6장 포인터

한림대학교 소프트웨어융합대학 양은샘.

6장 포인터

- 안녕하세요? 여러분!
- 이번 장에서는 같은 포인터에 대해 학습합니다.
- C 언어에서는 포인터없이 프로그램하는 것이 불가능하므로 아주 중요한 개념입니다.
- 지난 시간에 학습한 내용을 리뷰한 후 학습을 시작하도록 하겠습니다.

지난 시간 Review

5.1 배열의 형식

5.2 배열의 참조

5.3 문자와 문자열

5.4 배열의 초기화

5.5 2차원 배열

5.6 다차원 배열

5.7 배열 응용

□ 개념 확인 학습

□ 적용 확인 학습

□ 응용 프로그래밍

6장 포인터

6.1 포인터의 의미와 사용

6.2 포인터 연산

6.3 배열과 포인터

6.4 문자열 상수 포인터

6.5 이차원 배열과 포인터

6.6 포인터 배열

6.7 다중 포인터

6.8 포인터 응용

□ 개념 확인 학습

□ 적용 확인 학습

□ 응용 프로그래밍

학습 목표

- 포인터를 사용하여 데이터를 저장 할 수 있다.
- 문자 배열과 문자열 배열을 포인터로 접근하고 선택적으로 이용할 수 있다.
- 포인터를 사용하여 2차원 배열의 데이터에 접근 할 수 있다.
- 포인터 적용하여 다양한 조건의 문제들을 해결 할 수 있다.

- 개념 확인 학습으로 배운 내용을 정리한다.
- 적용 확인 학습으로 개념 습득 여부를 확인한다.
- 응용 프로그래밍으로 문제해결력을 키운다.

포인터의 의미

- 포인터(Pointer)란 임의의 기억장소(메모리)에 대한 주소를 가리키며
- 포인터 변수는 메모리 주소를 저장할 수 있는 변수를 의미.

- 데이터가 저장된 기억장소의 주소 값을 포인터 변수가 갖게 되면,
  - 그 주소에 해당하는 값은 포인터 연산자를 이용하여 간접적으로 참조할 수 있음.

포인터 변수의 선언

- 포인터 변수의 선언은 *(asterisk)를 사용하며 형식은 다음과 같음.

포인터의 형식

데이터형 *포인터변수명;

포인터 변수의 선언 예

```
char *pch;          //문자형 포인터 변수 pch를 선언
int *pi;            //정수형 포인터 변수 pi를 선언
double *pd;         //실수형 포인터 변수 pd를 선언
```

[표 6.1] 포인터 연산자

연산자	기능
&	일반 변수의 주소를 추출해주는 연산자.
*	포인터 변수가 가리키고 있는 주소에 들어있는 값. 즉, 번지에 있는 값을 참조하는 연산자.

포인터 변수의 메모리 구조 예

- 프로그램에서 사용되는 일반 변수들은
  - 주기억장치라는 메모리에 저장.
  - 이때 효율적인 메모리 관리를 위해 컴퓨터는 메모리에 바이트 단위로 고유한 주소를 부여.
  - 이 메모리의 주소를 알고자 할 때 사용되는 것이 주소 연산자(&).

- 간접 참조연산자(*) 연산자는
  - 포인터 변수를 선언할 때와. 포인터 변수가 가리키고 있는 주소에 들어있는 실제 값을 참조할 때 사용.
  - 값을 참조할 때는 포인터 변수가 가리키는 주소에서부터 각 자료형의 크기만큼의 데이터를 참조.

포인터 변수의 메모리 구조 예

```
char ch='A', *pch;
pch = &ch;
//*pch는 'A'가 됨.

int i=100, *pi;
pi = &i;
//*pi는 100이 됨.
```

	주소	주소의 실제 값
	...	...
ch	1002 번지	A
	1003 번지	
i	1004 번지	100
	1005 번지	
	1006 번지	
	1007 번지	
	...	...

포인터 연산자의 사용

[예제 6.1] 포인터 연산자의 사용

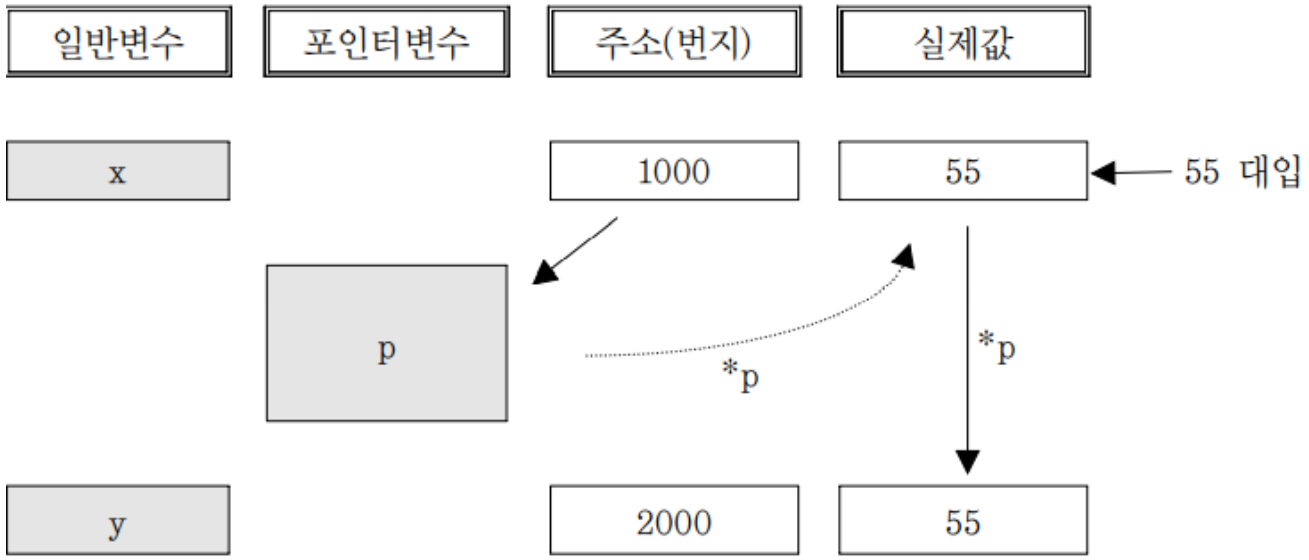
```
#include <stdio.h>
int main()
{
        int i = 100, *pi;

        pi = &i;
        printf("i의 주소 = %ld₩n", pi);
        printf("i의 값 = %ld₩n", *pi);
        return 0;
}
```

```
i의 주소 = 6422296
i의 값 = 100
```

포인터 연산에 대한 주소와 값의 관계

```
int *p;     //포인터 변수 *p 선언
int x, y;   //일반 변수 x, y 선언
x = 55;     //일반 변수 x에 55 값을 대입
p = &x;     //일반 변수 x의 주소 값을 포인터 변수 p에 대입
y = *p;     //포인터 p 가리키는 곳의 실제 값을 y에 대입
            //결국 일반 변수 y에는 55의 값이 대입 됨.
```



[그림 6.1] 포인터 연산에 대한 주소와 값의 관계

포인터를 이용한 값의 복사

[예제 6.2] 포인터를 이용한 값의 복사

```
#include <stdio.h>
int main()
{
      int *p, x, y;

      x = 55;
      p = &x;
      y = *p;

      printf("x의 주소 = %u₩n", &x);
      printf("p의 주소 = %u₩n", p);
      printf("y의 주소 = %u₩n", &y);

      printf("x의 값 = %u₩n", x);
      printf("*p의 값 = %u₩n", *p);
      printf("y의 값 = %u₩n", y);
      return 0;
}
```

```
x의 주소 = 6422296
p의 주소 = 6422296
y의 주소 = 6422292
x의 값 = 55
*p의 값 = 55
y의 값 = 55
```

포인터 변수의 크기

[예제 6.3] 포인터 변수의 크기

```
#include <stdio.h>
int main()
{
    int i = 1, *pi;
    char c = 'a', *pc;
    float f = 0.1, *pf;
    double d = 0.1, *pd;

    pi = &i;  pc = &c;  pf = &f;  pd = &d;

    printf("sizeof(int)=%d, sizeof(pi)=%d₩n", sizeof(int), sizeof(pi));
    printf("sizeof(char)=%d, sizeof(pc)=%d₩n", sizeof(char), sizeof(pc));
    printf("sizeof(float)=%d, sizeof(pf)=%d₩n", sizeof(float), sizeof(pf));
    printf("sizeof(double)=%d, sizeof(pd)=%d₩n", sizeof(double), sizeof(pd));
    return 0;
}
```

```
sizeof(int)=4, sizeof(pi)=4
sizeof(char)=1, sizeof(pc)=4
sizeof(float)=4, sizeof(pf)=4
sizeof(double)=8, sizeof(pd)=4
```

포인터의 자료형

- 포인터 변수의 자료형은
  - 객체의 처리 방법을 결정하기 때문에 포인터의 형과 포인터가 가리키는 자료형은 같아야 함.

[예제 6.4] 포인터의 자료형

```
#include <stdio.h>
int main()
{
      int x;
      double *px;

      px = &x;            //ERROR : 포인터 px에 정수 x의 주소를 치환한다.
                          //int는 double보다 저장 길이가 짧다.
      *px = 100.23;       //포인터를 이용하여 x에 값을 치환할 수 없다.
      printf("x = %d", x);
      return 0;
}
```

```
    7 |   px = &x;
      |      ^
x = 1374389535
```

널 포인터(Null Pointer)

널 포인터(Null Pointer)

어떤 것도 가리키지 않은 포인터

널 포인터 의미

```
#define NULL ((void *)0)
//(void *)는 아직 결정되지 않은 자료형의 주소
//<stdio.h>에 정의되어 있는 포인터 상수로서 0번지의 주소 값을 의미
```

널 포인터 사용

```
int *p=NULL;
```

포인터 연산

- 포인터 변수와 관련된 연산식에 사용되는 연산자는
  - '+', '++', '-', '--'로 정수만을 더하거나 뺄 수 있음.

- 포인터 연산은 일반적인 연산과 달리 포인터의 기초형과 관련되어 수행 됨.
  - 자료형의 크기만큼 주소 값이 증가하거나 감소.
  - 더하기 1을 한다고 해도, 자료형에 따라 char이면 sizeof(char) byte만큼, int이면 sizeof(int) bytes만큼 증가하거나 감소.

포인터 연산의 의미

```
int i=100, *pi;
pi = &i;
pi++; //pi = pi + 1;
//pi가 가리키는 메모리 주소를 대상체의 자료형 크기만큼 증가시킨다. 대상체
가 정수형이므로 sizeof(int)=4bytes 증가하게 된다. 즉, 다음 위치의 정수를
가리킨다.
```

포인터 변수와 증감 연산자와의 관계

연산 형태	의미
(*p)++	현재 가리키고 있는 곳의 값을 1증가 시킨다.
*p++	*p; p++; 두 문장의 결합과 같다. 값을 참조한 후(*p), 주소를 증가시켜(p++) 다음 원소를 가리킨다.
*(p+1)	주소를 증가시켜 다음 원소를 가리킨 후, 그곳의 값을 참조한다.
*++p	p++; *p; 두 문장의 결합과 같다. 주소를 증가시켜(p++) 다음 원소를 가리킨 후, 그곳의 값(*p)을 참조한다.

포인터 연산과 메모리 주소

[예제 6.5] 포인터 연산과 메모리 주소

```
#include <stdio.h>
int main()
{
    char *cp, ch;
    int *ip, i;
    double *dp, d;

    cp = &ch;  ip = &i;  dp = &d;
    printf("cp = %p, ip = %p, dp= %p₩n", cp, ip, dp); //현재의 주소

    cp++;  ip++;  dp++; //포인터 증가

    printf("cp++=%p, ip++=%p, dp++=%pWn", cp, ip, dp); //새로운 주소
    return 0;
}
```

```
cp = 0061FF13, ip = 0061FF0C, dp= 0061FF00
cp++=0061FF14, ip++=0061FF10, dp++=0061FF08
```

[예제 6.6] 포인터 연산

```
#include <stdio.h>
int main()
{
    char *p, ch='A';

    p = &ch;
    for (int i = 0; i < 20; i++) {
        printf("%c", (*p) + i);
    }
    return 0;
}
```

```
ABCDEFGHIJKLMNOPQRST
```

배열과 포인터

- 배열명은 그 배열의 선두 번지를 가리키는 일종의 포인터 상수
  - 인덱스(Index)를 이용한 배열의 접근은 포인터를 사용하여 표현 가능.
  - 단, 포인터 변수는 수정 가능하지만, 배열명은 배열의 첫 번째 요소를 참조하는 고정된 주소.

[표 6.3] 포인터와 배열의 값과 주소

int in[5], *p; p=in; //배열의 시작 주소를 포인터변수에 저장					
값	in[0]	in[1]	in[2]	in[3]	in[4]
값	p[0]	p[1]	p[2]	p[3]	p[4]
값	*in	*(in+1)	*(in+2)	*(in+3)	*(in+4)
값	*p	*(p+1)	*(p+2)	*(p+3)	*(p+4)
주소	&in[0]	&in[1]	&in[2]	&in[3]	&in[4]
주소	&p[0]	&p[1]	&p[2]	&p[3]	&p[4]
주소	in	in+1	in+2	in+3	in+4
주소	p	p+1	p+2	p+3	p+4

포인터를 이용한 배열의 접근

[예제 6.7] 포인터를 이용한 배열의 접근 (정수형)

```
#include <stdio.h>
int main()
{
    int *p, array[] = {1, 3, 5, 7, 9, 11};

    p = &array[2];

    printf("*p = %d, ", *p);

    (*p)++;
    printf("(*p)++ = %d, ", *p);

    *(p++);
    printf("*(p++) = %d₩n", *p);
    return 0;
}
```

```
*p = 5, (*p)++ = 6, *(p++) = 7
```

[예제 6.8] 포인터를 이용한 배열의 접근 (문자열형)

```
#include <stdio.h>
int main()
{
    char *p, str[] = "rainy";

    p = str;

    printf("p의 주소 = %u, 값 = %c₩n", p, *p);
    printf("str의 주소 = %u, 값 = %c₩n", str, str[0]);
    printf("p+1의 주소 = %u, 값 = %c₩n", p+1, *(p+1));
    printf("str+1의 주소 = %u, 값 = %c₩n", str+1, str[1]);

    printf("str = ");
    while(*p) {
        printf("%c", *p);
        p++;
    }
    return 0;
}
```

```
p의 주소 = 6422294, 값 = r
str의 주소 = 6422294, 값 = r
p+1의 주소 = 6422295, 값 = a
str+1의 주소 = 6422295, 값 = a
str = rainy
```

문자열 상수 포인터

- “포인터변수명”은 문자열이 저장되어있는 기억 장소의 시작 번지.

문자열에 대한 포인터의 초기화 형식

데이터형 *포인터변수명 = “문자열”;

문자열에 대한 포인터 예

```
char *p_string = "C Programming"; //문자열의 길이는 널 포함 14
```

이차원 배열과 포인터

일차원 배열과 포인터

```
int num[5], *p;
p = num;

char str[10], *p;
p = str;
```

이차원 배열과 포인터

```
int nums[][5], (*p)[5]; //이 때 괄호는 반드시 필요합니다.
p =  nums;

char strs[][10], (*p)[10]; //이 때 괄호는 반드시 필요합니다.
p = strs;
```

[예제 6.9] 이차원 배열과 포인터 예

```
pn[0][0]=9, *((pn[0])+0)=9, *(*(pn+0)+0)=9
  9  8  7  6  5  4  3  2  1  0
        red      green     yellow
```

```
#include <stdio.h>
int main()
{
    int num[][5] = {9, 8, 7, 6, 5, 4, 3, 2, 1};
    int(*pn)[5];
    char strs[][10] = {"red", "green", "yellow"};
    char(*ps)[10];

    pn = num;
    ps = strs;

    printf("pn[0][0]=%d, *((pn[0])+0)=%d, *(*(pn+0)+0)=%d₩n",
        pn[0][0], *((pn[0]) + 0), *(*(pn + 0) + 0));

    for (int i = 0; i < sizeof(num) / sizeof(num[0]); i++) {
        for (int j = 0; j < sizeof(num[0]) / sizeof(int); j++) {
            printf("%3d", *(*(pn + i) + j));
        }
    }
    puts("");
    for (int i = 0; i < sizeof(strs) / sizeof(strs[0]); i++) {
        printf("%10s", *(ps + i));
    }
    return 0;
}
```

포인터 배열

- 포인터 배열은 포인터들로 구성된 배열
  - 포인터 형의 변수가 배열의 되는 경우.

포인터 배열 형식

```
데이터형  *포인터명[포인터개수];
```

포인터 배열 예

```
float fa, fb, fc;
float *pf[3] = { &fa,  &fb,  &fc };
```

포인터 배열 예

[예제 6.10] 포인터 배열

```
*(pf[0])=1.10, *(*(pf+1))=2.20, *(*(pf+2))=3.30
실수입력 : 5.5
실수입력 : 6.6
실수입력 : 7.7
fa=5.50, fb=6.60, fc=7.70
```

```
#include <stdio.h>
int main()
{
    float fa = 1.1, fb = 2.2, fc = 3.3;
    float *pf[3] = {&fa, &fb, &fc};

    printf("*(pf[0])=%.2f, *(*(pf+1))=%.2f, *(*(pf+2))=%.2f₩n",
           *(pf[0]), *(*(pf+1)), *(*(pf+2)));

    for (int i = 0; i < 3; i++) {
        printf("실수입력 : ");
        scanf("%f", pf[i]);
    }
    printf("fa=%.2f, fb=%.2f, fc=%.2f₩n", fa, fb, fc);

    return 0;
}
```

문자열 배열 및 메모리 구조

문자열 배열 및 메모리 구조

```
char nums[3][6] = {"zero", "one", "three"};
```

	[0]	[1]	[2]	[3]	[4]	[5]
nums[0]	z	e	r	o	₩0	?
nums[1]	o	n	e	₩0	?	?
nums[2]	t	h	r	e	e	₩0

포인터 배열 및 메모리 구조

```
char *pnums[3] = {"zero", "one", "three"};
```

	[0]	[1]	[2]	[3]	[4]	[5]
pnums[0]	z	e	r	o	₩0	
pnums[1]	o	n	e	₩0		
pnums[2]	t	h	r	e	e	₩0

[예제 6.11] 포인터 배열 (문자열)

```
#include <stdio.h>
int main()
{
    char a[3][6] = {"red", "green", "yellow"};
    char *b[] = {"red", "green", "yellow"};
    int size = sizeof(a)/sizeof(a[0]);

    for (int i = 0; i < size; i++) {
        printf("a[%d]=%s, ", i, a[i]);
    }
    puts("");
    for (int i = 0; i < size; i++) {
        printf("b[%d]=%s, ", i, b[i]);
    }
    puts("");
    for (int i = 0; i < size; i++) {
        printf("*(b+%d)=%s, ", i, *(b+i));
    }
    return 0;
}
```

```
a[0]=red, a[1]=green, a[2]=yellow,
b[0]=red, b[1]=green, b[2]=yellow,
*(b+0)=red, *(b+1)=green, *(b+2)=yellow,
```

다중 포인터

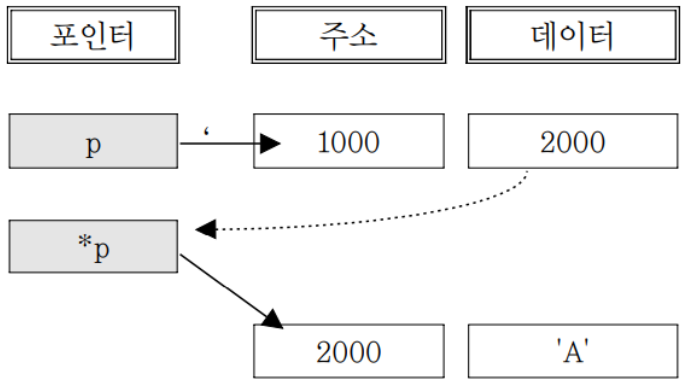
이중 포인터 형식

데이터형 **포인터명;

이중 포인터 예

```
char **p;
//**p : *p가 가리키고 있는 주소(값)에 저장된 값을 의미한다.
//*p : p가 가리키고 있는 주소에 저장된 값(주소)을 의미한다.
```

이중 포인터 메모리 블록도 예



[예제 6.12] 이중 포인터

```
#include <stdio.h>
int main()
{
    int *ptr1, **ptr2;
    int num = 300;

    ptr1 = &num;
    ptr2 = &ptr1;

    printf("num=%d, *ptr1=%d, **ptr2=%d₩n", num, *ptr1, **ptr2);
    return 0;
}
```

```
num=300, *ptr1=300, **ptr2=300
```

포인터 응용 (1)

[예제 6.13] 포인터를 사용하여 입력 받은 값의 평균을 구하기

```
#include <stdio.h>
int main()
{
      double sum=0, num[5], *p;
      p=num;
      for (int i=0; i<sizeof(num)/sizeof(double); i++) {
            printf("숫자를 입력하세요 : ");
            scanf("%lf", p+i);
      }
      for (int i=0; i<sizeof(num)/sizeof(double); i++) {
            sum += *(p+i);
      }
      printf("평균은 = %lf", sum/(sizeof(num)/sizeof(double)));
      return 0;
}
```

```
숫자를 입력하세요 : 2
숫자를 입력하세요 : 2
숫자를 입력하세요 : 2
숫자를 입력하세요 : 1
숫자를 입력하세요 : 1
평균은 = 1.600000
```

포인터 응용 (2)

[예제 6.14] 포인터를 사용하여 배열의 내용에서 소수 찾기

```
배열은    5    7   22   53   88   71  101
소수는    5    7   53   71  101
```

```
#include <stdio.h>
int main()
{
    int num[] = { 5, 7, 22, 53, 88, 71, 101}, *p;
    int deno, input, count = 0;

    p=num;

    printf("₩n 배열은 ");
    for (int i=0; i<sizeof(num)/sizeof(int); i++) {
        printf("%5d", *(p+i));
    }
    printf("₩n 소수는 ");
    for (int i=0; i<sizeof(num)/sizeof(int); i++) {
        for (deno=2; (*(p+i) % deno) !=0 ; deno++); //수행문 없음.

        if (deno == *(p+i)) {
                printf("%5d", *(p+i));
        }
    }
    return 0;
}
```

포인터 응용 (3)

[예제 6.15] 포인터를 사용하여 입력된 숫자들의 빈도수 출력

```
#include <stdio.h>
#include <conio.h>
int main()
{
        char ch;
        int frequency[10] = { 0 }, *p;

        p = frequency;

        printf("숫자가 아니면 반복이 종료됩니다.₩n");
        printf("0에서 9까지의 정수를 입력하세요.₩n");
        do {
                ch = getche();
                if(ch<'0' || ch>'9') break;
                p[ch-'0']++;
        } while(1);

        for(int i=0; i<10; i++) {
                if(*(p+i))
                        printf("\n%d는 %d회 입력하셨습니다.", i, *(p+i));
        }
        return 0;
}
```

```
숫자가 아니면 반복이 종료됩니다.
0에서 9까지의 정수를 입력하세요.
33333366666999
3는 6회 입력하셨습니다.
6는 5회 입력하셨습니다.
9는 3회 입력하셨습니다.
```

포인터 응용 (4)

[예제 6.16] 포인터를 사용하여 입력 받은 문자열의 길이 구하기

```
#include <stdio.h>
int main()
{
      char str[30], *p;
      int i;

      while (1) {
            p=str; //p는 변화하므로 언제나 배열의 시작을 가리키게 함.
            printf("₩n문자열 입력 : ");
            gets(p);

            if (*p == '₩0') break;

            for (i=0; *p; i++, p++);
            printf("문자열의 길이는 = %d", i);
      }
      return 0;
}
```

```
문자열 입력 : apple
문자열의 길이는 = 5
문자열 입력 : I like C.
문자열의 길이는 = 9
문자열 입력 :
```

포인터 응용 (5)

[예제 6.17] 포인터를 사용하여 입력 받은 문자열 복사하기

```
#include <stdio.h>
int main()
{
      char stra[30], strb[30], *sa, *sb;

      sa = stra;
      sb = strb;

      printf("₩n문자열 입력 : ");
      gets(sa);

      while(*sa) {
           *sb = *sa;
           sa++;
           sb++;
      };
      *sb = '₩0';
      printf("복사된 문자열 = %s", strb);

      return 0;
}
```

```
문자열 입력 : I like C.
복사된 문자열 = I like C.
```

개념 확인학습 & 적용 확인학습 & 응용 프로그래밍

- 다음 파일에 있는 문제들의 해답을 스스로 작성 해 보신 후 개념 & 적용 확인 학습 영상을 학습 하시기 바랍니다.
  - c_06장_포인터_ex.pdf
- 퀴즈와 과제가 출제되었습니다.
  - 응용 프로그래밍 영상을 학습하신 후 과제와 퀴즈를 수행 하시기 바랍니다.

Q & A

- “포인터”에 대한 학습이 모두 끝났습니다.

- 모든 내용을 이해 하셨나요?
- 아직 이해가 안되는 내용이 있다면 다시 한번 복습하시기 바랍니다.
- 질문은 한림 SmartLEAD 쪽지 또는 e-mail 또는 전화상담을 이용하시기 바랍니다.

- 퀴즈와 과제가 출제되었습니다. 마감시간에 늦지 않도록 주의해 주세요.

- 다음 시간에는 “함수”에 대해 알아보겠습니다.

- 수고하셨습니다.^^