C - 포인터(pointer)

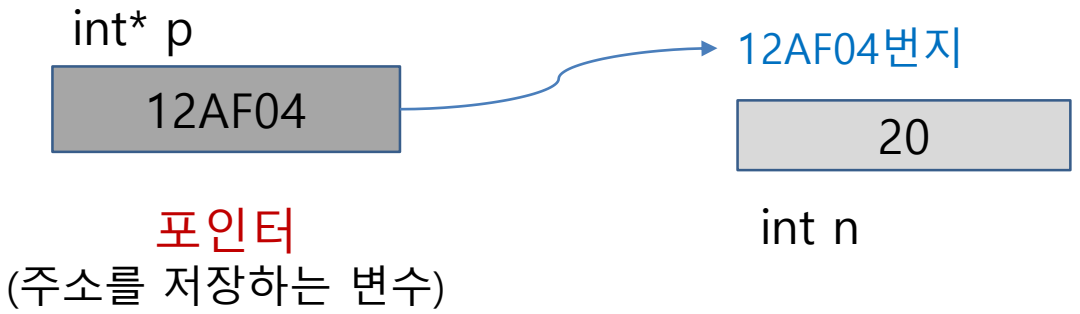
Visual Studio 2022

포인터(Pointer)

- 포인터란?

  모든 메모리는 주소(address)를 갖는다. 이러한 **메모리 주소를 저장**하기 위해 사용되는 변수를 포인터 변수라 한다.

  포인터 변수를 선언할 때에는 데이터 유형과 함께 '*' 기호를 써서 나타낸다.



포인터(Pointer)

- 포인터 변수의 선언 및 값 저장
  - 선언

자료형* 포인터 이름

```
char*  c;      // char형 포인터
int* n;        // int형 포인터
double* d;     // double형 포인터
```

  - 포인터의 크기 – 모든 자료형에서 8바이트로 동일하다.
    포인터에 저장할 수 있는 값은 메모리 번지뿐이며, 따라서 모든 포인터 변수는 동일한 크기의 메모리가 필요함.

sizeof(포인터)

포인터(Pointer)

- 정수형 포인터 변수

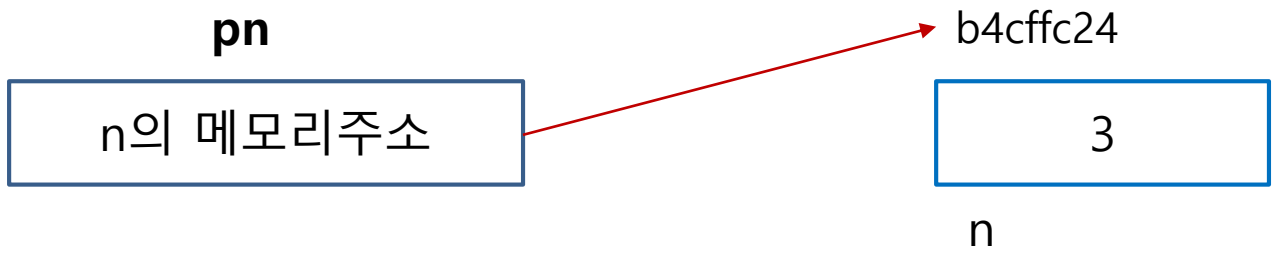
```
int n;
int* pn;

n = 3;
pn = &n;

printf("변수의 값: %d\n", n);
printf("변수의 메모리 번지: %x\n", &n);
printf("포인터 변수의 값: %x, \n", pn);
printf("포인터의 메모리 번지: %x\n", &pn);
printf("포인터가 가리키는 메모리의 값: %d\n", *pn);  //역참조

//자료형의 크기
printf("변수의 자료형의 크기: %dByte\n", sizeof(n));
printf("포인터의 자료형의 크기: %dByte\n", sizeof(pn));
```

포인터(Pointer)

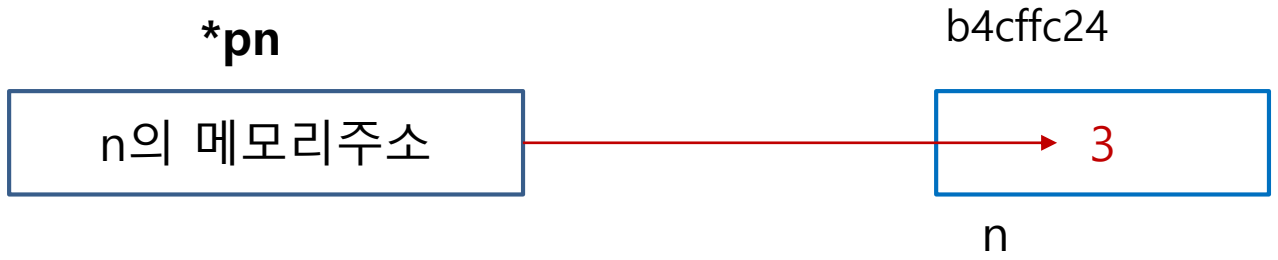
- 역참조 연산자(*)

포인터를 선언할 때도 *를 사용하고, 역참조 할때도 *를 사용

- 포인터는 변수의 주소만 가리킴



- 역참조는 주소에 접근하여 값을 가져옴



포인터(Pointer)

- 문자형 포인터 변수

```
char c;
char* pc;

c = 'A';
pc = &c;

printf("변수의 값: %c\n", c);
printf("변수의 메모리 번지: %x\n", &c);
printf("포인터의 값: %x\n", pc);
printf("포인터의 메모리 번지: %x\n", &pc);
printf("포인터가 가리키는 메모리 값: %c\n", *pc);

//변수와 포인터의 자료형의 크기
printf("변수의 자료형 크기: %dByte\n", sizeof(c));
printf("포인터의 자료형 크기: %dByte\n", sizeof(pc));
```

포인터(Pointer)

- 포인터 사용 예제

```c
//정수형 포인터 사용
int a = 10;
int* b;

printf("a의 값은 %d\n", a);

b = &a;
*b = 20;

printf("a의 값은 %d\n", a);
```

```c
//문자형 포인터 사용
char c1 = 'A';
char* c2;

printf("c1의 값은 %c\n", c1);

c2 = &c1;
*c2 = 'B';

printf("c1의 값은 %c\n", c1);
```

```
a의 값은 10
a의 값은 20
c1의 값은 A
c1의 값은 B
```

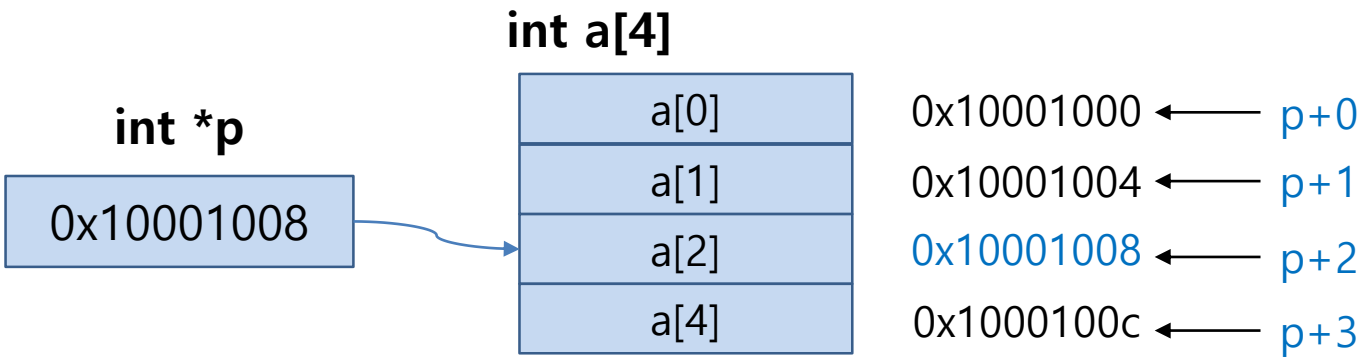
배열과 포인터

- 배열과 포인터

  - 배열은 데이터를 연속적으로 메모리에 저장한다.

  포인터 역시 메모리에 데이터를 저장하거나 저장된 데이터들을 읽어올수 있다.

```
int a[4], *p;        // 배열 a와 포인터 p선언
p = &a[2]            // p에 배열의 두 번째 항목 주소 복사
```



배열과 포인터(Pointer)

- 정수형 배열과 포인터

```
int a[4] = { 10, 20, 30, 40 };
int* pa;
int i;

//a는 배열의 시작 주소이다.
/*printf("%d %x %x\n", a[0], &a[0], a);  //a -> a + 0
printf("%d %x %x\n", a[1], &a[1], a + 1);
printf("%d %x %x\n", a[2], &a[2], a + 2);
printf("%d %x %x\n", a[3], &a[3], a + 3);*/

//배열 요소의 값과 주소 출력
for (i = 0; i < 4; i++)
{
    printf("%d %x %x\n", a[i], &a[i], a + i);
}
printf("=======================================\n");
```

배열과 포인터(Pointer)

- 정수형 배열과 포인터

```
pa = a;  //pa포인터에 a의 주소 저장
printf("포인터 pa의 값: %x\n", pa);
printf("포인터 *pa가 가리키는 메모리의 값: %d\n", *pa);

/*printf("%x %d\n", pa + 1, *(pa + 1));
printf("%x %d\n", pa + 2, *(pa + 2));
printf("%x %d\n", pa + 3, *(pa + 3));*/

//포인터 요소의 값과 주소 출력
for (i = 0; i < 4; i++)
{
    printf("%x %d\n", pa + i, *(pa + i));
}

//배열과 포인터의 크기
printf("a의 크기: %dbyte, pa의 크기: %dbyte\n", sizeof(a), sizeof(pa));
```

```
10 adaffbe8 adaffbe8
20 adaffbec adaffbec
30 adaffbf0 adaffbf0
40 adaffbf4 adaffbf4
========================================
포인터 pa의 값: adaffbe8
포인터 *pa가 가리키는 메모리의 값: 10
adaffbe8 10
adaffbec 20
adaffbf0 30
adaffbf4 40
a의 크기: 16byte, pa의 크기: 8byte
```

배열과 포인터(Pointer)

- 배열과 포인터의 연산

```
int x = 10, y = 20, z;
int* arr[3]; //정수형 포인터 배열 선언

// 배열에 주소 저장
arr[0] = &x;
arr[1] = &y;
arr[2] = &z;

//배열 선언과 동시에 초기화
//int* arr[3] = { &x, &y, &z };

*arr[2] = *arr[0] + *arr[1];

printf("arr[2]의 값: %d\n", *arr[2]);
printf("z의 값: %d\n", z);
```

배열과 포인터(Pointer)

- 배열과 포인터의 연산

```
int a[5] = { 1, 2, 3, 4, 5 };
int* b;
int i;

printf("기존 배열 a의 값 출력\n");
for (i = 0; i < 5; i++)
    printf("%d ", a[i]);

printf("\n배열 a의 각 요소에 10을 더하여 변경\n");
b = a;

for (i = 0; i < 5; i++)
    *(b + i) += 10;
```

배열과 포인터(Pointer)

- 배열과 포인터의 연산

```
printf("\n변경된 배열 a의 값 출력 1\n");
for (i = 0; i < 5; i++)
    printf("%d ", a[i]);
printf("\n");

printf("\n변경된 배열 a의 값 출력 2\n");
for (i = 0; i < 5; i++)
    printf("%d ", *(b + i));
```

```
기존 배열 a의 값 출력
1 2 3 4 5
배열 a의 각 요소에 10을 더하여 변경

변경된 배열 a의 값 출력 1
11 12 13 14 15

변경된 배열 a의 값 출력 2
11 12 13 14 15
```

배열과 포인터(Pointer)

- 배열과 포인터 – 문자열 입력

```
//배열과 포인터
char a[20];
char* b;

printf("문자열을 입력하세요: ");
scanf_s("%s", a, sizeof(a));

b = a;
printf("입력된 문자열: %s\n", b);
```

문자열과 포인터(Pointer)

- 문자열과 포인터

  문자열은 문자들이 메모리 공간에 연속적으로 저장되어 있어서 주소로 관리되고, 문자열의 시작 주소를 알면 모든 문자열에 접근할 수 있다.

```c
char msg[] = "Good Luck";
char* p = msg;
int i;

//문자열 출력
printf("%s\n", msg);

//문자열은 맨 뒤에 '\0' 포함, 포인터는 8byte
printf("%d %d\n", sizeof(msg), sizeof(p));

//문자열의 시작 주소와 배열의 이름은 동일하다.
printf("%x %x\n", msg, p);
```

```
Good Luck
10 8
e96ffa08 e96ffa08
Good Luck
ood Luck
od Luck
d Luck
Good Luck
```

문자열과 포인터(Pointer)

- 문자열과 포인터

```
//포인터로 출력
printf("%s\n", p); //p + 0과 같음
printf("%s\n", p + 1);
printf("%s\n", p + 2);
printf("%s\n", p + 3);

//포인터 역참조로 출력
/*printf("%c\n", *p);
printf("%c\n", *(p + 1));
printf("%c\n", *(p + 2));
printf("%c\n", *(p + 3));*/

int size = sizeof(msg) / sizeof(msg[0]);

//문자로 출력
for (i = 0; i < size; i++)
{
    printf("%c", *(p + i));
}
```

포인터(Pointer) – 함수

- 함수의 매개변수로 포인터 사용하기

```
void changeArray(int* ptr)
{
    ptr[1] = 50;
}

int main()
{
    //배열 요소 변경 - 포인터 사용
    int arr[] = { 10, 20, 30 };

    changeArray(arr);  //int *ptr = arr

    for (int i = 0; i < 3; i++)
    {
        printf("%d\n", arr[i]);
    }

    return 0;
}
```

함수에서 포인터의 전달

- Call-by-Value(값에 의한 호출) vs Call-by-Reference(참조에 의한 호출)

값을 매개로 함수호출

callByVal(int n)

↓

callByVal(num)

주소를 매개로 함수 호출

callByRef(int *p)

↓

callbyRef(&num)

값 & 참조에 의한 호출

- 값 & 참조에 의한 호출

```
int callByVal(int);
int callByRef(int* );

int main()
{
    int num = 10;

    puts("=== 값에 의한 호출 ===");
    callByVal(num);
    printf("%d\n", num);

    puts("=== 참조에 의한 호출 ===");
    callByRef(&num);
    printf("%d\n", num);

    return 0;
}
```

```
=== 값에 의한 호출 ===
10
=== 참조에 의한 호출 ===
11
```

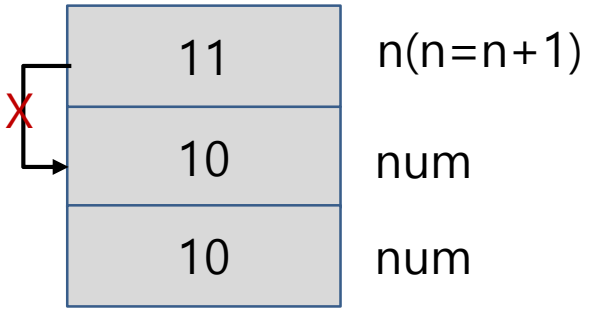
값 & 참조에 의한 호출

- 값 & 참조에 의한 호출

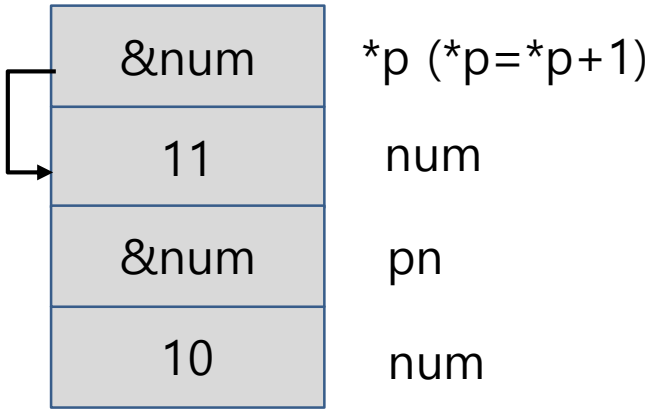
```
//값 호출 함수 정의
int callByVal(int n)
{
    n++; //n = n + 1;
    return n;
}

//참조 호출 함수 정의
int callByRef(int* p)
{
    *p = *p + 1;
    return *p;
}
```

값 & 참조에 의한 호출

- Call-By-Value(값에 의한 호출) vs Call-By-Reference(참조에 의한 호출)



매개변수 n은 호출된후 11을 반환하고 소멸됨,
main()의 지역변수 num은 그대로 10을 유지함



매개 포인터 p는 주소에 저장된 값에 접근하고 역참조 계산으로 num은 11이 됨.
호출된 후 p의 메모리 공간은 소멸됨.

값 & 참조에 의한 호출

- 값과 참조에 의한 호출

```
void func1(int, int);
void func2(int* i, int* j);

int main()
{
    int a = 3, b = 12;

    printf("--- main()내 func1 호출 ---\n");
    func1(a, b);
    printf("%d %d\n", a, b);

    printf("--- main()내 func2 호출 ---\n");
    func2(&a, &b);
    printf("%d %d\n", a, b);

    return 0;
}
```

값 & 참조에 의한 호출

- 값과 참조에 의한 호출

```c
//매개변수가 일반 변수인 함수
void func1(int i, int j)
{
    i *= 3;
    j /= 3;

    printf("%d %d\n", i, j);
}

//매개변수가 포인터인 함수
void func2(int* i, int* j)
{
    *i *= 3;
    *j /= 3;

    printf("%d %d\n", *i, *j);
}
```

```
--- main()내 func1 호출 ---
9 4
3 12
--- main()내 func2 호출 ---
9 4
9 4
```

포인터 배열에서 최대값 찾기

- 포인터 배열에서 최대값 찾기

```c
int findMax(int[], int);
int findMax2(int*, int);
int main()
{
    int arr[] = { 21, 35, 71, 2, 97, 66 };
    int max1, max2;

    max1 = findMax(arr, 6);

    max2 = findMax2(arr, 6);

    printf("최대값: %d %d\n", max1, max2);

    return 0;
}
```

포인터 배열에서 최대값 찾기

- 포인터 배열에서 최대값 찾기

```c
//매개변수를 배열로 전달
int findMax(int arr[], int len)
{
    int maxVal, i;
    maxVal = arr[0];
    for (i = 1; i < len; i++)
    {
        if (arr[i] > maxVal) {
            maxVal = arr[i];
        }
    }

    return maxVal;
}
```

```c
//매개변수를 포인터로 전달
int findMax2(int* arr, int len)
{
    int maxVal, i;
    maxVal = *(arr + 0);

    //printf("%d\n", maxVal);

    for (i = 1; i < len; i++)
    {
        if (*(arr + i) > maxVal) {
            maxVal = *(arr + i);
        }
    }
    return maxVal;
}
```

포인터 배열에서 최대값 찾기

- 포인터 배열에서 최대값의 위치 찾기

```c
//매개변수를 배열로 전달
int findMaxIdx(int arr[], int len)
{
    int maxIdx, i;
    maxIdx = 0;
    for (i = 1; i < len; i++)
    {
        if (arr[i] > arr[maxIdx]) {
            maxIdx = i;
        }
    }

    return maxIdx;
}
```

```c
//매개변수를 포인터로 전달
int findMaxIdx2(int *arr, int len)
{
    int maxIdx, i;
    maxIdx = 0;
    for (i = 1; i < len; i++)
    {
        if (*(arr + i) > *(arr + maxIdx))
        {
            maxIdx = i;
        }
    }

    return maxIdx;
}
```

포인터 배열에서 최대값 찾기

- 포인터 배열에서 최대값의 위치 찾기

```
int findMaxIdx(int[], int);
int findMaxIdx2(int*, int);
int main()
{
    int arr[] = { 21, 35, 71, 2, 97, 66 };
    int maxIdx1, maxIdx2=0;

    //최대값
    maxIdx1 = findMaxIdx(arr, 6);
    maxIdx2 = findMaxIdx2(arr, 6);

    printf("최대값의 위치: %d %d\n", maxIdx1, maxIdx2);

    return 0;
}
```

문자열과 포인터(Pointer)

- 문자열 포인터로 단어 저장하기

```
//문자열 포인터로 단어 저장하기
char* animals[] = {"ant", "bear", "chicken", "deer", "elephant"};
int len = sizeof(words) / sizeof(words[0]);

//특정 단어 접근
/*printf("%s\n", animals[0]);
printf("%s\n", animals[2]);*/

//전체 조회
for (int i = 0; i < len; i++)
{
    printf("%s\n", animals[i]);
}
```

실습 – 영어 타이핑 게임

- **게임 방법**
  - 게임 소요 시간을 측정함
  - 컴퓨터가 저장된 영어 단어를 랜덤하게 출력함
  - 사용자가 단어를 따라 입력함
  - 출제된 단어와 입력한 단어가 일치하면 "통과!", 아니면 "오타! 다시 도전"
    출력횟수는 총 10번이고, 끝나면 "게임을 종료합니다." 출력

```
영어 타자 게임, 준비되면 엔터>

문제 1
monkey
monkey
통과!

문제 2
deer
deer
통과!

문제 3
deer
der
오타! 다시 도전!
```

```
문제 8
horse
horse
통과!

문제 9
tiger
tiger
통과!

문제 10
fox
fox
통과!
게임 소요 시간: 20.9초
```

실습 – 영어 타이핑 게임

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

int main()
{
    char* words[] = { "ant", "bear", "chicken", "deer", "elephant", "fox",
            "horse", "monkey", "lion", "tiger"};
    char* question;   //문제
    char answer[20];  //사용자
    int n = 1;  //문제 번호
    clock_t start, end;
    double elapsedTime; //게임 소요 시간

    srand(time(NULL)); //seed 설정
    int size = sizeof(words) / sizeof(words[0]);

    printf("영어 타자 게임, 준비되면 엔터>");
    getchar();

    start = clock();  //시작 시간
```

실습 – 영어 타이핑 게임

```
while (n <= 10)
{
    printf("\n문제 %d\n", n);
    int rndIdx = rand() % size;
    question = words[rndIdx];

    printf("%s\n", question);  //문제 출제
    scanf_s("%s", answer, sizeof(answer));  //사용자 입력

    if (strcmp(question, answer) == 0)
    {
        printf("통과!\n");
        n++;   //다음 문제
    }
    else
    {
        printf("오타! 다시 도전!\n");
    }
}
end = clock(); //종료 시간
elapsedTime = (double)(end - start) / CLOCKS_PER_SEC;
printf("게임 소요 시간: %.1f초\n", elapsedTime);
```

동적 할당

◈ 동적 할당의 필요성

배열의 크기는 프로그램이 컴파일되는 과정에서 결정된다. 이를 "정적할당 " 이라 한다.

정적할당을 사용하면 간혹 메모리가 낭비되는 경우가 발생한다.

만약 프로그램이 실행되는 동안 필요한 크기만큼의 배열을 생성할 수 있다면 메모리의 낭비를 줄일 수 있다.

- 정수형 값 10개를 저장하기 위한 1차원 배열 생성(동적 할당)

```
int* array = (int *)malloc(sizeof(int) * 10);
```

동적으로 할당되는 배열의 메모리 주소는 프로그램이 실행되기 이전에는 알 수 없는 값이므로 이 주소를 저장하기 위해 포인터가 필요하다.

★ **malloc(memory allocation) 함수**

필요한 메모리의 크기를 바이트 단위로 할당할 수 있도록 해준다. malloc 함수는 할당된 메모리 주소를 반환한다. 반환되는 메모리 번지에 어떤 유형의 데이터를 저장할 것인지 지정해 주어야한다.

★ **free()** 함수

동적으로 할당된 메모리를 시스템에 반납하도록 해 준다.

동적 메모리 할당

◆ 동적 할당 사용

```c
#include <stdio.h>
#include <stdlib.h>  //malloc() 사용

int main()
{
    //정수형 배열 4개 선언 - 정적할당
    /*int n[4];

    n[0] = 10;
    n[1] = 20*/

    //정수형 배열 4개 선언 - 동적할당
    int* pn = (int *)malloc(sizeof(int) * 4);

    /*pn[0] = 10;
    pn[1] = 20;
    printf("%d %d\n", pn[0], pn[1]);*/
```

동적 메모리 할당

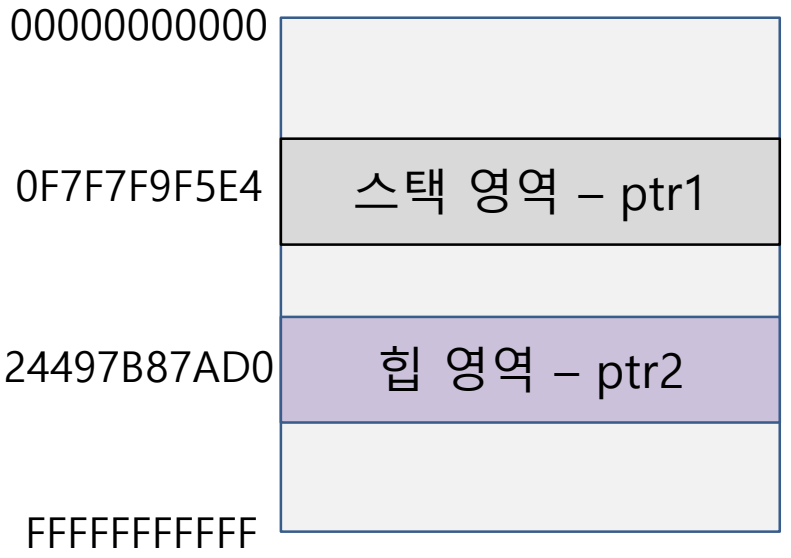
◆ 동적 할당 사용

```
    // 2의 배수로 저장
    for (int i = 0; i < 4; i++)
    {
        pn[i] = i * 2;
    }

    //출력
    for (int i = 0; i < 4; i++)
    {
        printf("%d\n", pn[i]);
    }

    free(pn);  //메모리 해제

    return 0;
}
```

동적 메모리 할당

◆ 동적 할당 사용

```
int num1 = 10;
int* ptr1;
int* ptr2;

ptr1 = &num1;  //정적 할당(메모리 스택영역)
//동적 할당(힙 영역)
ptr2 = (int*)malloc(sizeof(int) * 3);
if (ptr2 == NULL)
{
    printf("동적 메모리 할당에 실패했습니다.\n");
    exit(1); //강제 종료
}
ptr2[0] = 11;
ptr2[1] = 12;
printf("%d %x\n", *ptr1, ptr1);
printf("%d %x\n", *ptr2, ptr2);  //*(ptr2 + 0)

free(ptr2);  //메모리 해제
```



동적 메모리 할당 예제

◈ 정수형 배열 4개 동적 할당

```
int* ip;
int i;

ip = (int *)malloc(sizeof(int) * 4);

if (ip == NULL)
{
    printf("동적 메모리 할당에 실패했습니다.\n");
    exit(1);
}
//배열로 저장하기
ip[0] = 10;
ip[1] = 20;
ip[2] = 30;
ip[3] = 40;

for (i = 0; i < 4; i++)
{
    printf("%d\n", ip[i]);
}
```

동적 메모리 할당 예제

◈ 정수형 배열 4개 동적 할당

```
//역참조로 출력
for (i = 0; i < 4; i++)
{
    printf("%d\n", *(ip + i));
}
printf("\n");

//값 변경
*ip = 50;
*(ip + 1) = 60;
*(ip + 2) = 70;
*(ip + 3) = 80;

for (i = 0; i < 4; i++)
{
    printf("%d %d\n", ip[i], *(ip + i));
}

free(ip); //메모리 해제
```

```
10
20
30
40

50 50
60 60
70 70
80 80
```

동적 메모리 할당 예제

◈ 알파벳을 저장할 문자형 배열 26개 동적 할당

```
char* pc;
pc = (char *)malloc(sizeof(char) * 26);
int i;

if (pc == NULL)
{
    printf("동적 메모리 할당에 실패했습니다.\n");
    exit(1); //강제 종료
}
/*
    //*pc = 'A';
    *(pc + 0) = 'A';
    printf("%c\n", *(pc + 0));

    *(pc + 1) = 'A' + 1;
    printf("%c\n", *(pc + 1));

    *(pc + 2) = 'A' + 2;
    printf("%c\n", *(pc + 2));
*/
```

동적 메모리 할당 예제

◈ 알파벳을 저장할 문자형 배열 26개 동적 할당

```
//저장
for (i = 0; i < 26; i++)
{
    *(pc + i) = 'A' + i;
}

//출력
for (i = 0; i < 26; i++)
{
    printf("%c ", *(pc + i));
}

free(pc); //메모리 해제
```

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

동적 메모리 할당 예제

◈ 2차원 포인터 배열

```
//2차원 포인터 배열 동적 할당
int** pp;  //정수형 포인터의 포인터

pp = (int**)malloc(sizeof(int*) * 2); //8B x 2 = 16B
/*pp[0] = (int*)malloc(sizeof(int) * 2);
pp[1] = (int*)malloc(sizeof(int) * 2);*/

for (int i = 0; i < 2; i++)
{
    pp[i] = (int*)malloc(sizeof(int) * 2); //정수형 포인터
}

//값 저장
pp[0][0] = 1;
pp[0][1] = 2;
pp[1][0] = 3;
pp[1][1] = 4;
```

동적 메모리 할당 예제

◈ 2차원 포인터 배열

```
//값 출력
for (int i = 0; i < 2; i++)
{
    for (int j = 0; j < 2; j++)
    {
        printf("%d\n", pp[i][j]);
    }
}

//메모리 해제
/*free(pp[1]);
free(pp[0]);*/
for (int i = 0; i < 2; i++)
{
    free(pp[i]);
}
```

실습 문제 – 포인터

빈 칸에 알맞은 코드를 입력하여 실행 결과대로 출력하시오.

```
char arr[] = { '1', 'B', 'C', 'D', 'E' };
char* p;

p = [          ]

printf("%c%c", [          ]);
```

☞ 실행 결과

```
C1
```