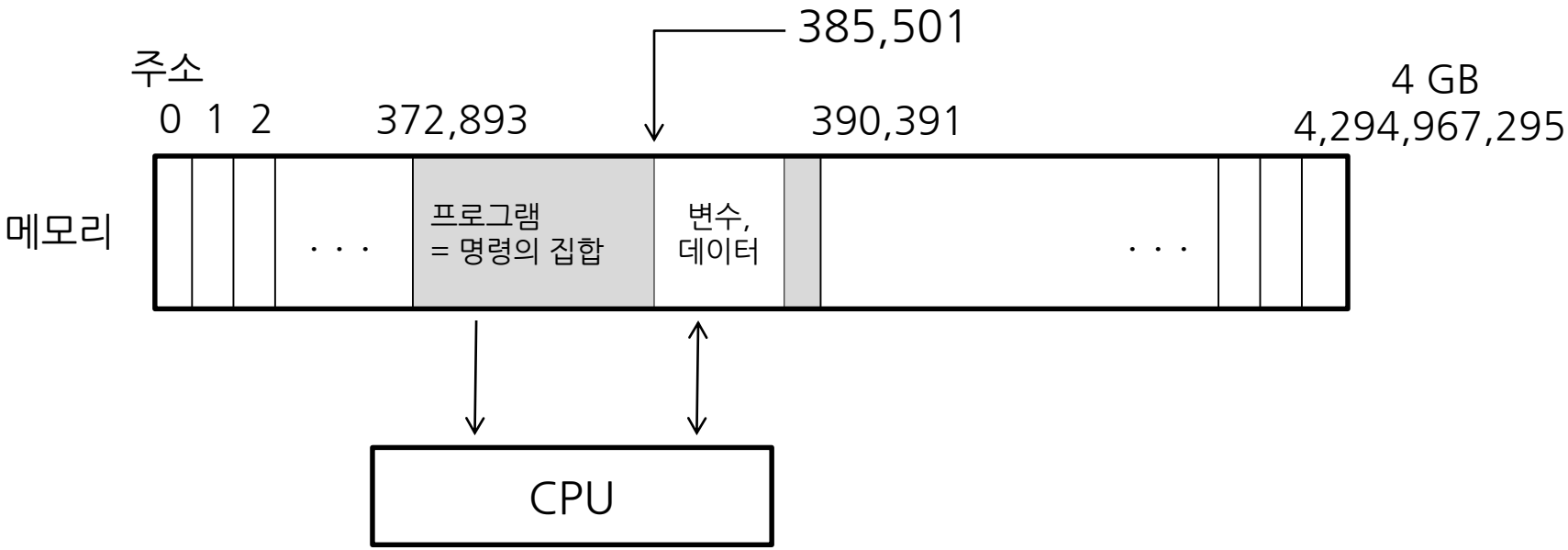
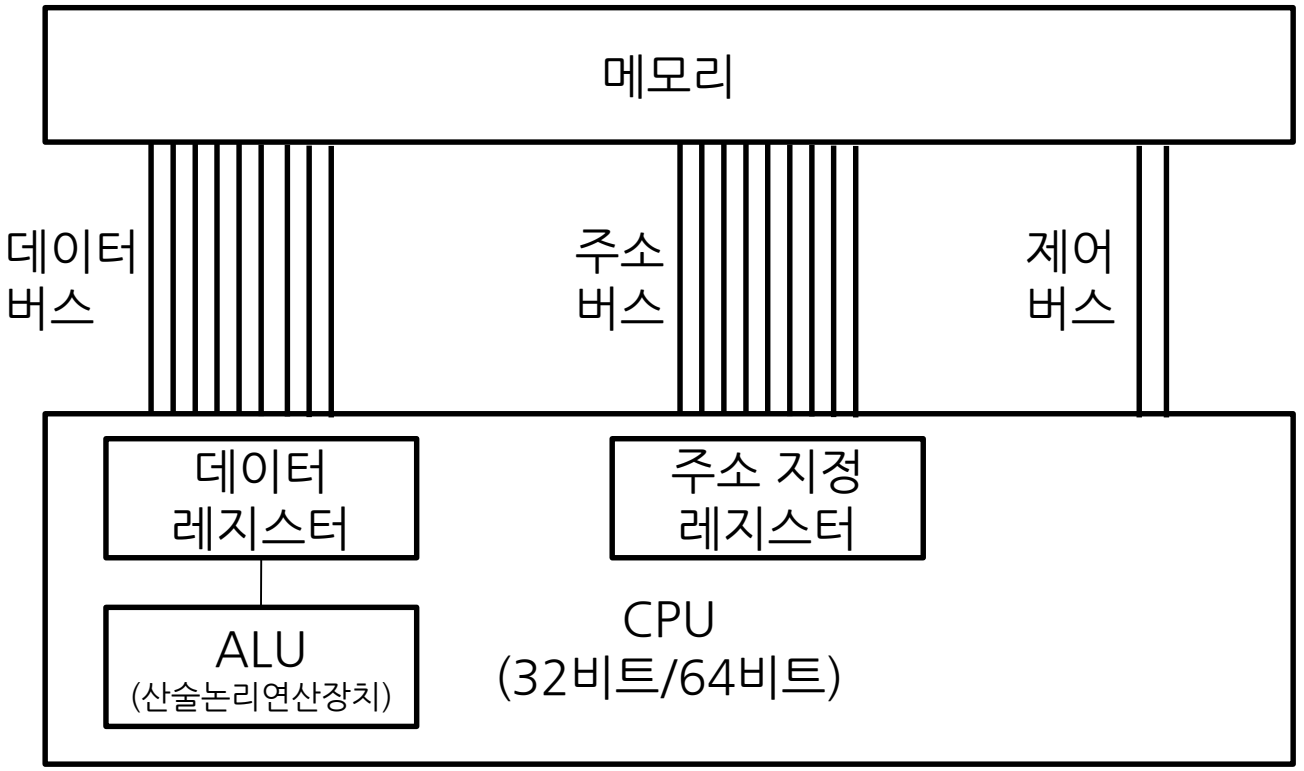
1차시 - 포인터(1)

폰노이만 구조



- 프로그램 실행에서 메모리는 중요한 역할을 한다.
- CPU에게 변수는 단지 메모리 공간일 뿐이다.
- 메모리는 주소가 정해져 있다. (용어 : addressing)

컴퓨터 시스템의 이해



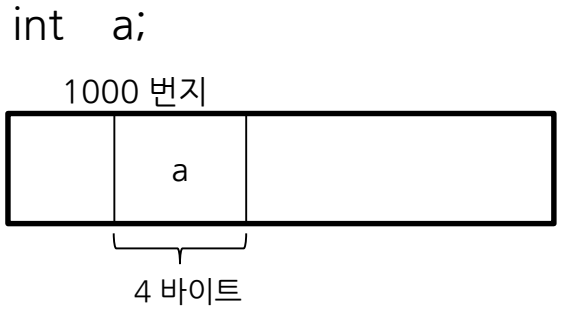
◆ a = b + c 를 실행하는 순서는 ? (CPU의 관점)

포인터란

- 포인터란 메모리의 주소이다.

포인터 변수의 선언

- **변수**
  - 자료를 저장하기 위해 메모리의 공간을 할당



- **포인터 변수**
  - 포인터(메모리의 주소)를 저장하는 변수 (“1000번지”를 저장하는 변수)
- **포인터 변수 ptr 의 선언**

```
int   i;      //  int형 데이터를 저장하는 변수 i의 선언
int  *ptr ;   //  포인터를 저장하는 변수 ptr의 선언
```

변수 이름이 ptr 이다 (*ptr이 아니다. *은 포인터 변수임을 알려줄 뿐이다)

- **포인터 변수 선언의 예**

```
int  *p, *q, *r; // 포인터 변수 p, q, r 의 선언
int*  p, q, r;   // 포인터 변수 p와 int형 변수 q, r의 선언
```

포인터 변수의 선언

변수 선언과 함께 초기화

```
int i = 1;
int *p = &i;     // i 변수의 주소로 p 변수를 채운다.
int *p = 1000;   // 가능하지만, 의미는 없다. (이유는 9페이지)
```

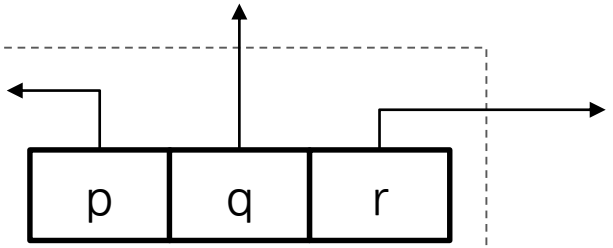
앞에 붙은 int의 의미

- 컴퓨터에서 중요한 것은 **데이터**이다.
- 포인터(주소)란 데이터가 있는 곳의 **위치**이다.
- int는 **그 위치에 있는 데이터의 자료형**을 의미한다.

포인터 변수의 선언

```
int   *p;
float *q;
char  *r;
```

세 개의 변수는 모두 같다.
(주소를 저장하는 변수)



- **세 개의 변수는 몇 바이트를 차지할까?**
  - 어떤 자료형이든 포인터 변수는 주소를 저장한다.
  - **몇 비트 컴퓨터, 몇 비트용 SW**인지기 중요하다.
  - 지금은 32비트이다.
  - unsigned int와 같다. (0 ~ 4GB)
  - 64비트 컴퓨터에서 64비트용 프로그램은 포인터도 64비트이다.

포인터 변수의 확인

```
#include <stdio.h>
 int main( )
{
        printf("size of pointer is %d₩n", sizeof( int * ) );
}
```

- **출력되는 값은?**
  - 4 바이트 (32비트) -> 32비트 시스템
  - 8 바이트 (64비트) -> 64비트 시스템
- **64비트 시스템인데 4라고 표시되는 이유**
  - 운영체제가 32비트이거나
  - Visual Studio가 32비트 빌드 플랫폼이거나

포인터 연산자 &, *

- **& (참조, reference) - 단항 연산자**
  - 일반 문장에서만 사용 : 변수 앞에만 붙일 수 있다.
  - 해당 변수의 메모리 주소를 가져온다.
  - 자료형은?

```
int *p = &i;      // 변수 i의 주소를 가져와 p 에 넣는다.
```

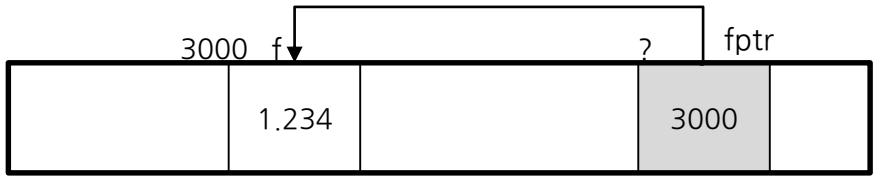
- *** (간접 참조, dereference) - 단항 연산자**
  - 일반 문장에서 사용 : 해당 주소에서 데이터를 가져온다.
  - 포인터 변수의 앞 또는 수식(주소) 앞에 붙인다.

```
int i = 10, *p = &i;
printf("%d\n", *p);
printf("%d\n", *(1000) );
```

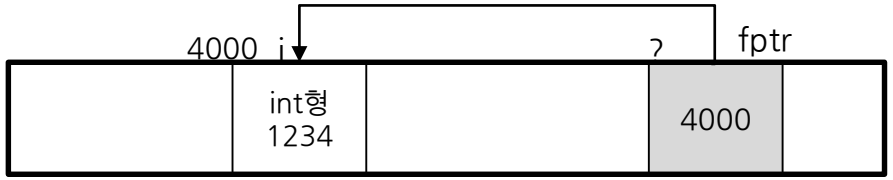
포인터 연산자 &, *

- * (간접 참조, dereference)

```
float   f = 1.234;
float   *fptr = &f;

printf("%f₩n",  fptr);
printf("%f₩n", *fptr);
```




```
int   i = 1234;
float *fptr = &i;  // 경고

printf("%f₩n", *fptr);
```



*fptr : float형으로 해석

- 포인터의 자료형이 다르면 무조건 경고가 나온다.
- int형과 float형은 저장 방식이 완전히 다르다.

포인터 연산자 &, *

- **& (참조, reference)**

```
int   i;
int   *p = &i;

printf("address of variable i is %p", p );    // %p는 주소 표시용 서식 문자(16진수)
```

  - 실행할 때마다 다른 값이 표시된다. 그 이유는?
  - 메모리는 누가 관리하는가?
  - 중요한 것은 i 의 주소가 아니라 p가 i 를 가리키고 있다는 사실

포인터 연산자 *, &

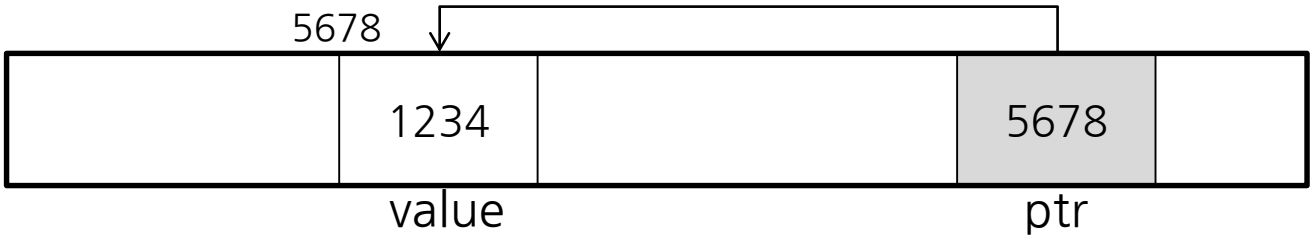
- ***와 &는 서로 상반된다.**
  - &는 변수의 주소를 가져오고, *는 주소의 데이터를 가져온다.
  - *(&i) == i ;
    - i 변수의 주소에 있는 데이터란 변수 i의 데이터이다.
  - p = &i 일 때 *p == i 일까?
  - (*p)++은 i++과 같을까?
  - &(*i) == i ; 일까? 그 이유는?

- **변수 앞에 붙인 & 연산자가 낯익은 이유는?**
  - scanf

포인터 연산자 *, &

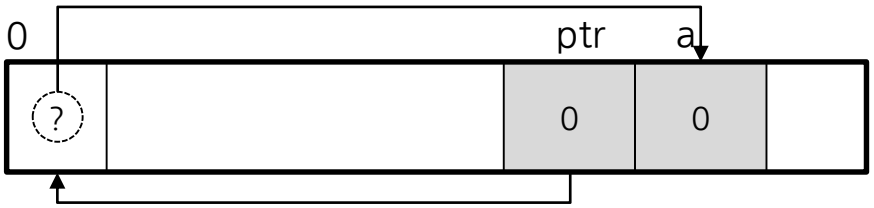
```
#include <stdio.h>
int main(void)
{
    int  value = 1234;
    int *ptr ;

    ptr  = &value;
    printf("%d\n", *ptr );
    value++;

    (*ptr)++;
    printf("%d\n", *ptr);
    return 0;
}
```

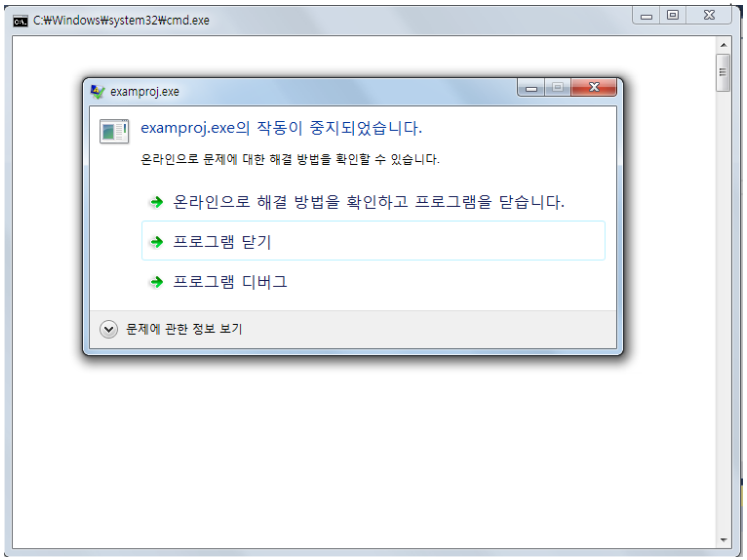
우선순위	연산자
1순위	( ) [ ] -> .
1.9순위	++, -- (**후치**)
2순위	! ~ ++ --(전치) + - * & (type) sizeof



포인터를 사용하는 프로그램

```
#include <stdio.h>
int  main( )
{
      int *ptr = 0 ;
      int   a = 0 ;

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

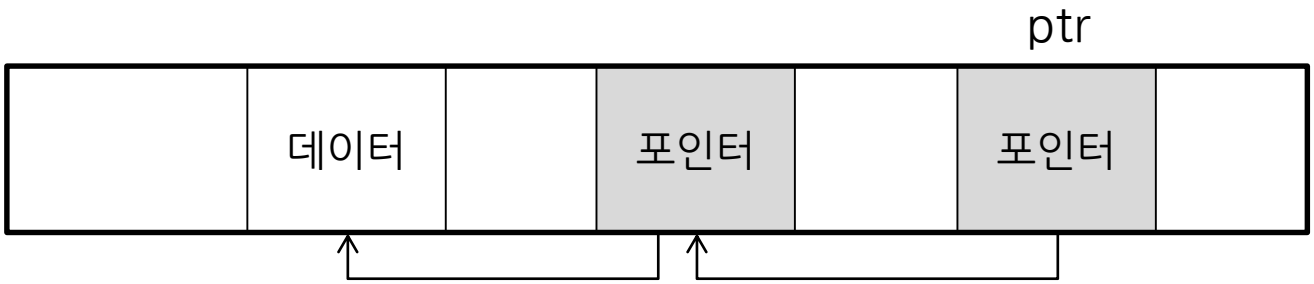




결과는? 이유는?

나에게 제공된 메모리만 써야 한다.

다중 포인터

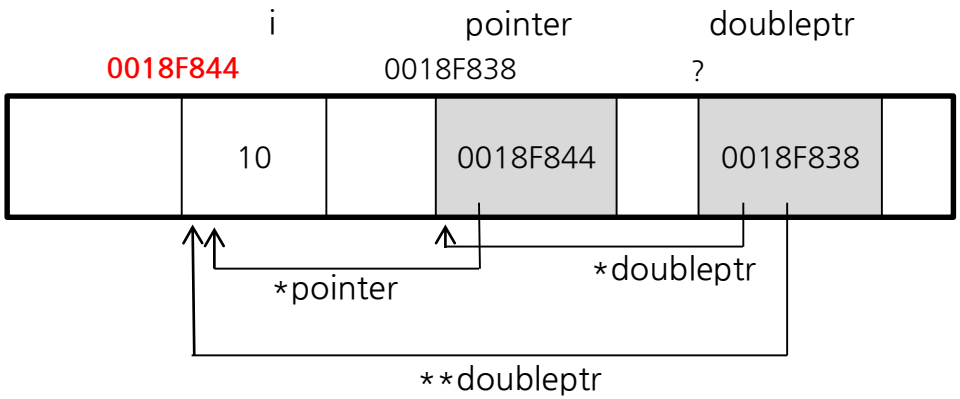


- **포인터 변수의 주소를 쫓아갔는데, 다시 포인터가 있을 때**
  - 한번 더 쫓아가야 데이터가 나타났다
- **선언 : int **ptr ; // 이것도 포인터 변수**
- **두 번 찾아가야 int형 데이터가 나타난다.**
  - **ptr = 123;

다 쓸 곳이 있다.

다중 포인터

```
int main( )
{
        int         i;
        int         *pointer;
        int         **doubleptr;

        i = 10;
        pointer = &i;
        doubleptr = &pointer;

        printf("i = %d₩n", i);
        printf("pointer = %p, *(pointer) = %d₩n", pointer, *pointer);
        printf("doubleptr = %p, *(doubleptr) = %p, **(doubleptr) = %d₩n",
                                        doubleptr, *doubleptr, **doubleptr);
}
```



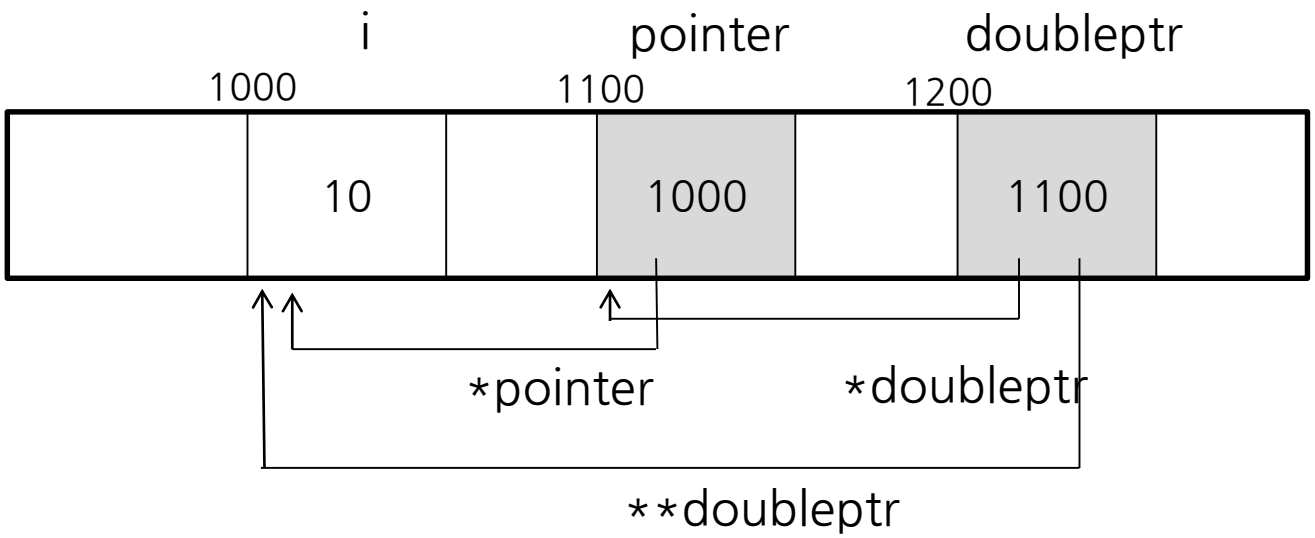

```
i = 10
pointer = 0018F844, *(pointer) = 10
doubleptr = 0018F838, *(doubleptr) = 0018F844, **(doubleptr) = 10
```

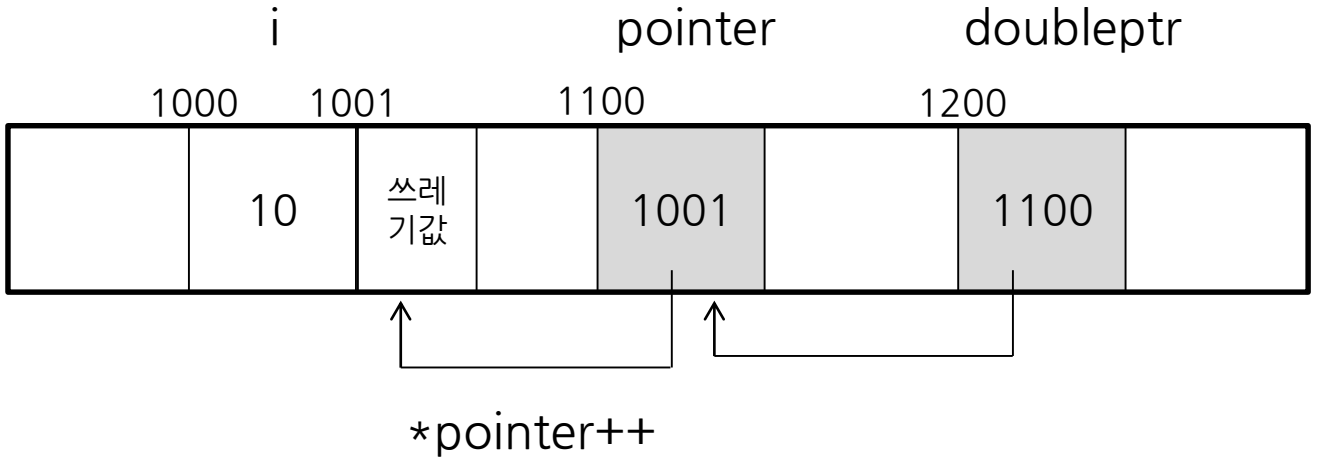
다중 포인터

```
int main(void) {
        int   i;
        int   *pointer = &i;
        int   **doubleptr = &pointer;

        i = 10;
        printf("%d %d %d\n", i, *pointer, **doubleptr);

        i++;
        (*pointer)++;
        (**doubleptr)++;

        printf("%d %d %d\n", i, *pointer, **doubleptr);

        *pointer ++;
        printf("%d %d %d\n", i, *pointer, **doubleptr);
}
```

괄호를 친 것은 이유가 있다.

다중 포인터


i
pointer
doubleptr
1000
1100
1200
10
1000
1100
*pointer
*doubleptr
**doubleptr


i
pointer
doubleptr
1000
1001
1100
1200
10
쓰레기값
1001
1100
*pointer++

Quiz)

- **int i; 에서 &i의 자료형은 무엇인가?**
- **int ***j; 에서 j의 자료형은 무엇인가? 또 *j의 자료형, **j 의 자료형, ***j 의 자료형은 무엇인가?**
  - j 값을 m 변수에 저장하려면 어떤 자료형으로 선언해야 하는가?
    - int ***m = j ;
    - int **n = *j;
  - **j, ***j의 값을 화면에 표시하려면 서식 문자는 각각 무엇을 써야 하는가?
- ***와 &중 무엇을 더 많이 사용하는가?**
  - 주소를 알려주는 방법은 & 말고도 여러 가지가 있다.
  - 데이터에 접근하려면 *을 써야 한다. 데이터에 접근이 많다. 다중 포인터도 있다.