프로그래밍 1
Lecture Note #12

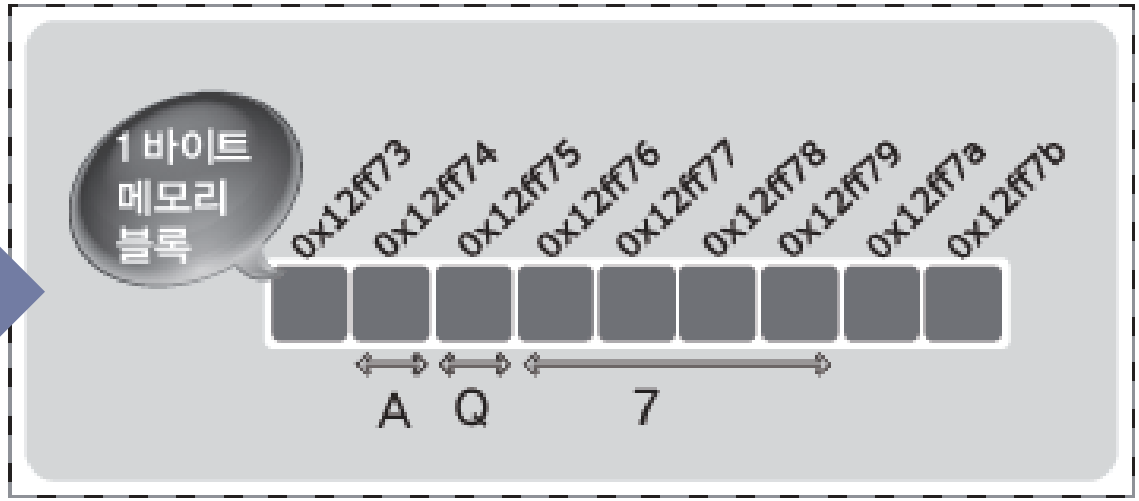
백윤철
ybaek@smu.ac.kr

내용

- 포인터 이해
- &연산자, *연산자
- Null 포인터

포인터 변수와 & 연산자 맛보기

```
int main(void) {
   char ch1 = 'A'
   char ch2 = 'Q';
   int num = 7;
   …
}
```



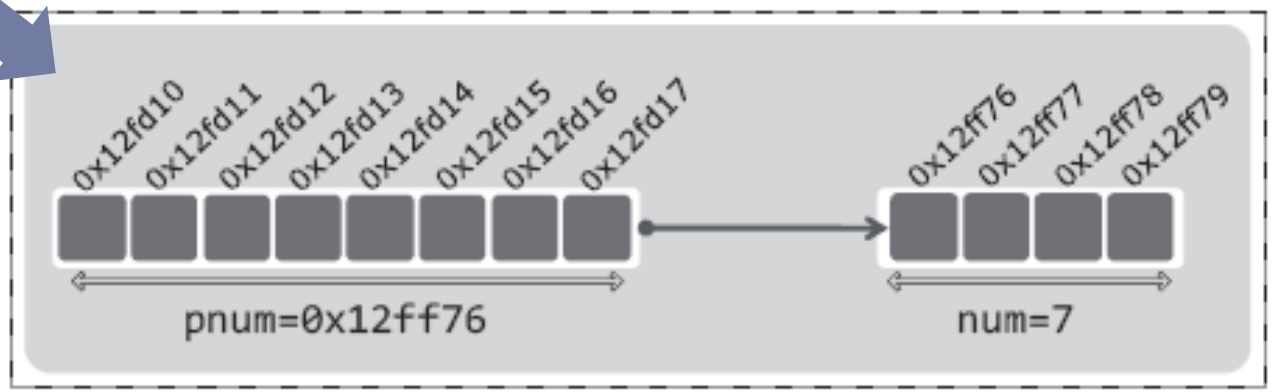
- 변수 num이 저장되기 시작한 주소 0x12ff76이 변수 num의 주소 값이 됨
- 이러한 정수 형태의 주소 값을 저장하는 목적으로 정의되는 것이 포인터 변수임

포인터 변수와 & 연산자 맛보기

"정수 7이 저장된 int형 변수 num을 정의하고 이 변수의 주소 값 저장을 위한 포인터 변수 pnum을 정의하자. 그러고 나서 pnum에 변수 num의 주소 값을 저장하자"

```
int main(void) {
   int num = 7;
   int* pnum;
   pnum = &num;
   …
}
```

포인터 변수 pnum의 선언
num의 주소 값을 pnum에 저장



포인터 변수 pnum이 변수 num을 가리킨다.

포인터 변수와 & 연산자 맛보기

- 포인터 변수의 크기는 시스템의 주소 값 크기에 따라서 다르다.
  - 16비트 시스템 → 주소 값 크기 16비트 → 포인터 변수의 크기 16비트!
  - 32비트 시스템 → 주소 값 크기 32비트 → 포인터 변수의 크기 32비트!

int * pnum 의 선언에서...

pnum	포인터 변수의 이름
int *	int형 변수의 주소 값을 저장하는 포인터 변수의 선언

포인터 변수 정의하기

- 가리키고자 하는 변수의 자료형에 따라서 포인터 변수의 정의방법에는 차이가 있음
- 포인터 변수에 저장되는 값은 모두 정수(주소값)으로 값의 형태는 모두 동일함
- 하지만 정의하는 방법에는 차이가 있음(차이가 있는 이유는 메모리 접근과 관련이 있다).

포인터 변수 정의하기

- `int *pnum1;`
  - int* 는 int형 변수를 가리키는 pnum1의 정의를 의미함
- `double *pnum2;`
  - double* 는 double형 변수를 가리키는 pnum2의 정의를 의미함
- `unsigned int *pnum3;`
  - unsigned int* 는 unsigned int형 변수를 가리키는 pnum3의 정의를 의미함

일반화

```
TYPE *ptr;
```

- TYPE형 변수의 주소 값을 저장하는 포인터 변수 ptr의 정의

포인터의 형(Type)

```
int*                int형 포인터
int *pnum1;         int형 포인터 변수 pnum1
double*             double형 포인터
double *pnum2;      double형 포인터 변수 pnum2
```

일반화

```
TYPE*               TYPE형 포인터
TYPE *ptr;          TYPE형 포인터 변수  ptr
```

- 포인터 변수의 정의에서 *의 위치에 따른 차이는 없음

```
int * ptr;
int* ptr;
int *ptr;
```

변수의 주소 값을 반환하는 & 연산자

- & 연산자는 변수의 주소 값을 반환하므로 상수가 아닌 변수가 피 연산자여야 함
- & 연산자의 반환 값은 포인터 변수에 저장함

```
int main(void) {
   int num = 7;
   int* pnum = &num;
…
}
```

변수의 주소 값을 반환하는 & 연산자

```
int main(void) {
   int num1 = 5;
   /*num1은 int형 변수이므로 pnum1은 int형 포인터 변수여야 함*/
   double *pnum1 = &num1; /* 불일치 */

   /*num2는 double형, pnum2는 double형 포인터 변수여야 함*/
   double num2 = 5;
   int *pnum2 = &num2; /* 불일치 */
   …
}
```

- **&연산의 반환 값은 같은 형의 포인터 변수에 저장해야 함**

포인터가 가리키는 메모리를 참조하는 *연산자

```
int main(void) {
   int num = 10;
   /* pnum이 num을 가리킴 */
   int *pnum = &num;
  /* pnum이 가리키는 공간(변수)에 20을 저장 */
   *pnum = 20;
  /* pnum이 가리키는 공간(변수)에 저장된 값을 출력 */
   printf("%d", *pnum);
   …
}
```

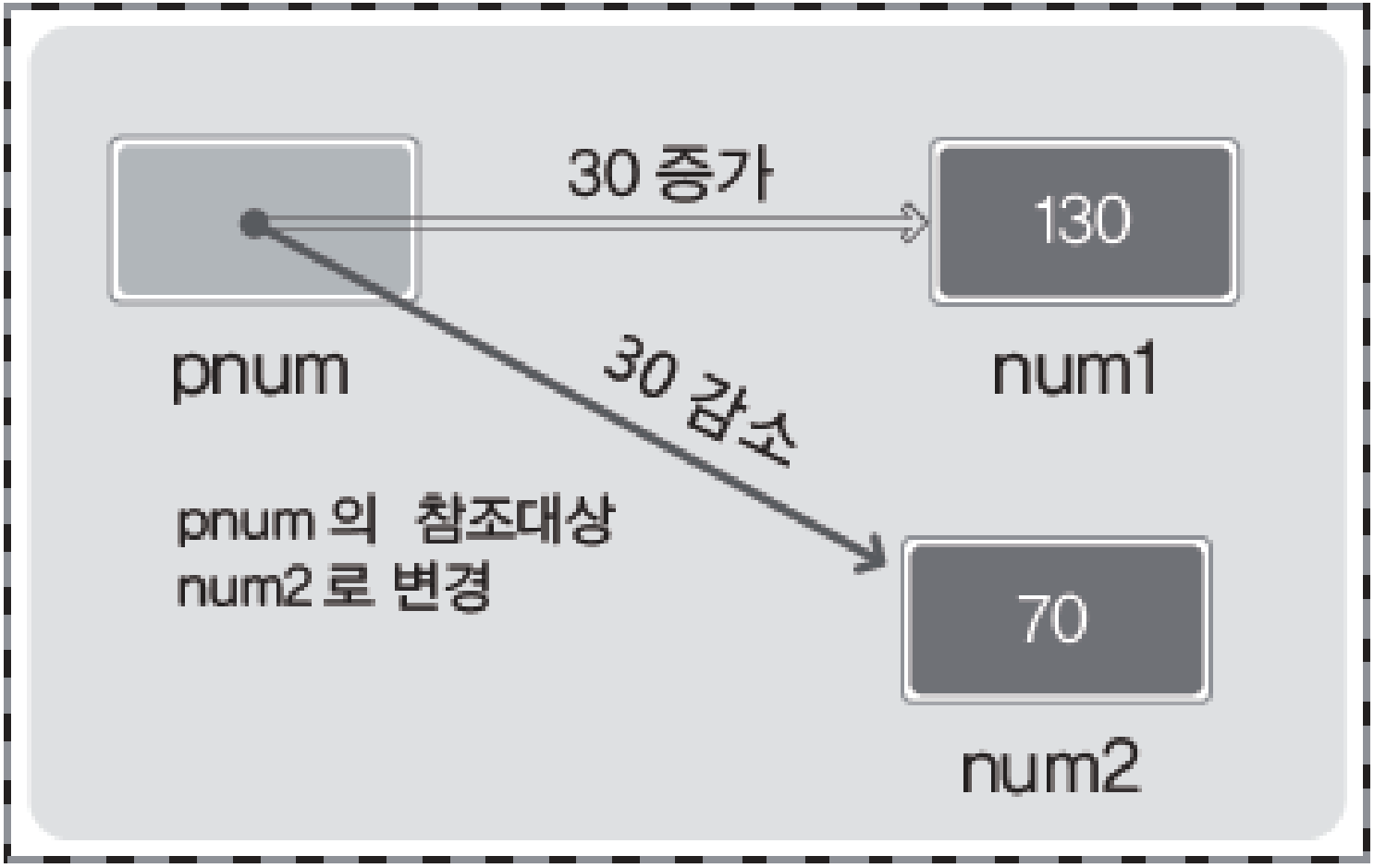
포인터가 가리키는 메모리를 참조하는 *연산자

```
int main(void) {
   int num1 = 100, num2 = 100;
   int *pnum;
   pnum = &num1;  /* pnum이 num1을 가리킴 */
   (*pnum) += 30; /* num1 += 30과 동일 */
   pnum = &num2;
   (*pnum) -= 30; /* num1 -= 30과 동일 */
   printf("num1: %d, num2: %d\n", num1, num2);
   return 0;
}
```

실행결과

```
num1:130, num2:70
```

포인터가 가리키는 메모리를 참조하는 *연산자



다양한 포인터 형이 존재하는 이유

- 포인터 형은 메모리 공간을 참조하는 방법의 힌트가 됨
- 다양한 포인터 형을 정의한 이유는 *연산을 통한 메모리의 접근 기준을 마련하기 위함
- 예:
  - int형 포인터 변수로 *연산을 통해 메모리(변수) 접근 시, 4바이트 메모리 공간에 부호 있는 정수의 형태로 데이터를 읽고 씀
  - double형 포인터 변수로 *연산을 통해 메모리(변수) 접근 시 8바이트 메모리 공간에 부호 있는 실수의 형태로 데이터를 읽고 씀

다양한 포인터 형이 존재하는 이유

```
int main(void) {
   double num = 3.14;
   int *pnum = &num; /* 형 불일치! 컴파일은 된다?*/

    /* pnum이 가리키는 것은 double형 변수인데, pnum이 int형 포인터 변수이므로 int형 데이터처럼 해석됨 */
   printf("%d", *pnum);
   return 0;
}
```

- 주소 값이 정수임에도 불구하고 int형 변수에 저장하지 않는 이유는 int형 변수에 저장하면 메모리 공간의 접근을 위한 *연산이 불가능함

잘못된 포인터의 사용과 NULL 포인터

```
int main(void)
{
   int *ptr;
   *ptr = 200;
   …
}
```

- ptr이 쓰레기 값으로 초기화 됨. 따라서 200이 저장되어야 하는 위치가 어디인지 알 수 없음! 매우 위험한 코드

```
int main(void) {
   int *ptr = 125;
   *ptr = 200;
   …
}
```

- 포인터 변수에 125를 저장했는데, 이 메모리 주소가 어디인지 모름! 역시 매우 위험한 코드

잘못된 포인터의 사용과 NULL 포인터

```
int main(void) {
   int *ptr1 = 0;
   int *ptr2 = NULL;
   …
}
```

- 잘못된 포인터 연산을 막기 위해 특정한 값으로 초기화하지 않는 포인터는 널 포인터로 초기화
- 널 포인터 NULL은 숫자 0을 의미함
- 0은 0번지를 뜻하는 것이 아니라, 아무것도 가리키지 않는다는 의미로 해석

정리

- 포인터 이해
- &연산자, *연산자
- Null 포인터