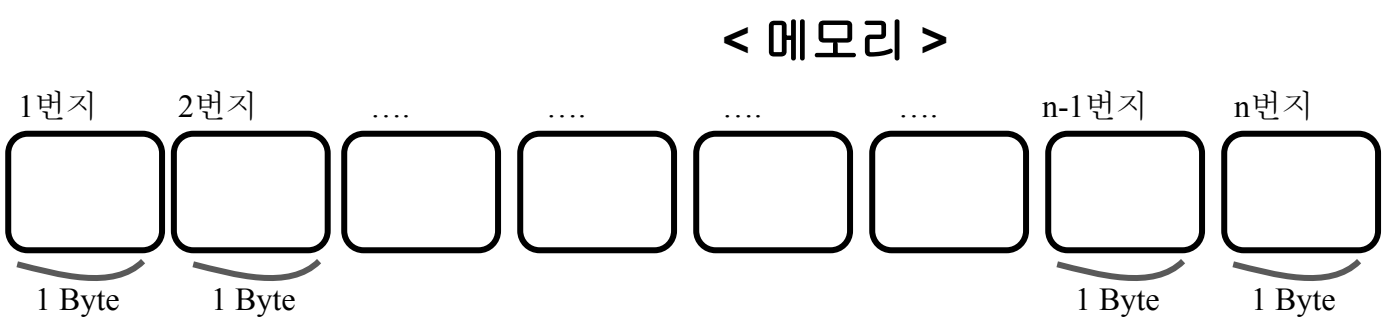
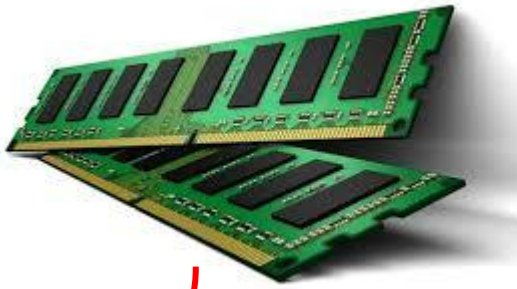
< 메모리 >



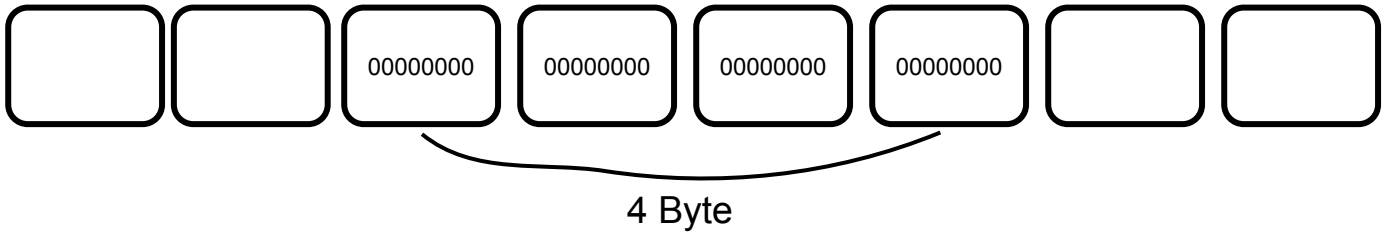


자료형(data type)		할당되는 메모리 크기	표현 가능한 데이터의 범위
정수형	char	1 바이트	-128 ~ +127
	short	2 바이트	-32768 ~ + 32767
	int	4 바이트	-2147483648 ~ + 2147483647
	long	4 바이트	-2147483648 ~ + 2147483647
실수형	float	4 바이트	$3.4 \times 10^{-37} \sim 3.4 \times 10^{+38}$
	double	8 바이트	$1.7 \times 10^{-307} \sim 1.7 \times 10^{+308}$
	long double	8 바이트 혹은 그 이상	차이를 많이 보임

1. 컴퓨터가 갖고있는 메모리가 1바이트 단위로 나뉘어짐
2. 바이트 단위로 나뉘어진 용량은, 주소를 통해 접근할 수 있음 (1번지당 1바이트)
3. 변수를 선언할때, 임의의 메모리에 자료형 크기만큼의 값이 할당됨

int num;

num = 0;



int num 선언
-> 임의의 메모리 주소에 4바이트가 할당되며, 변수명과 해당 메모리 영역이 링크된다

num = 0 대입
-> num 이 갖고있는 32개의 비트(bit)들을 통해 값 0을 만든다

< 변수의 선언 >

컴파일러에게 변수의 존재를 명시해주는 행위

컴파일러는 임의의 메모리를 할당하고, 변수와 해당 메모리를 링크시킨다

```
int iNum;
double dNum;
char ch;
```

< 변수의 정의 >

컴파일러에게 변수의 내용(값)을 결정하는 행위

대입연산(=) 을 사용하기때문에 대입한다고도 한다

```
iNum = 10;        // int형
dNum = 3.14;      // double형
ch = ‘a’;         // char형
```

< 함수의 선언 >

컴파일러에게 함수의 존재를 명시해주는 행위

함수가 정의된 위치가 호출된 위치보다 아래에 있을때 발생하는 오류를 방지한다

```
void func();

int main()
{
        func();
}

void func()
{
        …
}
```

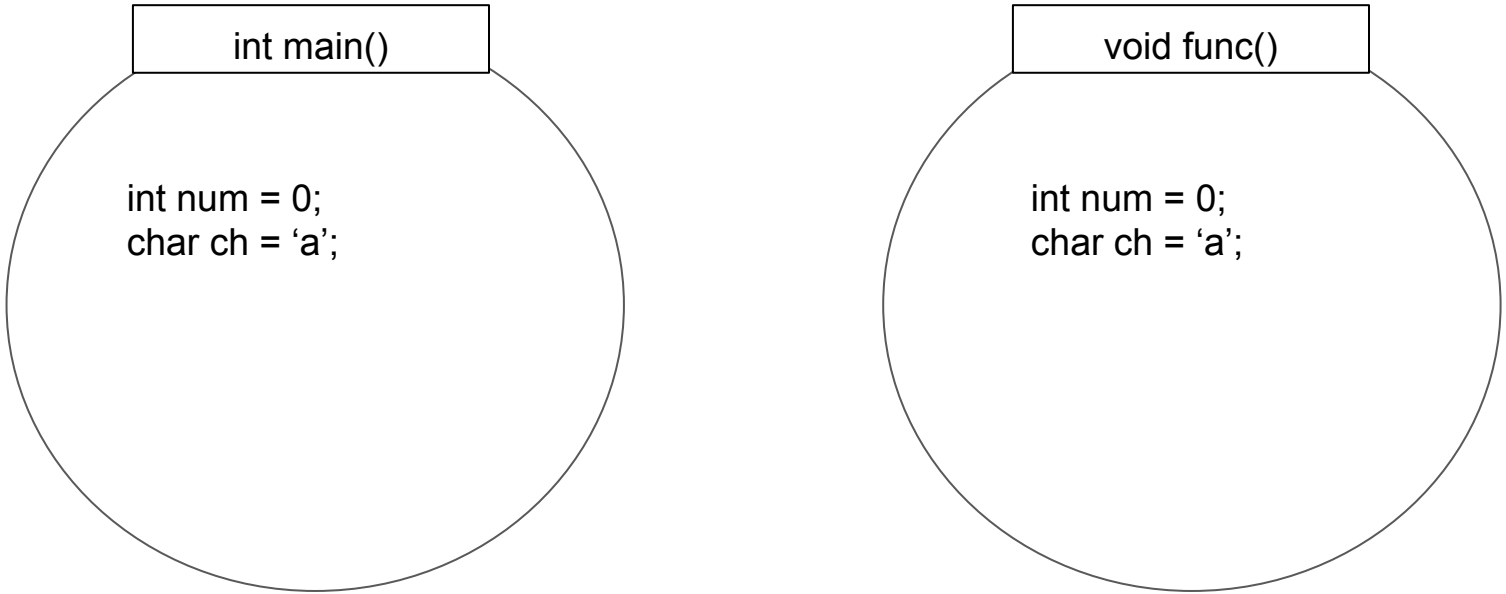
```
int main()
{
        func();
}

void func()
{
        …
}
```

< 함수의 정의 >

컴파일러에게 함수의 내용을 결정하는 행위

```
void func()
{
        printf(“This is func”);
}
```

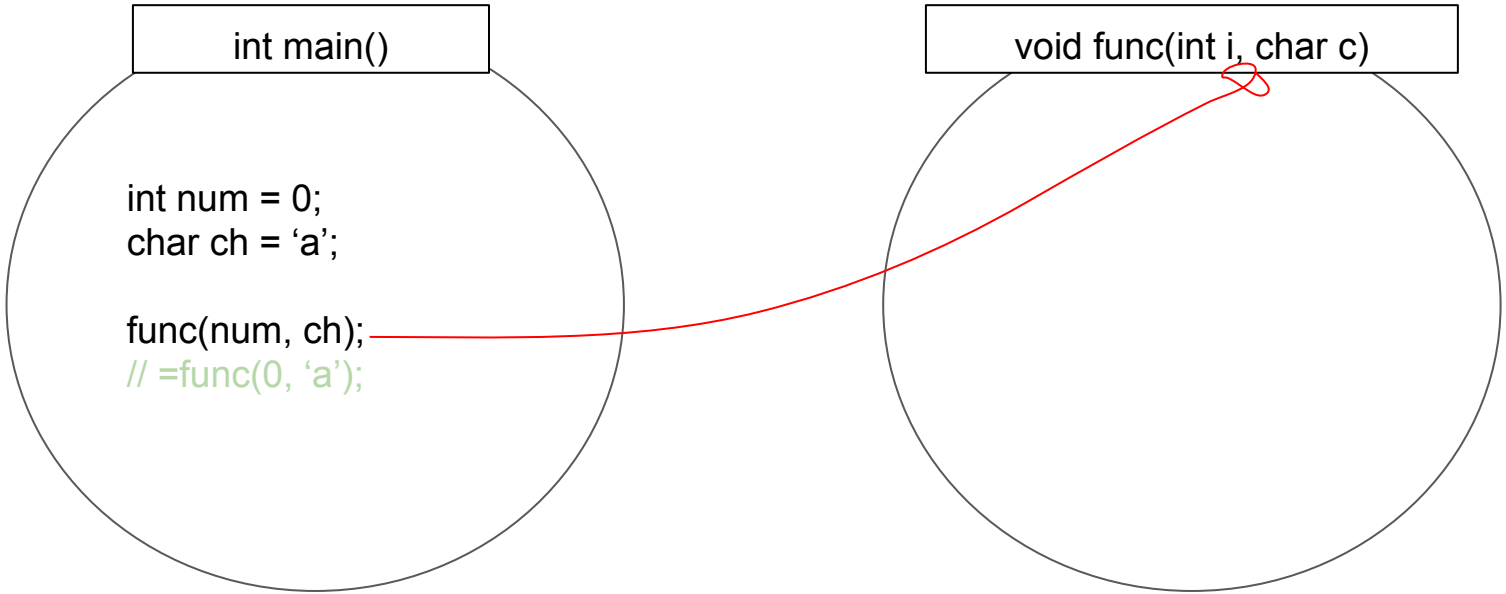
< 함수의 영역 >



[main]과 [func]는 다른 영역이다.
[main]에서 num변수와 ch변수는 서로 교류가 가능하지만
다른 영역인 [func]의 num변수와 ch와는 교류가 불가능하다

* 위 예제는 서로 다른 동네에서 동명이인이 있다고 생각하자 *

< 함수의 영역간 교류 >



다른 영역간의 교류는 함수 호출시 전달되는 인자를 통해 가능하다

전달된 인자는 호출된 함수의 매개변수가 받는다

전달되는것은 원본변수가 아닌 값만 전달된다

< 포인터 변수 >

변수의 주솟값을 담는 변수 (시작 주솟값)

```
int num = 0;        // int형 변수 선언
int* ptr;           // int형 포인터 변수 선언

ptr = &num          // ptr 에 num 변수의 주소를 대입
                    // 변수 좌측에 &(앰퍼샌드) 기호를 붙이면, 해당 변수의 주솟값을 얻는다

*ptr = 5;           // num = 5 와 같은 의미, 포인터 변수앞에 *(별) 기호를 붙이면
                    // 해당 포인터 변수가 갖고있는 주소를 참조한다
```

< 포인터 변수의 존재 의의 >

1. 다른 함수에 주솟값을 보내서, 원본을 직접 참조할 수 있게한다

2. 어떤 변수를 참조하려할때 복사하기에는 부담스러운 크기일때(구조체),
   주솟값만 알고있으면 굳이 복사할 필요는 없다

< 구조체 >

사용자 정의 자료형

여러 변수를 한번에 정의하고 싶을때 사용한다

```
struct Student {                    // 구조체 정의
        char class;
        int age;
        int weight;
};


struct Student gilDong;             // struct People 이라는 자료형을 갖는 변수 선언
gilDong.class = 'C';                // '.' 을 통해 구조체 내부 변수 참조
gilDong.age = 24;
gilDong.weight = 60;
```