C 프로그래밍 프로젝트

Chap 12. 포인터의 이해

2013.09.25.

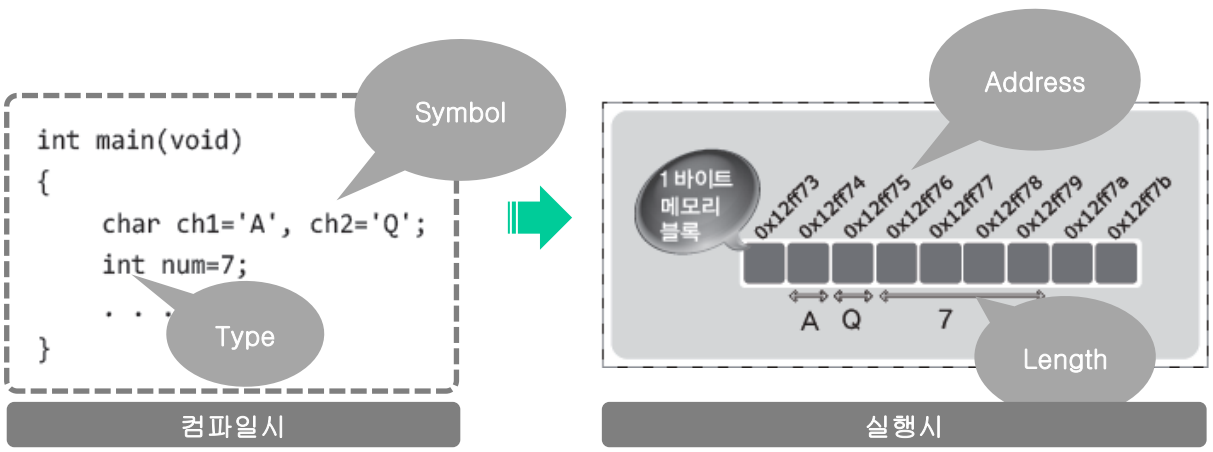
오 병 우

컴퓨터공학과

kit 국립금오공과대학교
Kumoh National Institute of Technology

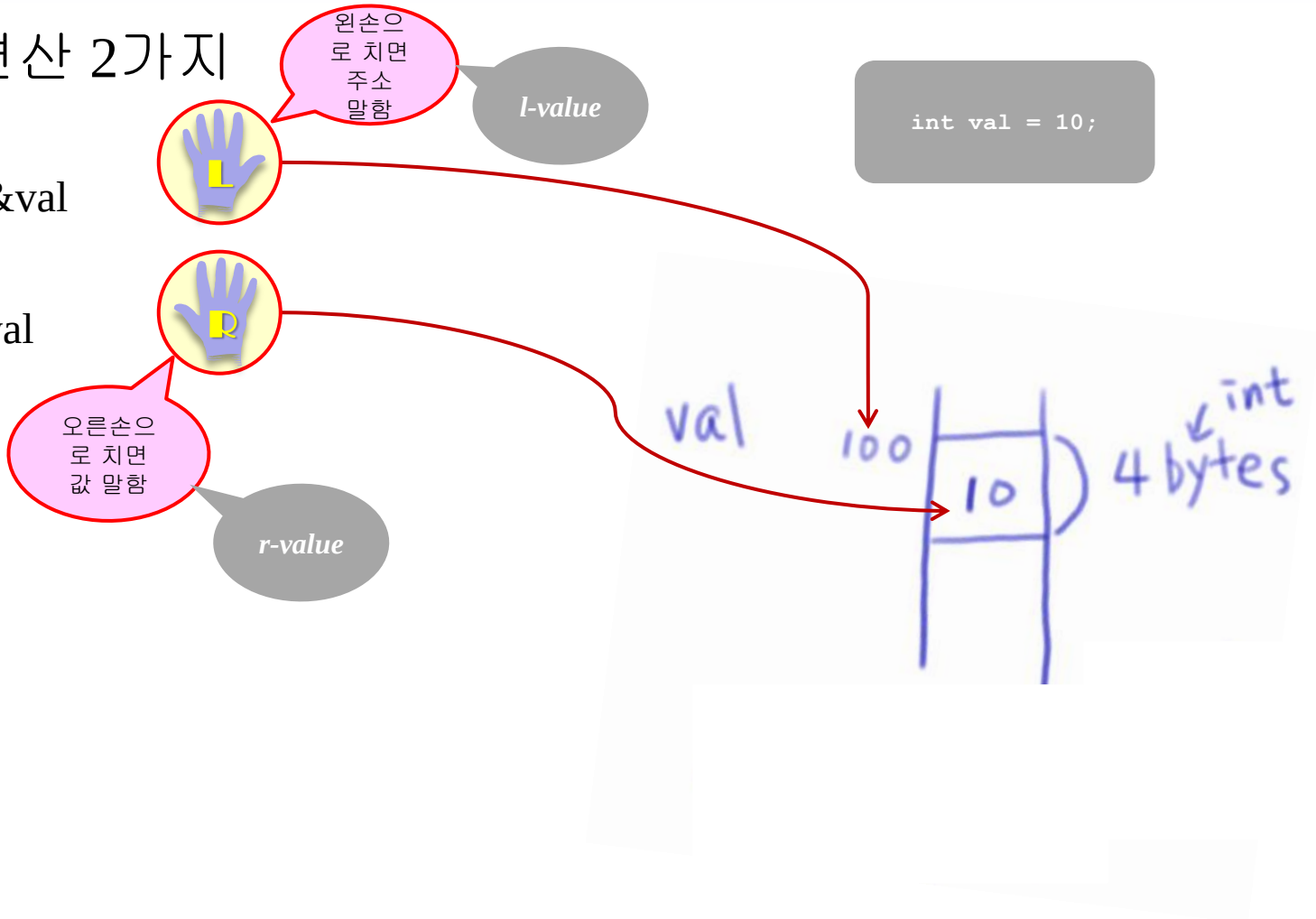
변수

- 값(Value)을 저장
- 컴파일시
  - 심볼 (Symbol)
  - 타입 (Type)
    - 자료형
- 실행시
  - 시작 주소 (Address or Offset)
    - 메모리에는 1 Byte 단위로 주소가 있음
  - 고정 길이 (Fixed Length)
    - Type에 의해 결정됨



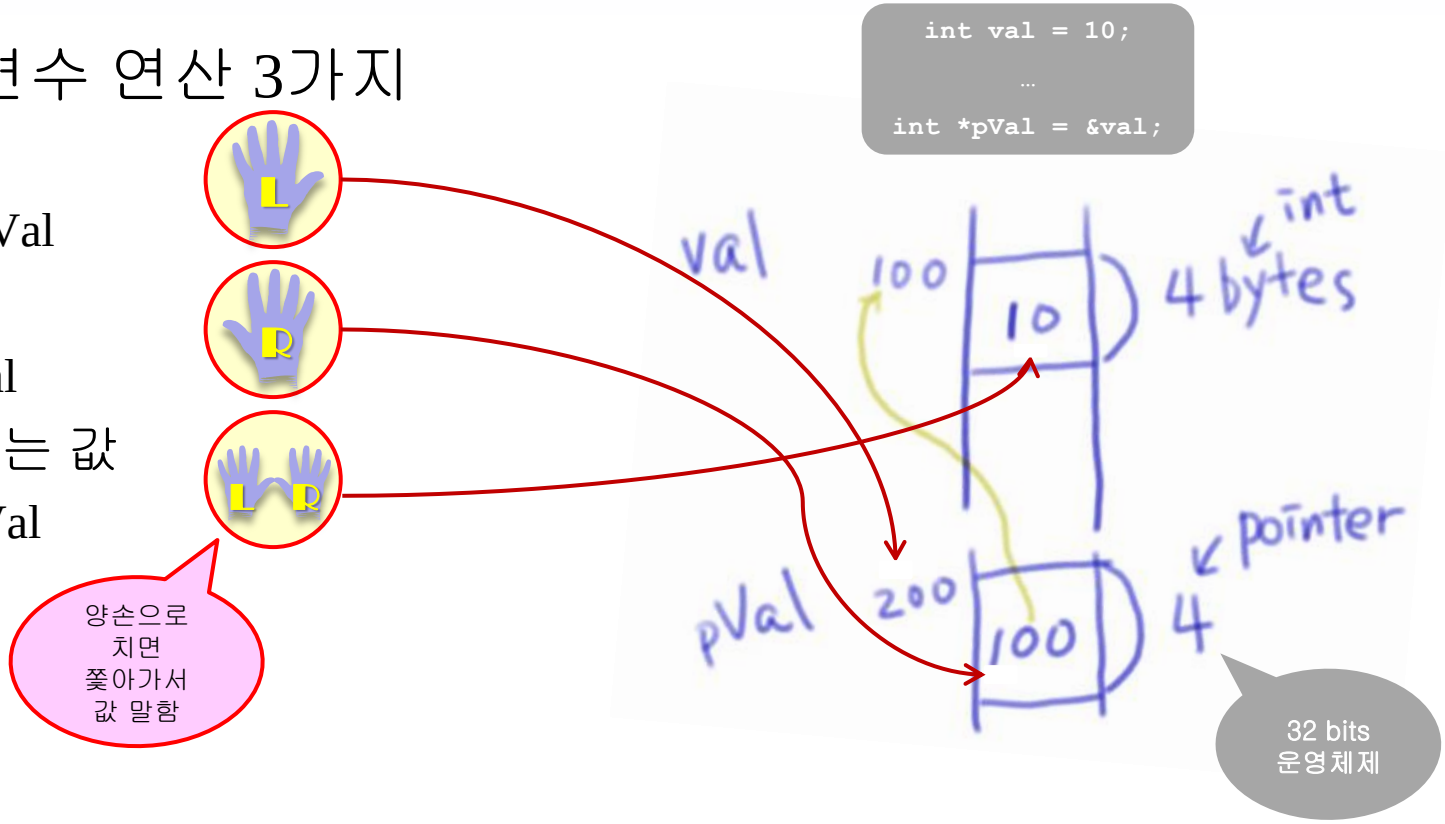
변수 접근

- 변수 연산 2가지
  - 주소
    - &val
  - 값
    - val



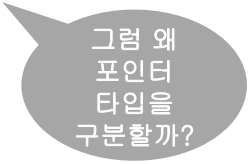
포인터 변수 접근

- 포인터 변수 연산 3가지
  - 주소
    - &pVal
  - 값
    - pVal
  - 가리키는 값
    - *pVal



포인터 변수

- 포인터 변수에 붙은 *의 의미
  - 가리키는 주소를 쫓아가서 그 대상을 의미함
  - 포인터 변수에 *가 붙는 순간 대상체의 개념이 들어감
    - 배열 이름에 *나 []가 붙는 경우도 대상체의 개념 사용
- 포인터 변수의 크기
  - 컴퓨터의 주소 체계에 따라 결정됨
    - char *, int *, double * 모두 일정함
  - 32-bit system – 4 bytes
  - 64-bit system – 8 bytes



포인터 변수와 & 연산자 맛보기

"정수 7이 저장된 int형 변수 num을 선언하고 이 변수의 주소 값 저장을 위한 포인터 변수 pnum을 선언하자. 그리고 나서 pnum에 변수 num의 주소 값을 저장하자."

코드로 옮긴 결과

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

포인터 변수 pnum의 선언
num의 주소 값을 pnum에 저장

포인터 변수의 크기는 시스템의 주소 값 크기에 따라서 다르다.

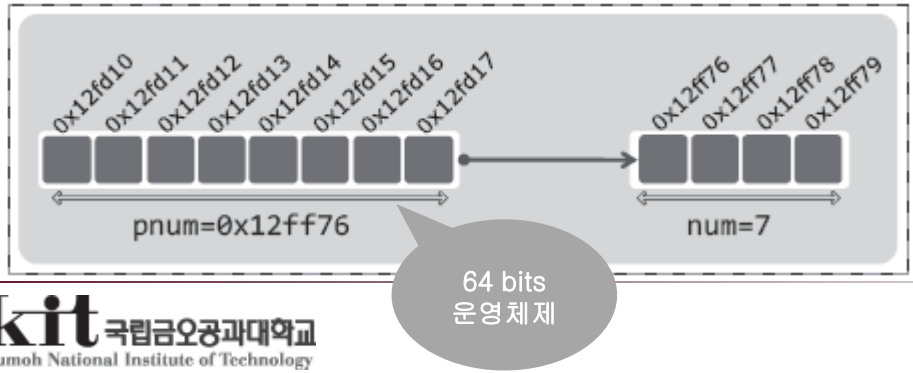
16비트 시스템 → 주소 값 크기 16비트 → 포인터 변수의 크기 16비트!

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

메모리의 저장상태



이 상태를 다음과 같이 표현한다.

포인터 변수 pnum이 변수 num을 가리킨다.

12-1 포인터란?

- 포인터의 타입과 선언
  - * 연산자 사용
  - A형 포인터 (A*): A형 변수의 주소값을 저장

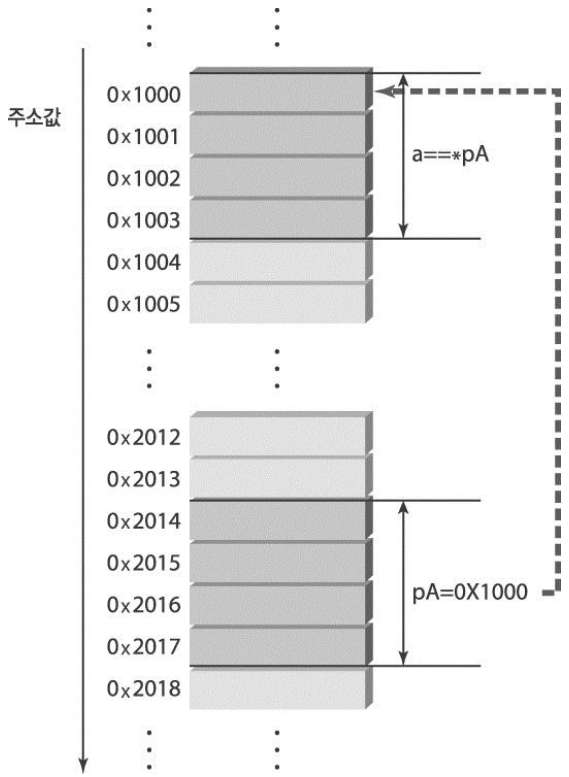
```
int main(void)
{
   int *a;          // a라는 이름의 int형 포인터
   char* b;         // b라는 이름의 char형 포인터
   double * c;      // c라는 이름의 double형 포인터
   .....
```

- 포인터의 타입을 왜 구분할까?
  - Hint: 포인터는 시작 주소만 알려 줌

12-1 포인터란?

- 주소 관련 연산자
  - 포인터 변수를 사용하기 위한 특별한 연산자
    - & 연산자: 변수의 주소값
    - * 연산자: 포인터가 가리키는 메모리 내용

```
int main(void)
{
   int a=2005;
   int *pA;
   pA = &a;
   printf("%d", a);       //직접 접근
   printf("%d", *pA);  // 간접 접근
   printf("%d", pA);    // 주소값
   .....
```



12-1 포인터란?

```
/* pointer1.c */
#include <stdio.h>

int main(void)
{
    int a=2005;
    int* pA=&a;

    printf("pA : %d \n", pA);
    printf("&a : %d \n", &a);

    (*pA)++;

    printf("a   : %d \n", a);
    printf("*pA : %d \n", *pA);

    return 0;
}
```

12-1 포인터란?

- 포인터의 타입을 왜 구분할까?
  - 포인터 자체는 늘 같은 크기
  - 간접 참조 시 얼마만큼 큰 데이터를 읽어야 하지?

```
#include <stdio.h>
 int main(void)
{
    int a=10;
    int *pA = &a;
    double e=3.14;
    double *pE=&e;

    printf("%d %f", *pA, *pE);
    return 0;
}
```

포인터: 시작 주소

타입: 크기

12-2 조심하자!

- 사례 1 위험한 코드

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

ptr이 쓰레기 값으로 초기화 된다. 따라서 200이 저장되는 위치는 어디인지 알 수 없다! 매우 위험한 행동!

- 사례 2 위험한 코드

```
int main(void)
{
     int * ptr=125;
     *ptr=10;
     . . . .
}
```

포인터 변수에 **125**를 저장했는데 이곳이 어디인가**?** 역시 매우 위험한 행동**!**

Null Pointer

- 초기화

안전한 코드

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

안전한 코드

잘못된 포인터 연산을 막기 위해서 특정한 값으로 초기화하지 않는 경우에는 널 포인터로 초기화하는 것이 안전하다.

널 포인터 NULL은 숫자 0을 의미한다. 그리고 0은 0번지를 뜻하는 것이 아니라, 아무것도 가리키지 않는다는 의미로 해석이 된다.

연습 문제

```
int main(void)
{
     int num=10;
     int * pnum=&num;
     *pnum=20;
     printf("%d", *pnum);
     . . . .
}
```

pnum이 num을 가리킨다.

pnum이 가리키는 공간(변수)에 20을 저장

pnum이 가리키는 공간(변수)에 저장된 값 출력

*pnum은 num을 의미한다.

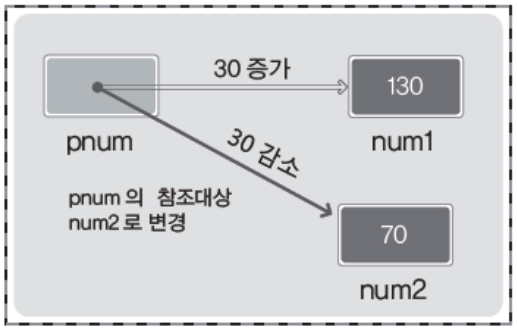
따라서 num을 놓을 자리에 *pnum을 놓을 수 있다.

```
int main(void)
{
    int num1=100, num2=100;
    int * pnum;

    pnum=&num1;     // 포인터 pnum이 num1을 가리킴
    (*pnum)+=30;    // num1+=30; 과 동일

    pnum=&num2;    // 포인터 pnum이 num2를 가리킴
    (*pnum)-=30;    // num2-=30; 과 동일

    printf("num1:%d, num2:%d \n", num1, num2);
    return 0;
}
```



실행결과

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

Summary

- Pointer
  - Pointer의 크기는 항상 일정함
    - 32 bits 운영체제에서는 4 Bytes
      - 예를 들어, char *pchar, int *pint, double *pdbl 세 개의 Pointer에 대해 sizeof(pchar), sizeof(pint), sizeof(pdbl) 모두 같음
        - » cf.) sizeof(*pchar), sizeof(*pint), sizeof(*pdbl)는 각각 다름 (1, 4, 8)
  - 저장하는 **값**이 메모리상의 주소를 가리킴
    - 시작 주소
  - Pointer의 **타입**이 대상체의 크기를 말해줌
    - 길이

```
int val = 10;
…
int* pVal = &val;
```

타입: 대상체의 크기

값: 시작 주소