포인터

2주차

- 포인터변수의 자료유형 변환과 연산

이중포인터, 다중포인터
(포인터의 포인터)

```
// file: multipointer.c
#include <stdio.h>

int main(void) {
    int i = 20;
    int *pi = &i;           //포인터 선언
    int **dpi = &pi;        //이중 포인터 선언

    printf("%p %p %p₩n", &i, pi, *dpi);

    *pi = i + 2;          // i = i + 2;
    printf("%d %d %d₩n", i, *pi, **dpi);

    **dpi = *pi + 2;    // i = i + 2;
    printf("%d %d %d₩n", i, *pi, **dpi);

    return 0;
}
```

[결과]
0056F828 0056F828 0056F828
22 22 22
24 24 24

3 값이 같은 것을 가리킴을 알기!

간접 연산자와 증감 연산자 활용

- 간접 연산자 *를 증감 연산자 ++, --와 함께 사용하는 경우
  - 간접 연산자 * : 전위 연산자. 연산자 우선순위가 2위
  - 증감 연산자 ++, -- : 전위이면 2위, 후위이면 1위

- 사용 사례
  - *p++는 *(p++) : (*p)++ 가 아님.
  - ++*p는 ++(*p)
  - *++p는 *(++p)

연산식		결과값	연산 후 *p의 값	연산 후 p 증가
*p++	*(p++)	*p: p의 간접참조 값	변동 없음	p+1: p 다음 주소
*++p	*(++p)	*(p+1): p 다음 주소 (p+1) 간접참조 값	변동 없음	p+1: p 다음 주소
(*p)++		*p: p의 간접참조 값	*p가 1 증가	p: 없음
++*p	++(*p)	*p + 1: p의 간접참조 값에 1 증가	*p가 1 증가	p: 없음

다양한 포인터 연산
(variousop.c)

```
#include <stdio.h>

int main(void) {
    int i;
    int *pi = &i;           //포인터 선언
    int **dpi = &pi;    //이중포인터 선언

    *pi = 5;
    *pi += 1; //*pi = *pi + 1와 같음
    printf("%d₩n", i);

    printf("%d₩n", (*pi)++);
    printf("%d₩n", *pi);

    *pi = 10;
    printf("%d₩n", ++*pi);   //++*pi과 ++(*pi)는 같음
    printf("%d₩n", ++**dpi); //++**dpi과 ++(**dpi)는 같음
    printf("%d₩n", i);
```

[결과]
6
6
7
11
12
12

++이 후위연산이므로 변환문자 대체에 사용된 후 i 값 증가

++이 전위연산이므로 변환문자 대체 전에 i 값 증가

포인터 상수
(const 이용)

```
int main(void) {
    int i = 10, j = 20;
    int *p = &i;
    *p = 5;
    p = &j;
    return 0;
}
```

p가 가리키는 값을 수정 못함

p를 수정 못함

```
int main(void)
{
    int i = 10, j = 20;
    const int *p = &i;

    *p = 5;
    p = &j;

    return 0;
}
```

```
int main(void)
{
    int i = 10, j = 20;
    int const *p = &i;

    *p = 5;
    p = &j;

    return 0;
}
```

```
int main(void)
{
    int i = 10, j = 20;
    int *const p = &i;

    *p = 5;
    p = &j;

    return 0;
}
```

빨간 색 문장이 오류!

포인터 상수

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

int main() {
    int i = 10, j = 20;
    const int *p = &i; //*p로써 간접참조 값을 수정할 수 없음

    p = &j;
    printf("%d₩n", *p);

    double d = 7.8, e = 2.7;
    double * const pd = &d;
    //pd = &e; //pd가 상수로써 다른 주소 값을 가질 수 없음
    *pd = 4.4;
    printf("%f₩n", *pd);

    return 0;
}
```

[결과]
20
4.400000

프로그래밍 연습 01

```c
#include <stdio.h>

int main(void) {
    char ch = '*';
    char *pch = &ch;

    printf("주소=%p 코드값=%02x 문자=%c₩n",
    pch, *pch, *pch);
    return 0;
}
```

[결과]
주소=00FBFCF7 코드값=2a 문자=*

프로그래밍 연습 03

```c
#include <stdio.h>

int main(void) {
    int data1 = 10, data2 = 20;
    int *p1 = &data1, *p2 = &data2;
    int **dp = &p1;
    int sum = 0;

    *p1 = 100;
    *p2 = 200;
    sum = **dp + *p2;
    printf("sum=%d₩n", sum);
}
```

[결과]
sum=300

프로그래밍 연습 05

```c
#include <stdio.h>

int main(void) {
    int n = 0x01020304;
    char *p = (char *)&n;
    int i;

    for (i = 3; i >= 0; i--)
        printf("%02x", *(p+i));
    printf("₩n");
    return 0;
}
```

[결과]
01020304

프로그래밍 연습 06

```c
#include <stdio.h>

int main(void) {
    int n = 0x41424344;
    char *p = (char*)&n;

    p++;
    printf("p의 문자 : %c₩n", *p);

    printf("*p++=%c₩n", *p++);
    printf("*++p=%c₩n", *++p);
    printf("(*p)++=%c₩n", (*p)++);
    printf("++*p=%c₩n", ++*p);

    return 0;
}
```

[결과]
p의 문자 : C
*p++=C
*++p=A
(*p)++=A
++*p=C

프로그래밍 연습 08

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void) {
    double d1, d2;
    double *pd1 = &d1, *pd2 = &d2;

    printf("두 실수 입력 : ");
    scanf("%lf %lf", &d1, &d2);

    printf("덧셈=%lf₩n", *pd1 + *pd2);
    printf("뺄셈=%lf₩n", *pd1 - *pd2);
    printf("곱셈=%lf₩n", *pd1 * *pd2);
    printf("나눗셈=%lf₩n", *pd1 / *pd2);

    return 0;
}
```

[결과]
두 실수 입력 : 1.5 2.34
덧셈=3.840000
뺄셈=-0.840000
곱셈=3.510000
나눗셈=0.641026