프로그래밍 기초

11장. 포인터 기초 Part 1

학습목표

- **포인터 변수를 이해하고 설명할 수 있다.**
  - 메모리와 주소, 주소연산자 &
  - *를 사용한 포인터 변수 선언과 간접참조 방법
  - 포인터 변수의 연산과 형변환

- **다중 포인터와 배열 포인터를 이해하고 설명할 수 있다.**
  - 이중 포인터의 필요성과 선언 및 사용 방법
  - 증감연산자와 포인터와의 표현식
  - 포인터 상수

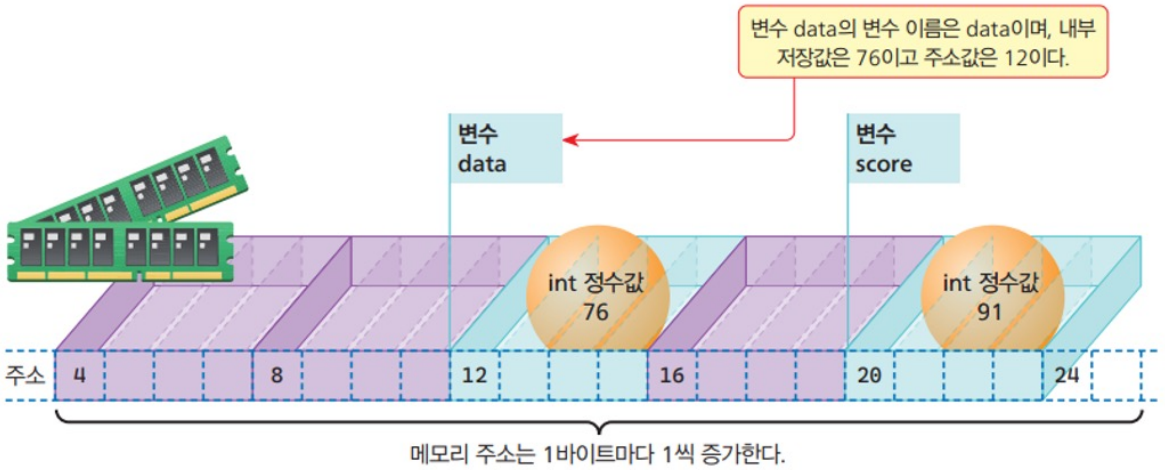
- **배열과 포인터 관계에 대하여 이해하고 설명할 수 있다.**
  - 배열이름은 포인터 상수이며 포인터 변수로도 참조
  - 1차원과 2차원 배열의 배열 포인터 활용
  - 포인터 배열

2

주소 개념

- **고유한 주소(address)**
  - 메모리 공간은 8비트인 1 바이트마다 고유한 숫자를 가짐
    - ➢ 0부터 바이트마다 1씩 증가
  - 메모리 주소는 변수이름과 함께 기억장소를 참조하는 또 다른 방법

- **메모리 주소가 필요한 이유**
  - 주소 정보를 이용하여 주소가 가리키는 변수의 값을 참조 가능
  - 주소 정보의 이전 또는 이후의 이웃한 저장 공간의 값도 쉽게 참조가 가능

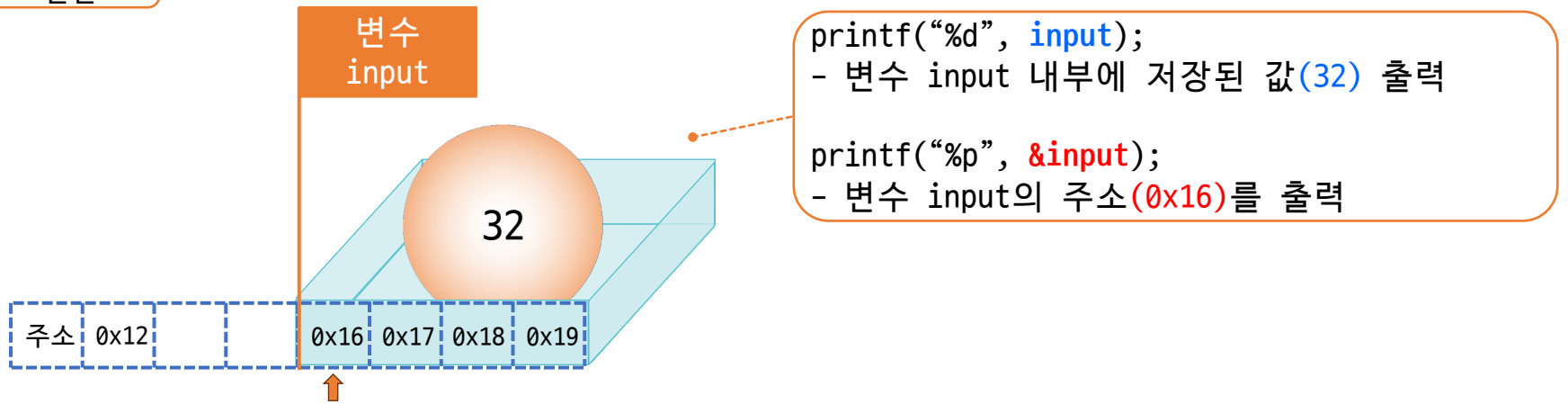


주소 연산자(&)

- **&(ampersand)가 변수의 메모리 주소를 반환**
  - 함수 scanf()에서 일반 변수 앞에는 주소연산자 &를 사용
    - ➢ 입력값을 저장하는 변수의 주소를 인자로 전달

```
int  input;
scanf("%d", &input);
```

input 변수의 주소 전달



printf("%d", input);
- 변수 input 내부에 저장된 값(32) 출력

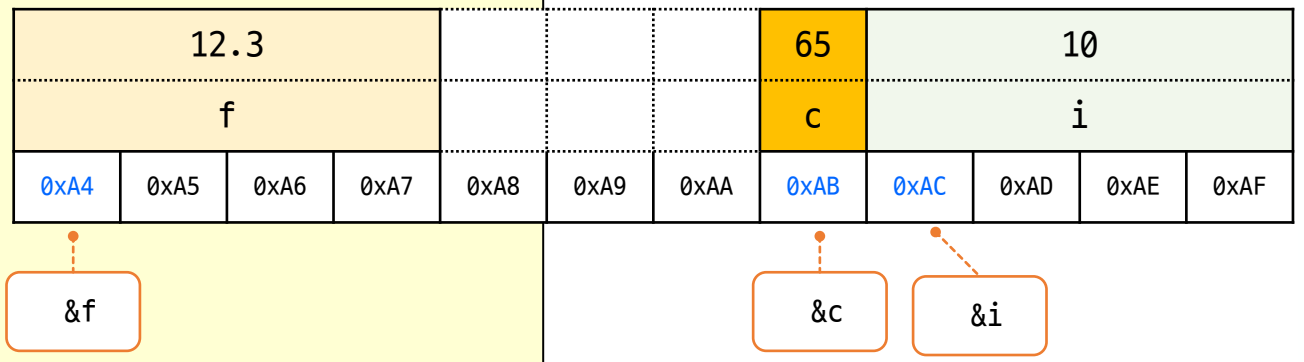
printf("%p", &input);
- 변수 input의 주소(0x16)를 출력

&input: 변수 input의 시작 주소를 반환

변수의 주소 (실습)

<var_addr.c>

```c
#include <stdio.h>

int main()
{
    int i = 10;
    char c = 65; // 'A': 65 (0x41)
    float f = 12.3;

    printf("i의 주소: %p\n", &i);
    printf("c의 주소: %p\n", &c);
    printf("f의 주소: %p\n", &f);

    return 0;
}
```




```
i의 주소: 0x16b13aaac
c의 주소: 0x16b13aaab
f의 주소: 0x16b13aaa4
```

주소 연산자(&)

- **변수의 주소 출력: printf("%p", …)**
  - 형식제어문자 printf("%p", &변수)로 직접 출력: 16진수로 출력
  - Windows의 64비트 시스템 주소 값: 8바이트(64비트)
  - 형식제어문자 %llu로 출력
    - %llu: long long unsigned를 의미 (64비트)
  - 자료형 uintptr_t
    - typedef unsigned __int64 uintptr_t;
    - __int64: long long int (64비트 정수 자료형)

<Windows 11>

```
#ifdef _WIN64
__MINGW_EXTENSION typedef unsigned __int64 uintptr_t;
#else
typedef unsigned int uintptr_t;
#endif /* _WIN64 */
```

<Mac>

```
typedef unsigned long uintptr_t;
```

- **연산자 sizeof(&input)의 반환 값**
  - 자료형 size_t 유형의 주소의 크기, 형식제어문자 %zu로 출력
    - z는 size를 u는 unsigned를 의미
  - 자료형 size_t: unsigned long long
  - 연산자 sizeof의 반환 값: %zd 로도 가능

메모리 주소연산자와 주소 출력

<01address.c>

```c
#include <stdio.h>

int main(void)
{
    int input;

    printf("정수 입력: ");
    scanf("%d", &input);
    printf("입력 값: %d\n", input);
    printf("주소 값: %p(16진수)\n", &input);
    printf("주소 값: %llu(10진수)\n", (uintptr_t)&input);

    printf("주소 값 크기: %zu\n", sizeof(&input));

    return 0;
}
```

%p: 16진수 출력
(빈 공간을 0으로 채워서 대문자로 출력)

```
정수 입력: 100
입력 값: 100
주소 값: 000000000061FE1C(16진수)
주소 값: 6422044(10진수)
주소 값 크기: 8
```

포인터 변수 개념과 선언

- **포인터 변수: 메모리의 주소를 저장하는 변수**
  - 변수의 주소는 반드시 포인터 변수에 저장
  - 특정 변수(메모리)의 주소를 가리킴(pointer)

```
자료형 *변수이름;
자료형* 변수이름;
```

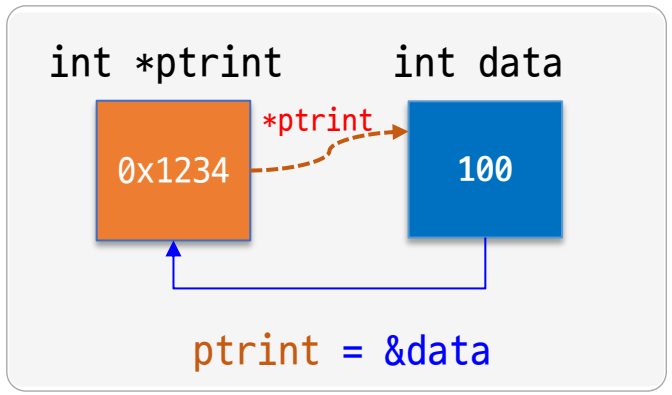
```
int *ptrint;
short *ptrshort;
char *ptrchar;
double *ptrdouble;
```

- **포인터 변수 선언**
  - 자료형과 변수 이름 사이에 연산자 *(asterisk)를 삽입
  - int *ptrint 또는 int* intptr
    - ➢ int 포인터 ptrint라고 읽음
  - 변수의 주소를 저장: 변수 자료형과 동일한 포인터 변수에 저장

```
int data = 100;
int* ptrint;

ptrint = &data;
printf("*ptrint: %16d\n", *ptrint);
```

포인터 변수 ptrint에 data의 주소를 저장

주소의 값



포인터 변수 개념과 선언 (실습)

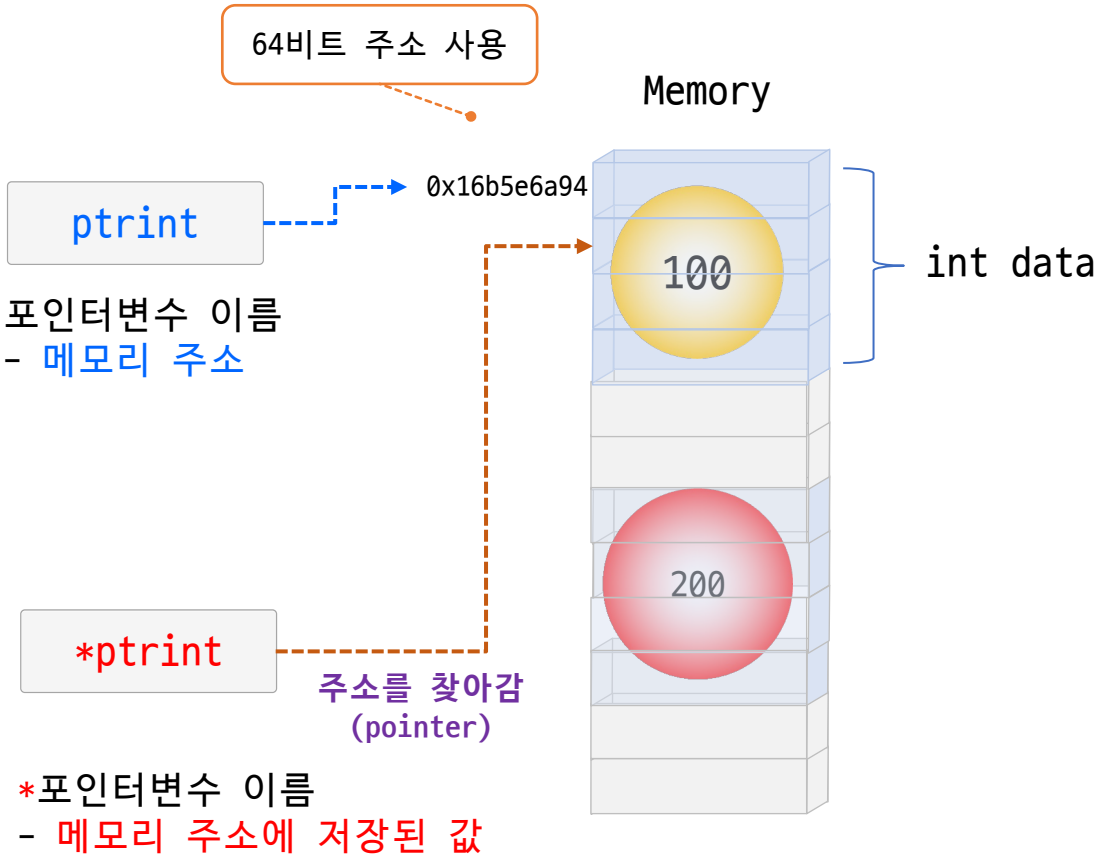
포인터 변수 개념과 사용

- ptrint와 *ptrint 차이

<pointer_ex01.c>

```c
#include <stdio.h>

int main()
{
    int data = 100;
    int* ptrint = &data; // data변수의 주소 전달

    // 주소 출력
    printf("ptrint: %p\n", ptrint);

    // ptrint 주소의 값 출력
    printf("*ptrint: %d\n", *ptrint);
    return 0;
}
```

```
ptrint: 0x16b5e6a94
*ptrint: 100
```



포인터를 사용해야 되는 상황: 함수 내부에서 변수의 값 변경 (실습)

- C언어에서 함수의 리턴값은 1개만 가능: 여러 변수의 값을 변경해서 전달 받을 경우

<pointer_ex02.c>

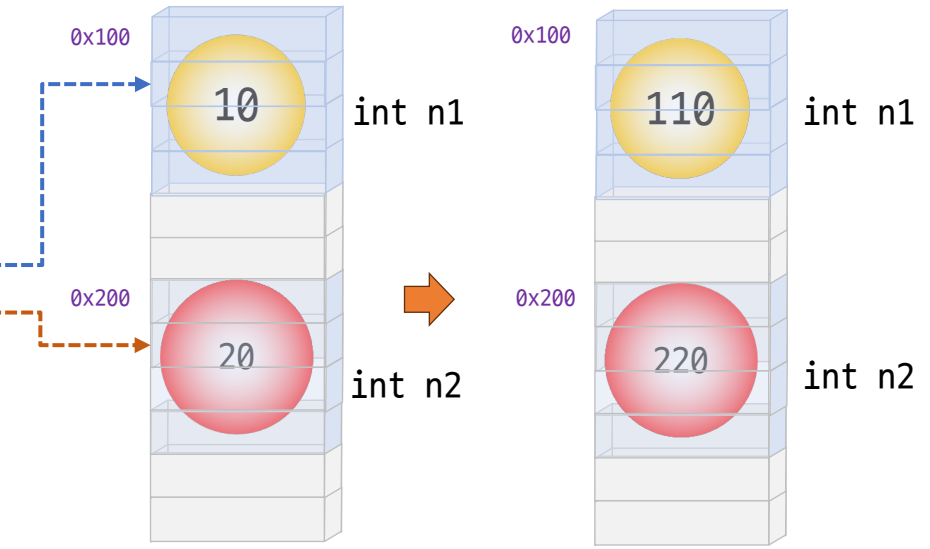
```c
#include <stdio.h>

void change_number(int *pnum1, int *pnum2)
{
    printf("\n");
    printf("1. *pnum1: %d, *pnum2: %d\n", *pnum1, *pnum2);
    // n1, n2의 주소를 찾아가서 값을 변경
    *pnum1 = *pnum1 + 100;
    *pnum2 = *pnum2 + 200;
    printf("2. *pnum1: %d, *pnum2: %d\n", *pnum1, *pnum2);
    printf("\n");
}

int main()
{
    int n1=10, n2=20;
    printf("Before change_number() n1=%d, n2=%d\n", n1, n2);
    change_number(&n1, &n2);
    printf("After change_number() n1=%d, n2=%d\n", n1, n2);
    return 0;
}
```

변수 n1과 n2의 주소 전달

*pnum1 + 100

*pnum2 + 200

함수 내부에서 바뀐 값이 저장되어 있음




```
Before change_number() n1=10, n2=20

1. *pnum1: 10, *pnum2: 20
2. *pnum1: 110, *pnum2: 220

After change_number() n1=110, n2=220
```

포인터 변수와 배열의 주소 및 값 접근 방법

- **배열 이름: 배열의 시작 주소를 의미**

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

printf("Address of array[]: %p\n", array);
```

```
Address of array[]: 0x16fa16e70
```

- 배열의 요소(값) 접근: 인덱스 사용

```
printf("array[0] : %d\n", array[0]);
```

```
array[0]   : 1
```

- **포인터 변수**
  - 포인터 변수 이름: (시작) 주소를 의미

```
int array[] = {1, 2, 3, 4, 5};
int *parray = array; //배열 array의 시작 주소 대입

printf("Address of parray: %p\n", parray);
```

```
Address of parray: 0x16fa16e70
```

- 값 접근: *포인터변수이름 또는 인덱스 사용

```
printf("parray[0] : %d\n", parray[0]);
printf("*parray : %d\n", *parray); //첫 요소의 값
printf("*(parray+1): %d\n", *(parray + 1));
```

```
parray[0]  : 1
*parray    : 1
*(parray+1): 2    // parray[1]과 동일
```

포인터 사용 예제 (실습)

<pointer_test1.c>

```c
#include <stdio.h>

int main()
{

    int array[] = {1, 2, 3, 4, 5};
    int *parray = array; // 배열 array의 주소를 대입

    printf("Address of array[]: %p\n", array);
    printf("Address of parray : %p\n", parray);

    printf("parray[0] : %d\n", parray[0]);
    printf("*parray   : %d\n", *parray);    // 첫 번째 요소[0]의 값

    printf("*(parray+1): %d\n", *(parray + 1)); // 인덱스 [1]의 값
    printf("parray[1]  : %d\n", parray[1]);
    printf("*(parray+2): %d\n", *(parray + 2));
    return 0;
}
```

parray: 포인터 변수 이름 자체
- 시작 주소를 의미

*parray: *포인터 변수 이름
- 해당 주소의 값을 의미

array[]와 parray는 동일한 주소

```
Address of array[]: 0x16b402e70
Address of parray : 0x16b402e70
parray[0]  : 1
*parray    : 1
parray[1]  : 2
*(parray+2): 3
```

포인터 변수 선언 (실습)

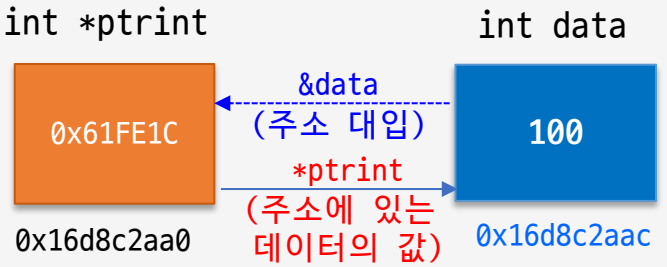
- **포인터 변수 선언과 주소값 대입**

<02pointer.c>

```c
#include <stdio.h>

int main(void)
{
    int data = 100;
    int *ptrint;
    ptrint = &data;

    printf("변수명 주소값 저장값\n");
    printf("-----------------------------------------\n");
    printf("   data %p %16d\n", &data, data);
    printf(" ptrint %p %p\n", &ptrint, ptrint);
    printf("*ptrint %p %16d\n", ptrint, *ptrint);

    printf("sizeof(ptrint): %zu\n", sizeof(ptrint));

    return 0;
}
```

&data: 변수 data가 저장된 메모리의 주소
ptrint: 변수 data의 메모리의 주소
*ptrint: 변수 data의 값
&ptrint: 포인터 변수 ptrint의 주소

포인터 변수 int *ptrint도 메모리 공간을 차지함(주소를 가짐)

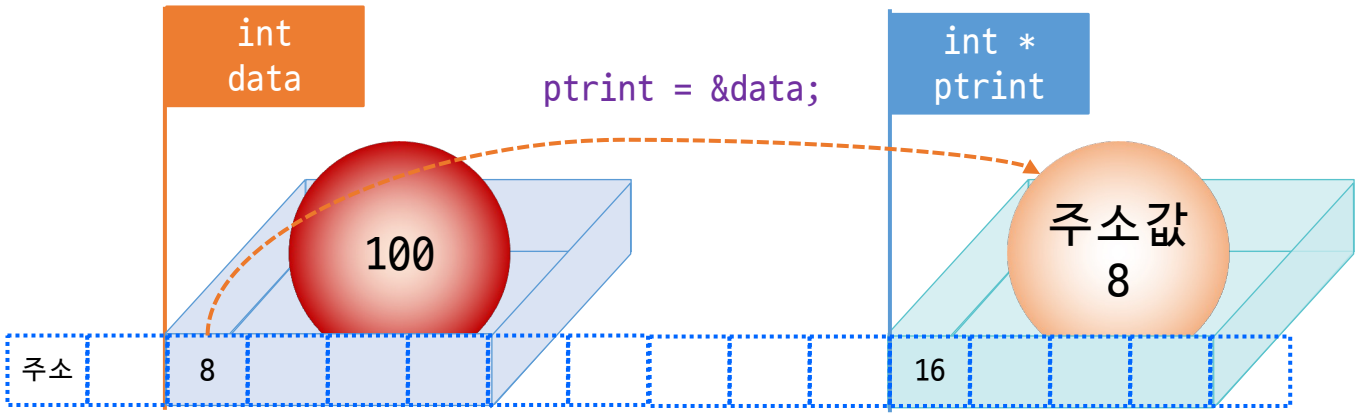


ptrint -> data의 주소(0x16d8c2aac)
*ptrint -> 0x16d8c2aac 주소의 값

```
변수명     주소값          저장값
-----------------------------------------
   data  0x16d8c2aac              100
 ptrint  0x16d8c2aa0  0x16d8c2aac
*ptrint  0x16d8c2aac              100
sizeof(ptrint): 8
```

포인터 변수 선언과 대입

- **포인터 변수**
  - 선언된 후 초기값이 없으면 쓰레기(garbage)값이 저장
  - int *ptrint = &data의 의미
    - ‘포인터 변수 ptrint는 변수 data를 가리킨다’ 또는 ‘참조(reference)한다’라고 표현
  - 포인터 변수의 크기
    - 64비트 시스템(Windows10)에서는 변수의 종류에 관계없이 모두 8바이트




```
int  data=100;
int  *ptrint;
ptrint = &data;
```

LAB: 다양한 자료형 포인터 변수 선언에 의한 주소값 출력

- char 포인터 변수 선언: char *pc
- int 포인터 변수 선언: int *pm
- double 포인터 변수 선언: double *px

lab1basicptr.c

```
01  #include <stdio.h>
02
03  int main(void)
04  {
05      char c = '@';
06      [                ]
07      int m = 100;
08      [                ]
09      double x = 5.83;
10      [                ]
11
12      printf("변수명   주소값              저장값\n");
13      printf("-----------------------------------\n");
14      [                                ]
15      [                                ]
16      [                                ]
17
18      return 0;
19  }
```

```
06      char *pc = &c;
08      int *pm = &m;
10      double *px = &x;
14      printf("%3s %12p %9c\n", "c", pc, c);
15      printf("%3s %12p %9d\n", "m", pm, m);
16      printf("%3s %12p %9f\n", "x", px, x);
```

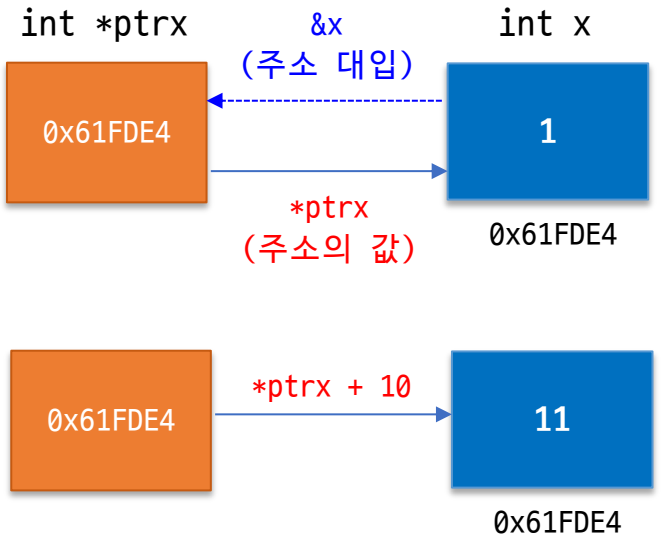
포인터 사용 예제 (실습)

<pointer_test2.c>

```c
#include <stdio.h>

int main()
{
    int x = 1;
    int *ptrx = &x; // ptrx는 변수 x를 가리킴

    printf("ptrx address: %p\n x address: %p\n", ptrx, &x);
    printf("1. *ptrx= %2d, x= %2d\n", *ptrx, x);

    *ptrx = *ptrx + 10;
    printf("2. *ptrx= %d, x= %d\n", *ptrx, x);

    x = x + 10;
    printf("3. *ptrx= %d, x= %d\n", *ptrx, x);

    return 0;
}
```

변수 x의 값을 직접 수정함

x의 값을 변경하면, ptrx도 변경됨



*ptrx = *ptrx + 10;
=> 주소를 방문하여 그 값을 변경

```
ptrx address: 000000000061FDE4
   x address: 000000000061FDE4
1. *ptrx=  1, x=  1
2. *ptrx= 11, x= 11
3. *ptrx= 21, x= 21
```

잘못된 포인터 사용

<wrong_pointer.c>

```c
#include <stdio.h>

int main()
{
    int a = 10;
    int *ptrA = 10;         // 값을 직접 대입: compile error

    //int *ptrA = &a;       // 정상 동작

    printf("ptrA addr: %p, value: %d\n", ptrA, *ptrA);
    /* 주소만 사용 */
    printf("ptrA addr: %p, prtA+1: %p\n", ptrA, ptrA + 1);

    return 0;
}
```

포인터 변수에 직접 값을 입력함
(Segmentation fault 에러 발생)

- Segmentation fault
  - 접근 권한이 없는 메모리 영역에 접근
  - 잘못된 방식으로 메모리에 접근할 때 발생

```
wrong_pointer.c:6:10: warning: incompatible integer to pointer conversion initializing 'int *' with an
expression of type 'int' [-Wint-conversion]
    int *ptrA = 10;
         ^      ~~
1 warning generated.
[1]    98240 segmentation fault
```

여러 포인터 변수 선언

- **여러 개의 포인터 변수를 한 번에 선언**
  - `int *ptr1, ptr2, ptr3; // ptr2와 ptr3는 단순 int형 변수`
  - `int *ptr1, *ptr2, *ptr3; // 모두 int형 포인터 변수`

- **포인터 변수 초기화**

```
int *ptr = NULL;
```

  - ➢ 0번 주소값인 NULL로 초기값 저장
  - ➢ NULL이 저장된 포인터 변수는 아무 변수도 가리키고 있지 않다는 의미

- **NULL**
  - stdio.h에 정의되어 있는 포인터 상수로서 0번지의 주소값을 의미

```
#define NULL ((void *)0)
```

  - ➢ (void *)는 아직 결정되지 않은 자료형의 주소를 의미
  - ➢ 모든 유형의 포인터 값을 저장할 수 있음

한 번에 여러 포인터 변수 선언과 NULL 주소값 대입

<03nullptr.c>

```c
#include <stdio.h>

int main(void)
{
    int data = 10;
    int *p1 = NULL, *p2 = &data;

    printf("%d\n", data);
    printf("%p %p\n", p1, p2);

    int *p3, data2;
    printf("p3: %p\n", p3); // int * 형의 쓰레기 값으로 오류
    printf("data2: %d\n", data2); // int 형의 쓰레기 값으로 오류

    return 0;
}
```

int *p3 = NULL, data2 = 0;

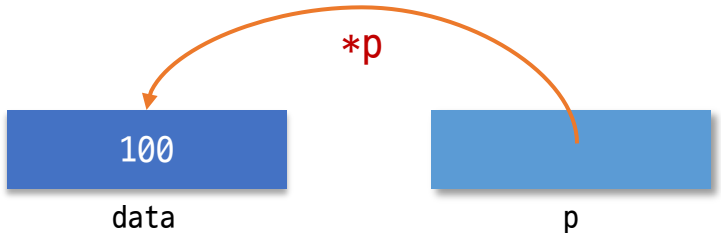
오류 발생은 없지만, 쓰레기 값 저장

```
10
0x0 0x16b10aa90
p3: 0xb68070
data2: 2
```

간접연산자 *와 포인터 연산

- **간접연산자(indirection operator) *를 사용한 역참조**
  - 포인터 변수가 가리키고 있는 변수를 참조
  - 단항 연산자인 간접연산자 *
    - *p는 피연산자인 p가 가리키는 변수 자체를 의미(가리키고 있는 변수의 값)
  - 포인터 p는 data의 주소 값을 가지므로 *p는 data와 같음

```
int data = 100;
int *p = &data;
printf("간접참조 출력: %d \n", *p);
```



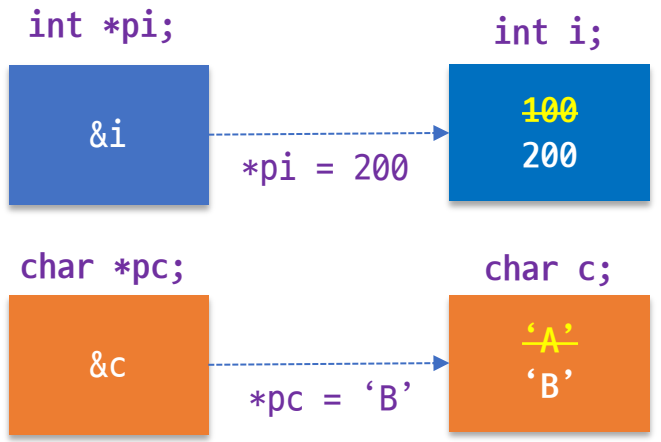
- **포인터 p가 가리키는 변수가 data이면 *p은 변수 data의 값을 의미**
  - 변수 data로 가능한 작업은 *p로도 가능
  - 문장 *p = 200;으로 변수 data 의 저장 값을 200으로 수정 가능

포인터 변수와 간접 연산자 *를 이용한 간접 참조

<04dereference.c>

```c
#include <stdio.h>

int main(void)
{
    int i = 100;
    char c = 'A' ;

    int *pi = &i;
    char *pc = &c;
    printf("간접참조 출력: %d %c\n", *pi, *pc);

    *pi = 200; //변수 i를 *pi로 간접참조하여 그 내용을 수정
    *pc = 'B'; //변수 c를 *pc로 간접참조하여 그 내용을 수정
    printf("직접참조 출력: %d %c\n", i, c);

    int data = 1000;
    pi = &data;
    printf("직접참조 출력: %d %d\n", data, *pi);
    //&data = 200; // 허용 안됨: 주소 연산자는 l-value로 사용할 수 없음

    return 0;
}
```

포인터 변수 pi에 변수 i의 주소 저장




```
간접참조 출력: 100 A
직접참조 출력: 200 B
직접참조 출력: 1000 1000
```

함수에 주소 전달 (포인터 및 주소 연산자 활용)

- **포인터 변수(함수 파라미터)에 변수의 주소 전달**
  - 일반 자료형: &변수 사용

```
void change_num(int *num) { }
```

```
int n = 10;
change_num(&n);
```

변수의 주소 전달

  - 배열: 배열의 이름 전달
    - 배열 이름: 배열의 시작 주소

```
void change_array(int *array, int size)
```

```
int data[5] = {0};
int size = sizeof(data) / sizeof(data[0]);

change_array(data, size);
```

배열의 시작 주소 전달

함수에 주소 전달 예제 (포인터 파라미터)

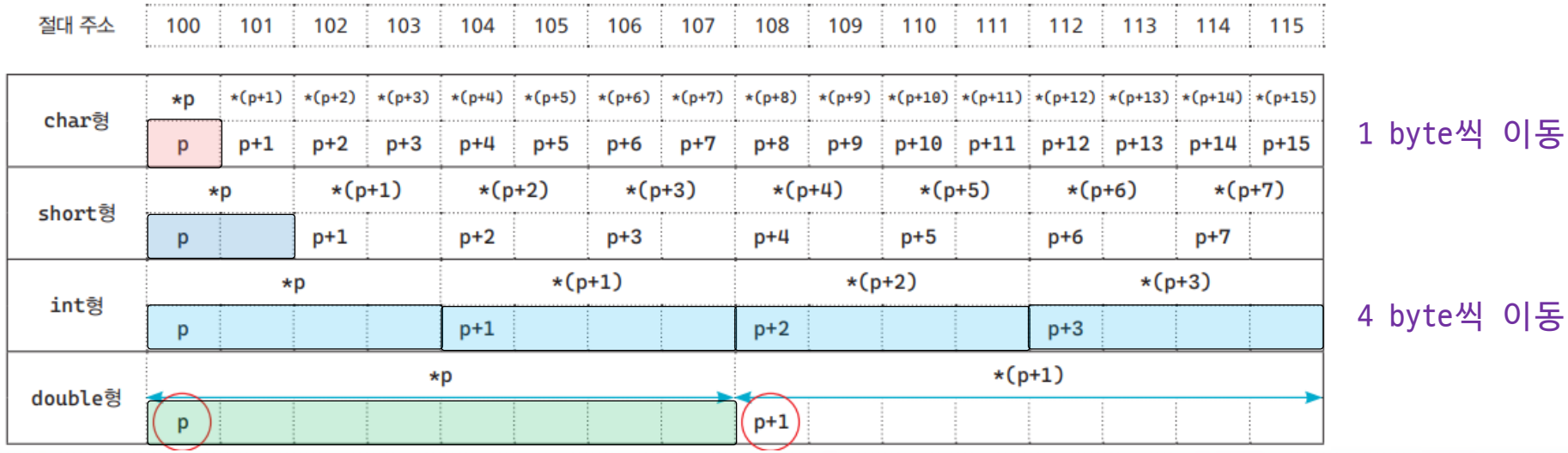
```c
#include <stdio.h>

void change_num(int *num)
{
    *num = *num + 1;
}

void change_array(int *array, int size)
{
    for (int i = 0; i < size; i++)
    {
        array[i] = array[i] + (i+1);
    }
}

void print_array(int *array, int size)
{
    for (int i = 0; i < size; i++)
        printf("%d ", array[i]);
    printf("\n");
}
```

```c
int main()
{
    int n = 10;
    change_num(&n);
    printf("n=%d\n", n);

    int data[5] = {0};
    int size = sizeof(data) / sizeof(data[0]);
    change_array(data, size);
    print_array(data, size);

    return 0;
}
```

```
n=11
1 2 3 4 5
```

포인터 변수의 연산

- **주소 연산**
  - 간단한 더하기와 뺄셈 연산으로 이웃한 변수의 주소 연산을 수행
    - 절대적인 주소의 계산이 아니며, 변수 자료형의 상대적인 위치에 대한 연산
  - p+1: p가 가리키는 자료형의 다음 주소로 실제 주소 값 (p[1])
    - p + [자료형 크기(바이트)]
  - p+2: p가 가리키는 자료형의 다음 다음 주소로 실제 주소 값 (p[2])
    - p + [자료형 크기(바이트)]*2
  - p+i: p가 가리키는 자료형의 다음 i번째 주소로 실제 주소 값 (p[i])
    - p + [자료형 크기(바이트)]*i



포인터 변수의 간단한 덧셈 뺄셈 연산

<05arithptr.c>

```c
#include <stdio.h>

int main(void)
{
    char *pc = (char *)100;   // int 값(4bytes)을 char *로 변환해서 주소 저장
    int *pi = (int *)100;     // 100을 int 주소로 변환해 저장
    double *pd = (double *)100; // 100을 double 주소로 변환해 저장
    // pd = 100; // double 포인터에 100으로 저장하면 경고 발생

    printf("char   %lld %lld %lld\n", (long long)(pc - 1),
                                (long long)pc, (long long)(pc + 1));

    printf("int    %lld %lld %lld\n", (long long)(pi - 1),
                                (long long)pi, (long long)(pi + 1));

    printf("double %lld %lld %lld\n", (long long)(pd - 1),
                                (long long)pd, (long long)(pd + 1));

    // printf("*pc=%c, *pi=%d\n", *pc, *pi); // segmentation fault 발생
    return 0;
}
```

주소의 증가 및 감소만 사용

데이터의 크기만큼 주소가 증가

```
char   99  100  101
int    96  100  104
double 92  100  108
```

LAB. 포인터를 이용하여 두 수의 값을 교환하는 프로그램

- **정수 int 자료형 두 변수 x, y에 저장된 두 값을 서로 교환하는 프로그램**
  - 임시변수인 dummy를 포함해 일반 변수는 사용하지 않고
  - 모두 포인터 변수인 px, py, pd 만을 사용

Lab 11-2 lab2swap.c 난이도: ★

```
01  #include <stdio.h>
02
03  int main(void)
04  {
05      int x = 500, y = 700, dummy;
06      printf("%d %d\n", x, y);
07
08      int *px = &x, *py = &y, *pd = &dummy;
09
10      // 변수 x과 y, dummy를 사용하지 않고 *px, *py, *pd를 사용해 두 변수를 서로 교환
11      [          ]  // 변수 dummy에 x를 저장
12      *px = *py;    // 변수 m에 n을 저장
13      [          ]  // 변수 n에 dummy를 저장
14
15      printf("%d %d\n", x, y);
16
17      return 0;
18  }
```

정답

```
11      *pd = *px;   // 변수 dummy에 x를 저장
13      *py = *pd;   // 변수 n에 dummy를 저장
```

Questions?