

CS SCHOOL

if...else 구문

```
if (num > 4) 만약 ~이라면  
{  
    printf("%d는 4보다 크군요!", num);  
}
```

```
if (num < 5) 만약 ~이라면  
{  
    printf("%d는 5보다 작군요!", num);  
}  
else if (num == 5) 위 조건은 아니지만 ~이라면  
{  
    printf("%d는 5군요!", num);  
}  
else 위에 있는 조건들이 다 아니라면  
{  
    printf("%d는 5보다 클 수 밖ですよ!", num);  
}
```

삼항 연산자

조건 ? A : B

조건이 참이면 A 반환
조건이 거짓이면 B 반환

```
int x = (num > 10) ? 1 : 0;
```

조건

참이면 1 반환
거짓이면 0 반환

```
int x = (num > 10) ? func(1) : func(0);
```

함수도 올 수 있다.

switch 구문

```
if (n == 1)
{
    result = a + b;
    printf("%d", result);
}
else if (n == 2)
{
    result = a - b;
    printf("%d", result);
}
else if (n == 3)
{
    result = a * b;
    printf("%d", result);
}
else if (n == 4)
{
    result = a / b;
    printf("%d", result);
}
else
{
    printf("Wrong Input");
}
```



분기가 너무 많아!!

```
switch (n)
{
    case 1:
        result = a + b;
        printf("%d", result);
        break;
    case 2:
        result = a - b;
        printf("%d", result);
        break;
    case 3:
        result = a * b;
        printf("%d", result);
        break;
    case 4:
        result = a / b;
        printf("%d", result);
        break;
    default:
        printf("Wrong Input");
}
```

switch 구문

1. 정수

```
switch (n)
{
    case 1:
        result = a + b;
        printf("%d", result);
        break;
    case 2:
        result = a - b;
        printf("%d", result);
        break;
    case 3:
        result = a * b;
        printf("%d", result);
        break;
    case 4:
        result = a / b;
        printf("%d", result);
        break;
    default:
        printf("Wrong Input");
}
```

2. case 레이블

3. break가 없다면 해당 레이블부터 아래로
쭉 실행된다!!!

4. default : 이도 저도 아니면 실행

switch 구문

```
if (n == 3 || n == 5 || n == 7 || n == 9)
    printf("Today's Happy Number!");
else if (n == 2 || n == 4 || n == 8 || n == 10)
    printf("Today's Sad Number!");
else if (n == 11 || n == 22 || n == 33)
    printf("Today's Good Number!");
else
    printf("Today's Bad number!");
```

```
switch (n)
{
    case 3: case 5: case 7: case 9:
        printf("Today's Happy Number!");
        break;
    case 2: case 4: case 8: case 10:
        printf("Today's Sad Number!");
        break;
    case 11: case 22: case 33:
        printf("Today's Good Number!");
        break;
    default:
        printf("Today's Bad number!");
}
```

→ 둘 이상의 레이블을 함께 둘 수 있다.

while 구문

참일 때만 실행

```
while (n > 10)
{
    printf("%d는 10 보다 크구려!", n);
    n--;
}
```

무한 루프

무조건 참

```
while (1)
{
    printf("무한 루프닷!");
}
```


while 구문
: continue와 break;

Break : 반복문을 빠져나온다.

```
hp = 10.0f;

while (1)
{
    if (hp > 5.5f)
        break;
}

printf("%f", hp);
```

A red curved arrow originates from the 'break;' statement inside the while loop and points to the 'printf' statement outside the loop, illustrating that the loop is exited.


Continue : 조건 검사로 이동

```
hp = 10.0f;

while (1)
{
    if (hp > 5.5f)
        continue;

    hp -= 0.5;
}

printf("%f", hp);
```

A red curved arrow originates from the 'continue;' statement inside the while loop and points back to the start of the loop body, just after the opening curly brace, illustrating that the loop continues from the beginning of the next iteration.

do ~ while 구문

```
do
{
    printf("원하는 체력을 입력하시오 : ");
    scanf("%f", &hp);
} while (hp < 0.0f);

printf("%f", hp);
```

일단 실행하고
그 결과에 따라 다시 실행할지 말지를
결정해야 하는 경우

for 구문

반복횟수를 세기 위한 변수
iterator

조건식

증감식

```
for (int i = 0; i < 10; i++)  
{  
    printf("%d번 째 반복", i);  
}
```

for 구문

초기식, 조건식, 증감식 모두 생략 가능
단, 조건식이 빠지면 무한 루프!!!

```
int i = 0;
for (; (i++) < 10;)
    printf("%d번 째 반복\n", i);
```

조건식만 있는 경우

```
for (int i = 0, j = 10; i != j; i++, j--){
    printf("[%d, %d]", i, j);
}
```

coma 연산자의 사용

```
for ( ; ; )
{
}
}
```

무한 루프

배열과 포인터 : 배열

1. 선언방법 및 접근

`int arr[5];` 자료형 배열이름[배열길이]

```
int arr[5];  
  
arr[0] = 1;  
arr[1] = 2;  
arr[2] = 3;  
arr[3] = 4;  
arr[4] = 5;
```

배열 선언

배열 접근

→ 인덱스(INDEX)

Arr[0]

Arr[1]

Arr[2]

Arr[3]

Arr[4]

Arr[5]

4byte

4byte

4byte

4byte

4byte

4byte

배열과 포인터 : 배열

2. 선언 및 초기화

```
int arr1[5] = { 1, 2, 3, 4, 5 };
```

```
int arr2[5] = { 1, 2, 3 };
```

나머지 부분 : 0으로 채움

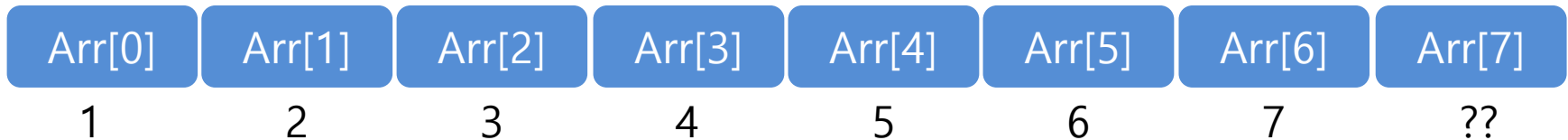
```
int arr3[] = { 1, 2, 3 };
```

개수에 맞게 배열 길이 설정

배열과 포인터 : 배열

3. 배열과 메모리

```
int arr[7] = { 1, 2, 3, 4, 5, 6, 7 };  
printf("%d", arr[7]); //어떤 값이 나올까?
```



- Arr가 시작하는 메모리 주소부터 차례로 int형 값을 읽어온다
- ➔ 다시 말해 **index를 통해 메모리 공간에 접근**한다!!
 - ➔ 그렇다면 index 7로 접근하면 어떤 값을 읽어올까?

배열과 포인터 : 포인터

4. 포인터 이 두 가지 정보를 가지지 않으면 포인터가 아닙니다.

1) 주소값 : ex) 0x4f3a5b22(가상주소공간)

2) 가리키는 자료형 ex) int형, float형, double형

int * ptr;



int형 포인터 : int나 float처럼 하나의 자료형으로 취급합니다.

배열과 포인터 : 포인터

5. 포인터를 얻기 위한 &연산자

```
int * ptr;  
int num = 5;  
  
ptr = &num;  
  
printf("%d\n", *ptr);
```



num이라는 메모리 공간을 가리키는 포인터는
int형 포인터!

배열과 포인터 : 포인터

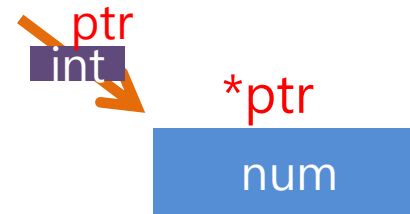
6. 포인터가 가리키는 메모리 공간의 값 : * 연산자

```
int * ptr;
int num = 5;

ptr = &num;

(*ptr) += 5;

printf("%d\n", *ptr);
```



포인터가 가리키는 공간에 접근하는 연산자!

마치 num인 것 처럼!!

메모리 공간에 접근.....
어디서 들어본 것 같은데....?!?!

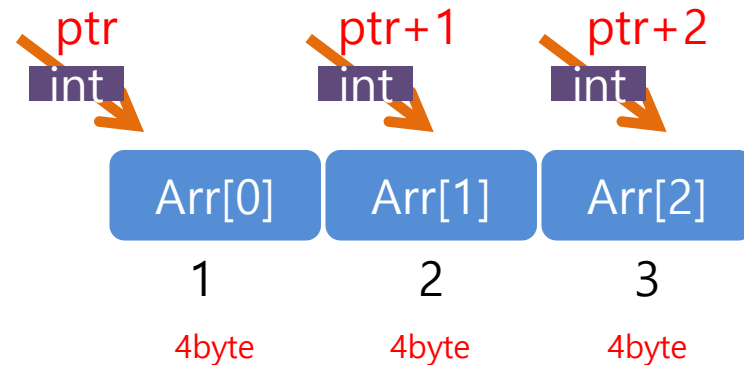
배열과 포인터 : 포인터

7. 포인터의 덧셈

```
int * ptr;
int arr[3] = { 1, 2, 3 };


ptr = &arr[0];
printf("%d \n", *ptr);

ptr = ptr + 1;
printf("%d \n", *ptr);
```



int형이므로 4byte 씩 주소값을 이동
만약 double형 이었다면 8byte씩 이동

ptr : 0x01
ptr + 1 : 0x05
ptr + 2 : 0x09



배열과 포인터 : 포인터

8. 배열과 포인터

```
int * ptr;  
  
int arr[7] = { 1, 2, 3, 4, 5, 6, 7 };  
  
ptr = &arr[0];  
  
for (int i = 0; i < 7; i++)  
    printf("arr[%d] = %d \n", i, arr[i]);  
  
for (int i = 0; i < 7; i++)  
    printf("arr[%d] = %d \n", i, *(ptr + i));
```

arr[i] == *(ptr+i)

배열과 포인터 모두 메모리 공간에 접근하는 방법

배열과 포인터 : 포인터

9. 포인터 정리

```
int * intPtr; //4byte
float * floatPtr; //4byte

int num = 0x418a0000;

intPtr = &num;
floatPtr = &num;

printf("%d\n", *intPtr);
printf("%f\n", *floatPtr);
```

포인터는
주소값만 말하는 게 아니라
데이터의 자료형도 담고 있다!