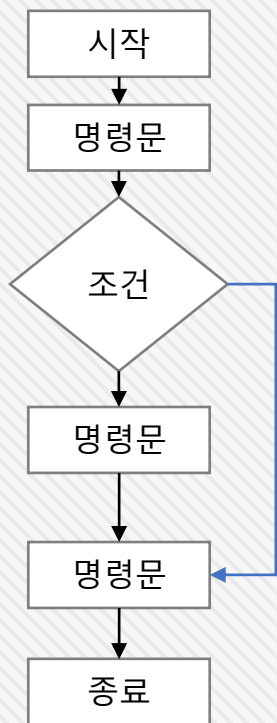
1. if 조건문

If 조건문

- 특정 조건이 만족 할 때 실행하는 문법.

If 조건문 구조

- if (조건 수식) 명령문;
- if (조건 수식) { 명령문들 };



```
int main()
int a = 0;
if ( a > 1 )
a = a + 1
printf(a)
return 0
```

```
#include <stdio.h>

int main() {
    int a = 0;
    if ( a > 1 ) a = a + 1;
    printf("a = %d", a);
    return 0;
}
```

The screenshot shows a Windows command prompt window with the title 'C:\WINDOWS\system32\cmd.exe'. The output displayed is 'a = 0' followed by 'Press any key to continue . . .'. This is because the condition 'a > 1' is false when a is 0, so the increment operation is not performed.

```
#include <stdio.h>

int main() {
    int a = 2;
    if ( a > 1 ) {
        a = a + 1;
    };
    printf("a = %d", a);
    return 0;
}
```

The screenshot shows a Windows command prompt window with the title 'C:\WINDOWS\system32\cmd.exe'. The output displayed is 'a = 3' followed by 'Press any key to continue . . .'. This is because the condition 'a > 1' is true when a is 2, so the increment operation is performed, resulting in a = 3.

1. if 조건문

if 조건문 주의 사항

- 세미콜론 ;의 위치에 따라서 결과는 틀려 짐

```
#include <stdio.h>

int main() {

    int a = 1;
    if ( a > 1 ) ;
        a ++;
    printf("a = %d", a);

    return 0;
}
```

a = 2

```
#include <stdio.h>

int main() {

    int a = 1;
    if ( a > 1 )
        a ++;
    printf("a = %d", a);

    return 0;
}
```

a = 3

- 대입 연산자와 관계 연산자

```
#include <stdio.h>

int main() {

    int a = 1;
    if ( a = 2 ) {
        a ++;
    }
    printf("a = %d", a);

    return 0;
}
```

a = 3

```
#include <stdio.h>

int main() {

    int a = 1;
    if ( a = 2 ) {
        a ++;
    }
    printf("a = %d", a);

    return 0;
}
```

a = 1

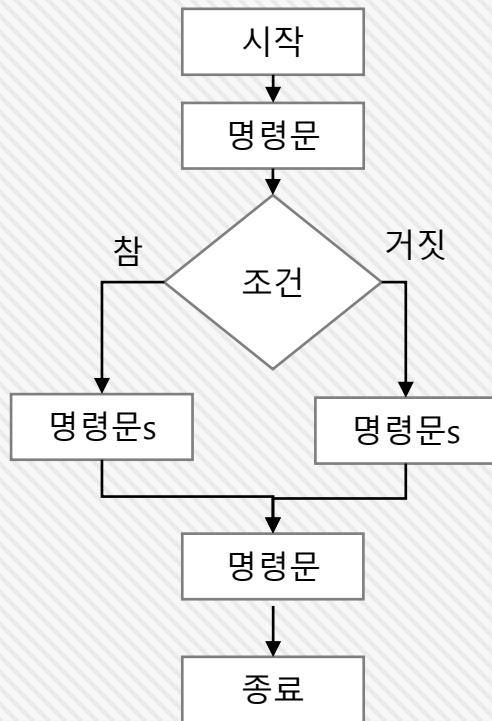
2. if ~ else 조건문

If ~ else 조건문

- 특정 조건이 만족 할 때 명령문 만족 하기 않을 때 명령문 실행

If 조건문 구조

- if (조건 수식) 명령문 else 명령문;
- if (조건 수식) { 명령문들 } else { 명령문들 } ;



```
#include <stdio.h>

int main() {

    int a = 4;
    if ( a < 5 ) a++;
    else a --;
    printf("a = %d", a);

    return 0;
}
```

```
C:\WINDOWS\system32\cmd.exe
a = 5
Press any key to continue . . .
```

```
#include <stdio.h>

int main() {

    int a = 5;
    if ( a < 5 ) a++;
    else a --;
    printf("a = %d", a);

    return 0;
}
```

```
C:\WINDOWS\system32\cmd.exe
a = 4
Press any key to continue . . .
```

```
#include <stdio.h>

int main() {

    int a = 5;
    if ( a < 5 ) {
        a++;
    } else {
        a--;
    }
    printf("a = %d", a);

    return 0;
}
```

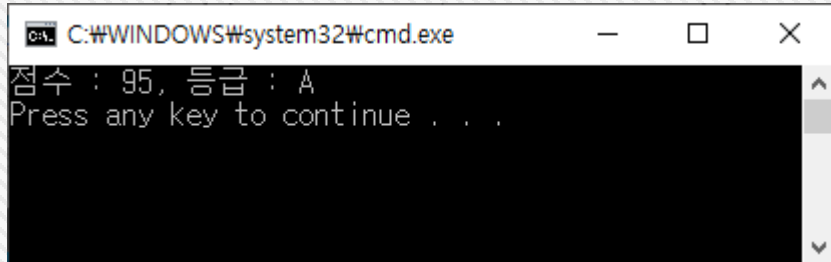
2. if ~ else 조건문

중복 구문은 한번에

```
#include <stdio.h>

int main() {
    int score = 95;
    char grade;
    if ( score >= 90 ) {
        grade = 'A';
        printf("점수 : %d, 등급 : %c", score, grade);
    } else {
        grade = 'B';
        printf("점수 : %d, 등급 : %c", score, grade);
    }

    return 0;
}
```



A screenshot of a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The window shows the output of the program: "점수 : 95, 등급 : A" followed by "Press any key to continue . . .".

```
#include <stdio.h>

int main() {
    int score = 95;
    char grade;
    if ( score >= 90 ) {
        grade = 'A';
    } else {
        grade = 'B';
    }
    printf("점수 : %d, 등급 : %c", score, grade);
    return 0;
}
```

삼항 연산자로 표시

```
#include <stdio.h>

int main() {
    int score = 95;
    char grade;
    grade = ( score >= 90 ) ? 'A' : 'B';
    printf("점수 : %d, 등급 : %c", score, grade);
    return 0;
}
```

2. if ~ else 조건문

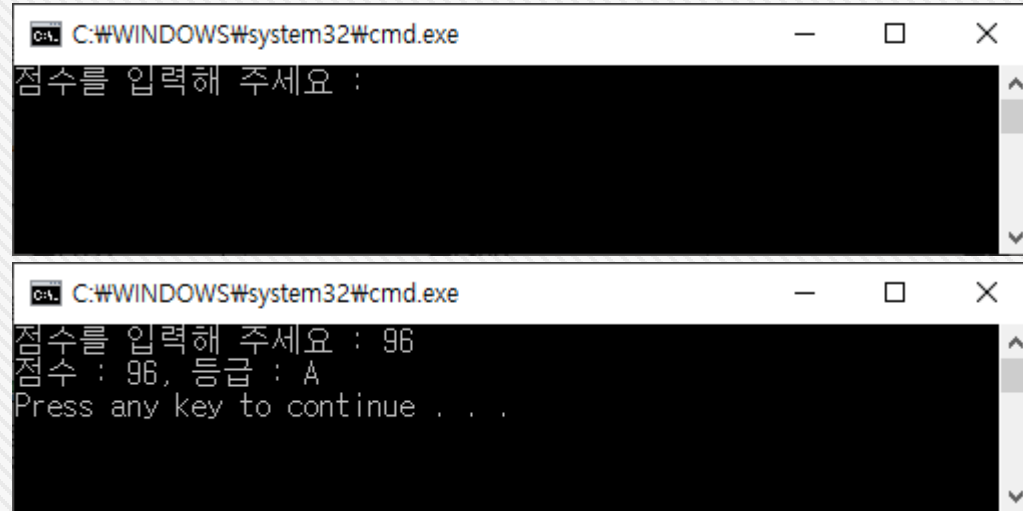
Keyboard로 점수를 입력 하여 등급 출력

```
#include <stdio.h>

int main() {
    int score;
    char grade;

    printf("점수를 입력해 주세요 : ");
    scanf("%d", &score);

    grade = ( score >= 90 ) ? 'A' : 'B';
    printf("점수 : %d, 등급 : %c", score, grade);
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
점수를 입력해 주세요 :

C:\WINDOWS\system32\cmd.exe
점수를 입력해 주세요 : 96
점수 : 96, 등급 : A
Press any key to continue . . .
```

2. if ~ else 조건문

4. 프로그램 제어 4-3. 조건문

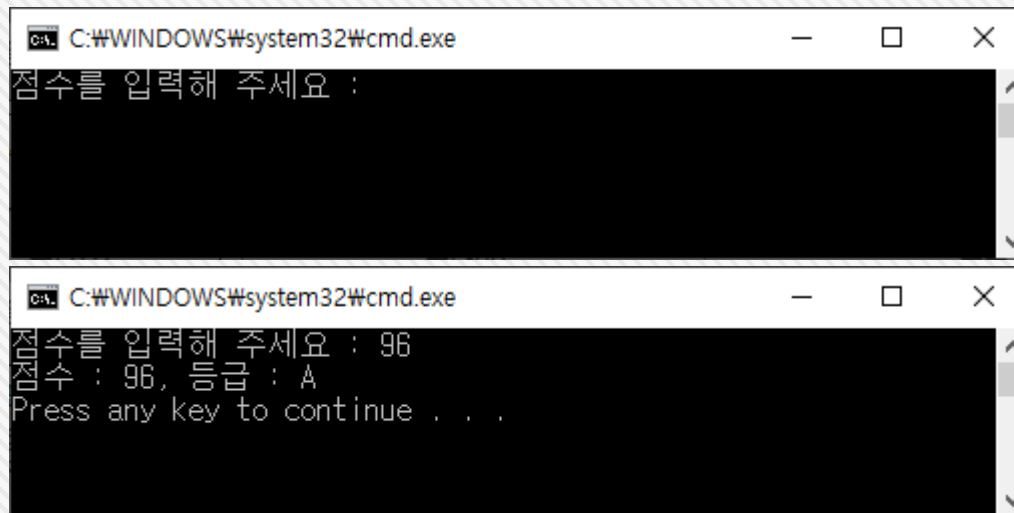
Keyboard로 점수를 입력 하여 등급 출력

```
#include <stdio.h>

int main() {
    int score;
    char grade;

    printf("점수를 입력해 주세요 : ");
    scanf("%d", &score);

    grade = ( score >= 90 ) ? 'A' : 'B';
    printf("점수 : %d, 등급 : %c", score, grade);
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
점수를 입력해 주세요 :

점수를 입력해 주세요 : 96
점수 : 96, 등급 : A
Press any key to continue . . .
```

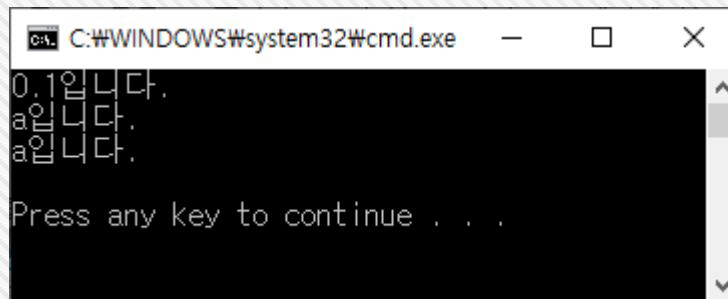
실수, 문자 비교

```
float num1 = 0.1f;
char c1 = 'a';

if (num1 == 0.1f) // 실수 비교
    printf("0.1입니다.\n");

if (c1 == 'a') // 문자 비교
    printf("a입니다.\n");

if (c1 == 97) // 문자를 ASCII 코드로 비교
    printf("a입니다.\n");
```



```
C:\WINDOWS\system32\cmd.exe
0.1입니다.
a입니다.
a입니다.
Press any key to continue . . .
```


3. 중첩된 조건문

4. 프로그램 제어

4-3. 조건문

중첩된 조건문

```
#include <stdio.h>

int main() {
    int score = 75;
    char grade;
    if ( score >= 90 ) {
        grade = 'A';
    } else {
        if (score >= 80) {
            grade = 'B';
        } else {
            if (score >= 70) {
                grade = 'C';
            } else {
                if (score >= 60) {
                    grade = 'D';
                } else {
                    grade = 'F';
                }
            }
        }
    }

    printf("점수 : %d, 등급 : %c", score, grade);
    return 0;
}
```

```
#include <stdio.h>

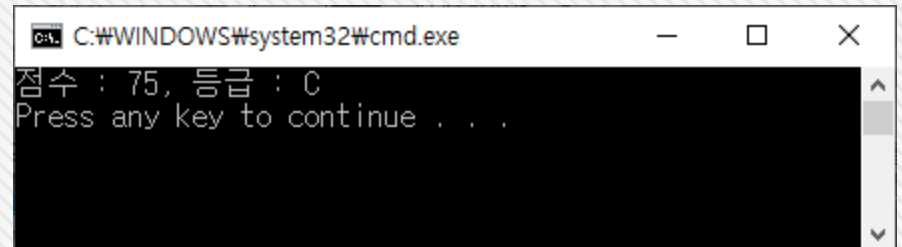
int main() {
    int score = 75;
    char grade;
    if ( score >= 90 ) grade = 'A';
    else
        if (score >= 80) grade = 'B';
        else {
            if (score >= 70) grade = 'C';
            else {
                if (score >= 60) grade = 'D';
                else grade = 'F';
            }
        }

    printf("점수 : %d, 등급 : %c", score, grade);
    return 0;
}
```

```
#include <stdio.h>

int main() {
    int score = 75;
    char grade;
    if ( score >= 90 ) grade = 'A';
    else if (score >= 80) grade = 'B';
    else if (score >= 70) grade = 'C';
    else if (score >= 60) grade = 'D';
    else grade = 'F';

    printf("점수 : %d, 등급 : %c", score, grade);
    return 0;
}
```



A screenshot of a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The window shows the output of the program: "점수 : 75, 등급 : C" followed by "Press any key to continue . . .". The cursor is positioned at the end of the second line.

4. switch

4. 프로그램 제어

4-3. 조건문

switch

- if문의 비효율적인 문제를 해결 하기 위함
- 변수 값이 이미 정해져 있는 상수와 비교 할 때 사용
- break 를 만나면 switch 블록을 벗어남

```
명령문;
switch(수식 또는 변수) {
    case 상수1:
        명령문;
        break;
    case 상수2:
        명령문;
        break;
    default:
        명령문;
        break;
}
명령문;
```

```
#include <stdio.h>

int main() {
    int score = 75;
    char grade;

    switch(score/10) {
        case 10:
        case 9:
            grade = 'A';
            break;
        case 8:
            grade = 'B';
            break;
        case 7:
            grade = 'C';
            break;
        case 6:
            grade = 'D';
            break;
        default:
            grade = 'A';
    }

    printf("점수 : %d, 등급 : %c", score, grade);
    return 0;
}
```

```
#include <stdio.h>

int main() {
    int score = 100;
    char grade;

    switch(score/10) {
        case 10:
            printf("100 입니다.\n");
        case 9:
            grade = 'A';
            break;
        case 8:
            grade = 'B';
            break;
        case 7:
            grade = 'C';
            break;
        case 6:
            grade = 'D';
            break;
        default:
            grade = 'A';
    }

    printf("점수 : %d, 등급 : %c", score, grade);
    return 0;
}
```

반복문

- “1에서 5까지 더하기”에서 1씩 더하는 작업이 반복 작업이 존재 하는 경우 사용 하는 문법
- 시작(1에서) , 종결(5까지), 반복 (1씩 증가), 더하기(명령문) 로 구성됨

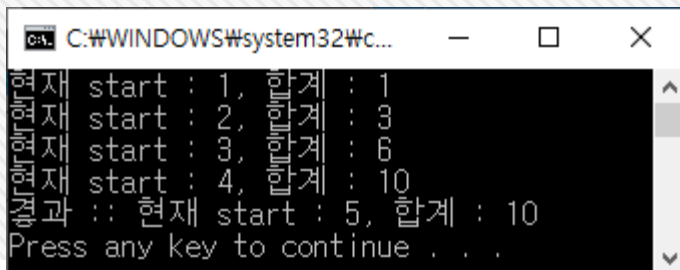
for

- for(시작 조건; 종결조건; 조건변화 수식)

```
#include <stdio.h>

int main() {
    int start, sum = 0;
    for (start = 1; start < 5; start++) {
        sum = sum + start;
        printf("현재 start : %d, 합계 : %d\n", start, sum);
    }

    printf("결과 :: 현재 start : %d, 합계 : %d", start, sum);
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
현재 start : 1, 합계 : 1
현재 start : 2, 합계 : 3
현재 start : 3, 합계 : 6
현재 start : 4, 합계 : 10
결과 :: 현재 start : 5, 합계 : 10
Press any key to continue . . .
```

- 시작 조건이 선언 시점에 할당 되어 있으면 생략 가능

```
#include <stdio.h>

int main() {
    int start = 1, sum = 0;
    for (; start < 5; start++) {
        sum = sum + start;
        printf("현재 start : %d, 합계 : %d\n", start, sum);
    }

    printf("결과 :: 현재 start : %d, 합계 : %d", start, sum);
    return 0;
}
```

for

- for문 안에 사용 하는 변수가 선언만 되어 있으며 시작조건에 할당 가능함
- for문을 무한 루프 형식과 break를 이용

```
#include <stdio.h>

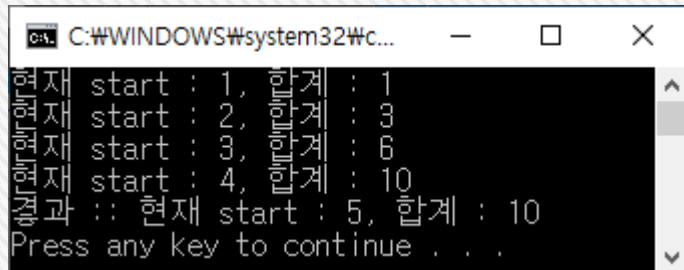
int main() {
    int start, sum;
    for (start = 1, sum = 0; start < 5; start++) {
        sum = sum + start;
        printf("현재 start : %d, 합계 : %d\n", start, sum);
    }

    printf("결과 :: 현재 start : %d, 합계 : %d", start, sum);
    return 0;
}
```

```
#include <stdio.h>

int main() {
    int start = 1, sum = 0;
    for (;;) {
        sum = sum + start;
        printf("현재 start : %d, 합계 : %d\n", start, sum);
        start++;
        if (start > 4) break;
    }

    printf("결과 :: 현재 start : %d, 합계 : %d", start, sum);
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
현재 start : 1, 합계 : 1
현재 start : 2, 합계 : 3
현재 start : 3, 합계 : 6
현재 start : 4, 합계 : 10
결과 :: 현재 start : 5, 합계 : 10
Press any key to continue . . .
```

2. while

while

- 종결 조건만으로도 반복문 실행
- while(종결 조건) {
 명령문 s
}

```
#include <stdio.h>

int main() {
    int start = 1, sum = 0 ;
    while (start < 5) {
        sum = sum + start;
        printf("현재 start : %d, 합계 : %d\\n", start, sum);
        start ++;
    }

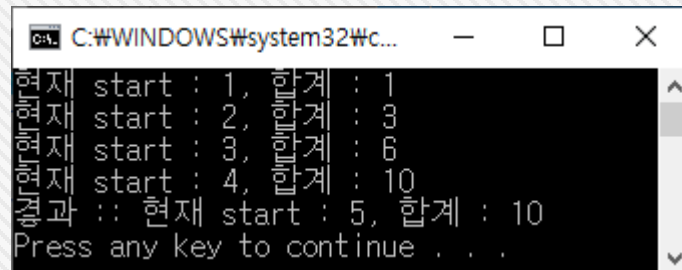
    printf("결과 :: 현재 start : %d, 합계 : %d", start, sum);
    return 0;
}
```

- while문을 무한 루프 형식과 break를 이용

```
#include <stdio.h>

int main() {
    int start = 1, sum = 0 ;
    while (1) {
        sum = sum + start;
        printf("현재 start : %d, 합계 : %d\\n", start, sum);
        start ++;
        if (start >= 5) break;
    }

    printf("결과 :: 현재 start : %d, 합계 : %d", start, sum);
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
현재 start : 1, 합계 : 1
현재 start : 2, 합계 : 3
현재 start : 3, 합계 : 6
현재 start : 4, 합계 : 10
결과 :: 현재 start : 5, 합계 : 10
Press any key to continue . . .
```

3. do ~ while

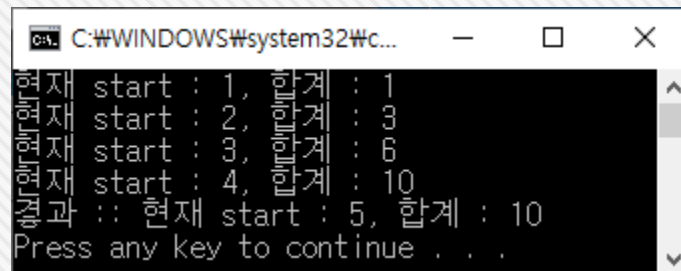
do ~ while

- 종결 조건만으로도 반복문 실행하는 것은 while문과 동일 하지만 무조건 한번 실행
- do {
 명령문 s
} while(종결 조건)

```
#include <stdio.h>

int main() {
    int start = 1, sum = 0 ;
    do {
        sum = sum + start;
        printf("현재 start : %d, 합계 : %d\n", start, sum);
        start ++;
    } while ( start < 5 );

    printf("결과 :: 현재 start : %d, 합계 : %d", start, sum);
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
현재 start : 1, 합계 : 1
현재 start : 2, 합계 : 3
현재 start : 3, 합계 : 6
현재 start : 4, 합계 : 10
결과 :: 현재 start : 5, 합계 : 10
Press any key to continue . . .
```

4. break, continue

4. 프로그램 제어

4-4. 반복문

break

- 반복문을 벗어나기 위함
- 하나의 블록만 벗어남
- 구구단 출력

```
#include <stdio.h>
int main() {
    int start , second ;
    for ( start = 2 ; start < 10; start++) {
        printf("%d 구단\n", start );
        for ( second = 1 ; second < 10; second++) {
            printf("%d * %d = %d\n", start, second, start * second);
        }
    }
    return 0;
}
```

- 구구단 출력 – 각각의 구구단 2단 3 까지만 출력

- 구구단 출력 – 각각의 구구단 3 까지만 출력

```
#include <stdio.h>

int main() {
    int start , second ;
    for ( start = 2 ; start < 10; start++) {
        printf("%d 구단\n", start );
        for ( second = 1 ; second < 10; second++) {
            printf("%d * %d = %d\n", start, second, start * second);
            if ( second > 2 ) break;
        }
    }
    return 0;
}
```

```
#include <stdio.h>

int main() {
    int start , second ;
    for ( start = 2 ; start < 10; start++) {
        printf("%d 구단\n", start );
        for ( second = 1 ; second < 10; second++) {
            printf("%d * %d = %d\n", start, second, start * second);
            if ( second > 2 ) break;
        }
        if ( start <= 2 ) break;
    }
    return 0;
}
```

4. break, continue

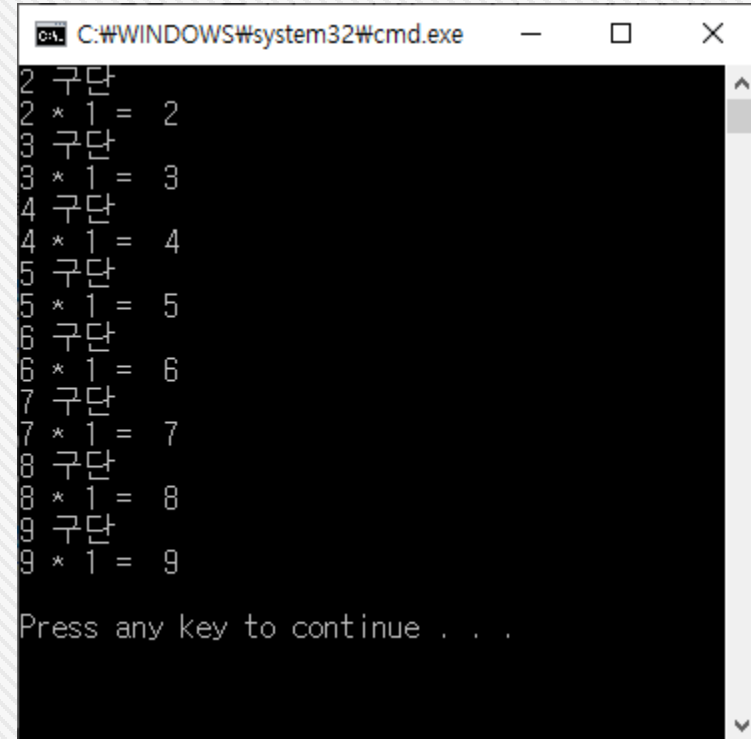
continue

- 1회성 최소
 - 구구단 출력 - 각각의 구구단 1만 출력

```
#include <stdio.h>

int main() {
    int start , second ;
    for ( start = 2 ; start < 10; start++) {
        printf("%d 구단\n", start );
        for ( second = 1 ; second < 10; second++) {
            if ( second > 1 ) continue;
            printf("%d * %d = %d\n", start, second, start * second);
        }
    }

    return 0;
}
```



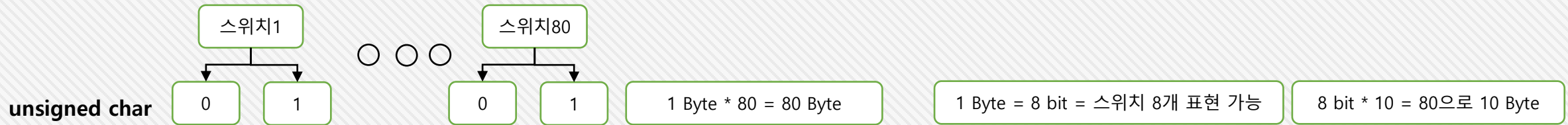
```
C:\WINDOWS\system32\cmd.exe
2 구단
2 * 1 = 2
3 구단
3 * 1 = 3
4 구단
4 * 1 = 4
5 구단
5 * 1 = 5
6 구단
6 * 1 = 6
7 구단
7 * 1 = 7
8 구단
8 * 1 = 8
9 구단
9 * 1 = 9

Press any key to continue . . .
```


1. 비트 연산

비트 연산이 필요한 이유

- 메모리 절약.
- 컴퓨터는 0 또는 1로 저장 하고 처리를 하는데 C언어의 최소 저장 공간은 1Byte (8 bit) 임.
- 만약 전원의 on/off를 표시 하기 위해서는 1byte가 필요 한데 전원 스위치가 80개 이면 80byte가 필요 한데 이것을 bit로 처리 하게 하며 10byte로 처리 가능 하므로 메모리를 절약 할 수 있음



- C언어는 2진수 상수를 제공하는 방법이 없으므로 16진수를 사용 하여 프로그램 함.

2 진수	16 진수	2 진수	16 진수	2 진수	16 진수	2 진수	16 진수
0000	0	0100	4	1000	8	1100	C (12)
0001	1	0101	5	1001	9	1101	D (13)
0010	2	0110	6	1010	A (10)	1110	E (14)
0011	3	0111	7	1011	B (11)	1111	F (15)

unsigned char
= 1 Byte

0 (7 bit)	1 (6 bit)	0 (5 bit)	1 (4 bit)	1 (3 bit)	0 (2 bit)	1 (0 bit)	1 (0 bit)
5				B			

C 언어 표현 : unsigned char a = 0x5B ; => 십진수 : $5 * (16^1) + 11 * 16^0 = 91$

1. 시프트 연산자

시프트 연산자

- << (오른쪽 에서 왼쪽), >> (왼쪽에서 오른쪽)을 사용 하여서 지정한 비트 수 만큼 이동

data = data << 2

부호 없는 값

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

0x22

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

overflow data 버려 짐

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

0x88

$$8 * 16^1 + 8 * 16^0 = 136$$

반공간 0으로 채워 짐

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

```
void main() {
```

```
    unsigned char data = 0x22;
```

```
    printf("십진수 : %d, 16진수 : %#x \n", data, data);
```

```
    data = data << 2;
```

```
    printf("십진수 : %d, 16진수 : %#x \n", data, data);
```

```
}
```

```
십진수 : 34, 16진수 : 0x22
십진수 : 136, 16진수 : 0x88
```

부호 있는 값

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

0x22

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

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

0x88

$$8 * 16^1 - 8 * 16^0 = 120$$

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

```
void main() {
```

```
    char data = 0x22;
```

```
    printf("십진수 : %d, 16진수 : %#x \n", data, data);
```

```
    data = data << 2;
```

```
    printf("십진수 : %d, 16진수 : %#x \n", data, data);
```

```
}
```

```
십진수 : 34, 16진수 : 0x22
십진수 : -120, 16진수 : 0xfffff88
```

- 음수로 변화는 경우 가 발생 => 예측 할 수 없는 값 발생 => 사용 하지 말 아함

2. 곱셈/나눗셈

4. 프로그램 제어 4-6. 시프트연산자

시프트 연산자

• << (오른쪽 에서 왼쪽)

0	0	0	0	0	0	0	1	0x01	1	* 2 ⁿ
0	0	0	0	0	0	1	0	0x02	2	
0	0	0	0	0	1	0	0	0x04	4	
0	0	0	0	1	0	0	0	0x08	8	
0	0	0	1	0	0	0	0	0x10	16	
0	0	1	0	0	0	0	0	0x20	32	
0	1	0	0	0	0	0	0	0x40	64	
1	0	0	0	0	0	0	0	0x80	128	

• >> 왼쪽에서 오른쪽)

1	0	0	0	0	0	0	0	0x80	128	/ 2 ⁿ
0	1	0	0	0	0	0	0	0x40	64	
0	0	1	0	0	0	0	0	0x20	32	
0	0	0	1	0	0	0	0	0x10	16	
0	0	0	0	1	0	0	0	0x08	8	
0	0	0	0	0	1	0	0	0x04	4	
0	0	0	0	0	0	1	0	0x02	2	
0	0	0	0	0	0	0	1	0x01	1	
0	0	0	0	0	0	0	0	0x00	0	

```
#include <stdio.h>

void main() {
    unsigned char data02 = 4; // 0x04

    unsigned char data02Result = data02 << 2;
    printf("십진수 : %d, data02 >> 2 : %d \n", data02, data02Result);
}
```

십진수 : 4, data02 >> 2 : 16

```
#include <stdio.h>

void main() {
    unsigned char data03 = 128; // 0x80;

    unsigned char data03Result = data03 >> 2;
    printf("십진수 : %d, data03 >> 2 : %d \n", data03, data03Result);
}
```

십진수 : 128, data03 >> 2 : 32

1. Bit 연산자

bit 연산자

- bit 단위로 연산을 할 때 사용
- 논리 연산자와 구별 하기 위해서 AND 연산 `&`, OR 연산 `|`, NOT 연산 `~`, XOR 연산 `^` 로 표시

연산자	설명
<code>&</code>	비트 AND
<code> </code>	비트 OR
<code>^</code>	비트 XOR (배타적 OR, Exclusive OR)
<code>~</code>	비트 NOT

A	B	AND(&)	OR()	XOR(^)	NOT (~)
0	0	0	1	0	1
0	1	0	1	1	
1	0	0	1	1	0
1	1	1	0	0	

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

```
void main() {
```

```
    unsigned char a = 0x23;
```

```
    unsigned char b = 0x42;
```

```
    unsigned char c = a & b;
```

```
    unsigned char d = a | b;
```

```
    unsigned char e = a ^ b;
```

```
    unsigned char f = ~a;
```

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

```
    printf("AND(a & b) = c : %d (%#x) \n", c,c);
```

```
    printf("OR(a & b) = c : %d (%#x) \n", d,d);
```

```
    printf("XOR(a & b) = c : %d (%#x) \n", e,e);
```

```
    printf("NOT(~a) = c : %d (%#x) \n", f,f);
```

```
}
```

```
a (0x23): 35, b (0x42) : 66
AND(a & b) = c : 2 (0x2)
OR(a & b) = c : 99 (0x63)
XOR(a & b) = c : 97 (0x61)
NOT(~a) = c : 220 (0xdc)
```

	10진수	16진수	값 (1 byte)	
a	35	0x23	0010	0011
b	66	0x42	0100	0010
AND	2	0x2	0000	0010
OR	99	0x63	0110	0011
XOR	97	0x61	0110	0001
NOT	220	0xdc	1101	1100

2. Bit 연산자 + 할당 연산자

bit 연산자

- 할당 연산자와 혼합 해서 사용 할 수 있음

&=	비트 AND 연산 후 할당
=	비트 OR 연산 후 할당
^=	비트 XOR 연산 후 할당
<<=	비트를 왼쪽으로 시프트한 후 할당
>>=	비트를 오른쪽으로 시프트한 후 할당

```
C:\WINDOWS\system32\cmd.exe
AND (a &= 5)   = a : 0      (0)
OR  (b |= 5)   = b : 71     (0)
XOR (c ^= 5)   = c : 71     (0x47)
<< (d <<= 5)   = d : 64     (0x40)
>> (e >>= 5)   = e : 2      (0x2)

Press any key to continue . . .
```

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

```
void main() {
```

```
    unsigned char a = 0x42;
```

```
    unsigned char b = 0x42;
```

```
    unsigned char c = 0x42;
```

```
    unsigned char d = 0x42;
```

```
    unsigned char e = 0x42;
```

```
    a &= 5;
```

```
    b |= 5;
```

```
    c ^= 5;
```

```
    d <<= 5;
```

```
    e >>= 5;
```

```
    printf("AND (a &= 5)   = a : %d \t(%#x) \n", a, a);
```

```
    printf("OR  (b |= 5)   = b : %d \t(%#x) \n", b, a);
```

```
    printf("XOR (c ^= 5)   = c : %d \t(%#x) \n", c, c);
```

```
    printf("<< (d <<= 5)   = d : %d \t(%#x) \n", d, d);
```

```
    printf(">> (e >>= 5)   = e : %d \t(%#x) \n", e, e);
```

```
}
```

3. Bit 연산자 활용

iot 예제

- iot 장비에서 8개의 장비에서 전원 on/off 신호가 전달 된다. On :1, Off: 0 전원이 켜져 있는 장비를 표시 하라.

8번 장비	7번 장비	6번 장비	5번 장비	4번 장비	3번 장비	2번 장비	1번 장비
0	1	0	0	1	1	1	0

```
#include <stdio.h>

void main() {
    unsigned char deviceValue ;

    printf("16 진수 문자를 입력 하세요 : ");
    scanf("%x", &deviceValue) ;
    printf("\n입력 값 : %#x \n", deviceValue);
    if (deviceValue & 1) {
        printf ( " 1 번 장비 on , %#x\n", deviceValue & 1);
    };

    if (deviceValue & 2) {
        printf ( " 2 번 장비 on , %d\n", deviceValue & 2);
    };

    if (deviceValue & 4) {
        printf ( " 3 번 장비 on , %d\n", deviceValue & 4);
    };

    .....
}
```