C - 포인터(pointer)

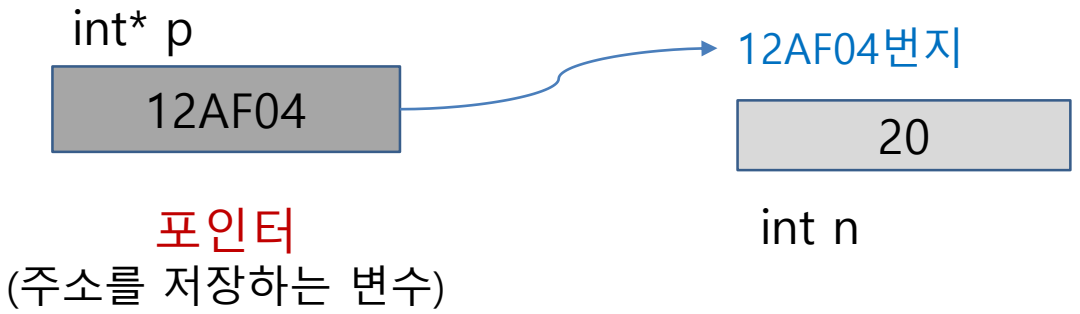
pointer

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

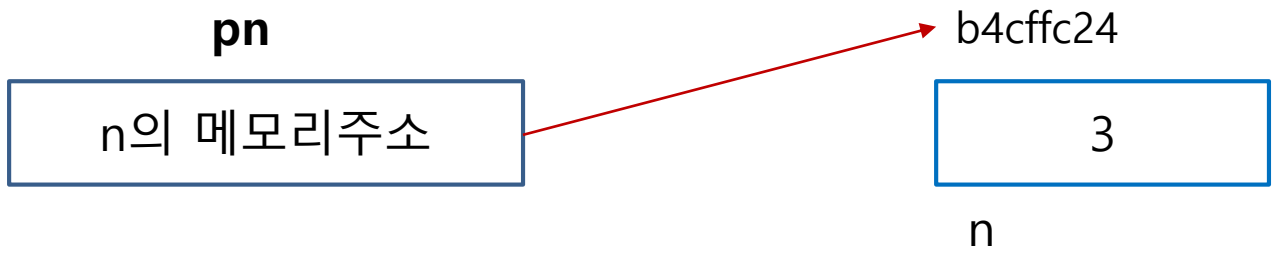
```
int n;
int* pn;

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)

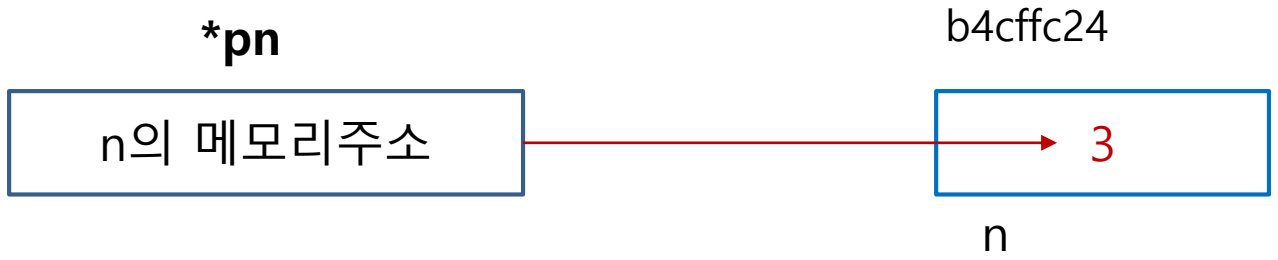
- 역참조 연산자(*)

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);
```

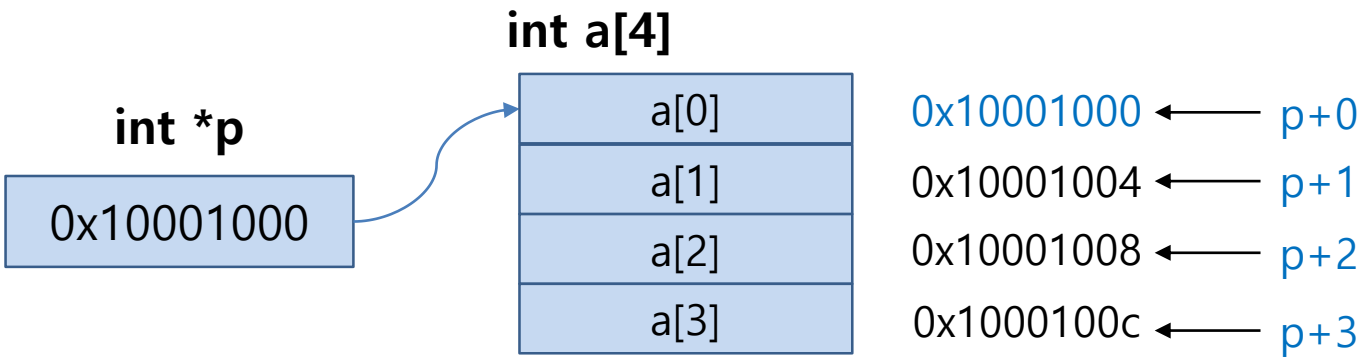
```
a의 값은 10
a의 값은 20
c1의 값은 A
c1의 값은 B
```

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[0]           // p에 배열의 첫 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 포인터

```c
int a[4] = { 10, 20, 30, 40 };

printf("배열 a[3]의 값은 %d\n", a[3]);
printf("배열 a[3]의 값은 %d\n", *(a + 3));
printf("배열 a[3]의 값은 %d\n", *a + 3);
```

```
배열 a[3]의 값은 40
배열 a[3]의 값은 40
배열 a[3]의 값은 13
```

배열과 포인터(Pointer)

- 포인터 배열의 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
```

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]);
```

문자열과 포인터(Pointer)

- 2차원 포인터 배열

```c
// 정적 배열 3개 준비 (행)
int row1[] = { 1, 2, 3 };
int row2[] = { 4, 5, 6 };
int row3[] = { 7, 8, 9 };

// 포인터 배열에 각 행의 시작 주소 저장
int* ptrArr[3];  //정수형 타입의 포인터 배열 생성
ptrArr[0] = row1;
ptrArr[1] = row2;
ptrArr[2] = row3;

// 출력
for (int i = 0; i < 3; i++) {           // 행
    for (int j = 0; j < 3; j++) {       // 열
        printf("%d ", ptrArr[i][j]);    // 포인터를 통한 2차원 접근
    }
    printf("\n");
}
```

```
1 2 3
4 5 6
7 8 9
```

문자열과 포인터(Pointer)

- 문자열 반환 함수

```c
//문자열 반환 함수
char* message() {
    return "행운을 빌어요!";
}

int main()
{
    char msg[50];

    strcpy(msg, message()); //문자열 호출후 msg에 저장

    printf("%s\n", msg);

    return 0;
}
```

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

```
void callByVal(int);
void 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 ===
11
10
=== 참조에 의한 호출 ===
11
11
```

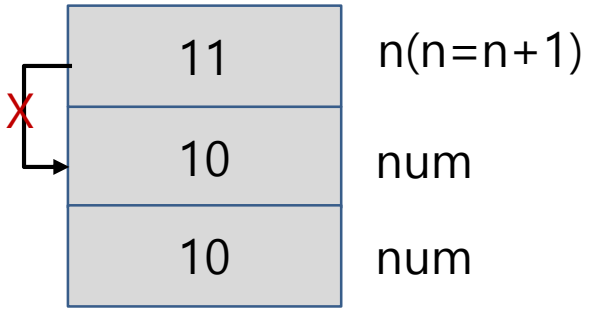
값 & 참조에 의한 호출

- 값 & 참조에 의한 호출

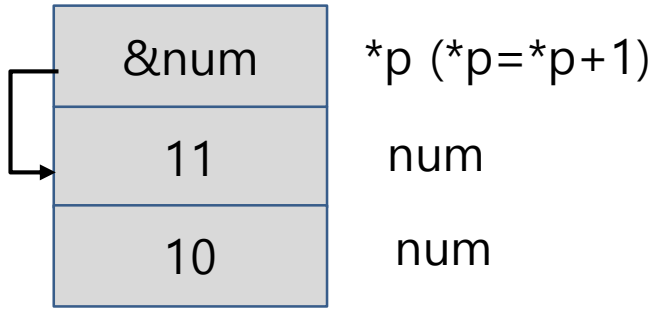
```
//값 호출 함수 정의
void callByVal(int n)
{
    n++; //n = n + 1;
    printf("%d\n", n);
}

//참조 호출 함수 정의
void callByRef(int* p)
{
    *p += 1; //*p = *p + 1;
    printf("%d\n", *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
```

값 & 참조에 의한 호출

- x부터 y 까지의 합계 계산하는 프로그램

```c
int calc(int x, int y);
int main()
{
    int value;
    value = calc(1, 10);

    printf("합계: %d\n", value);
    return 0;
}

int calc(int x, int y){
    int i, sum = 0;

    for (i = x; i <= y; i++){
        sum += i;
    }
    return sum;
}
```

참조에 의한 호출

- x부터 y 까지의 합계 계산하는 프로그램

```
void swap(int*, int*);
void calcSum(int, int);

int main()
{
    calcSum(1, 5);
    calcSum(5, 1);
    calcSum(4, 4);

    return 0;
}
```

```
1+2+3+4+5=15
1에서 5까지의 합은 15
1+2+3+4+5=15
1에서 5까지의 합은 15
4=4
4에서 4까지의 합은 4
```

참조에 의한 호출

- x부터 y 까지의 합계 계산하는 프로그램

```
void calcSum(int min, int max)
{
    int i, sum;

    //첫번째 인자가 두번째 인자보다 크면
    if (min > max)
        swap(&min, &max);

    printf("%d", min); //1
    sum = min;
    for (i = min + 1; i <= max; i++) {
        printf("+%d", i); //1+2+3+4+5
        sum += i;
    }
    printf("=%d\n", sum);
    printf("%d에서 %d까지의 합은 %d\n", min, max, sum);
}
```

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

```
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;
}
```

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;
}
```

영어 타이핑 게임1

- **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
```

영어 타이핑 게임 – 포인터 사용

- 영어 타자 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
```

영어 타이핑 게임 – 포인터 사용

- 영어 타자 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);
```

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
```

동적 할당

- 동적 할당의 필요성

변수나 배열을 선언하면 메모리 상에 '자동으로' 영역이 확보된다. 그러나 이 방법으로 동영상 관련 프로그램 등 많은 양의 메모리를 준비해야 할 경우 최악의 상황에선 프로그램이 정지할 수도 있다.
이런 경우 '프로그램상의 처리 ' 로써 메모리를 확보한다.

```
int* array = (int *)malloc(sizeof(int) * 100);
```

① 포인터를 준비한다
② 메모리를 확보하고 그 시작주소를 준비해 둔 포인터에 저장한다.
③ 필요 없어지면 메모리를 해제한다.

동적 할당

- 동적 메모리 사용

malloc() 함수

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

```
/*//배열 - 정적 할당(메모리 - 스택 영역)
int n[5];

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

//배열 - 동적 할당(메모리 - 힙 영역)
int* pn;
int i;

pn = (int *)malloc(sizeof(int) * 5);
if (pn == NULL)
{
    printf("동적 메모리 할당에 실패했습니다.\n");
    exit(1); //강제 종료
}
```

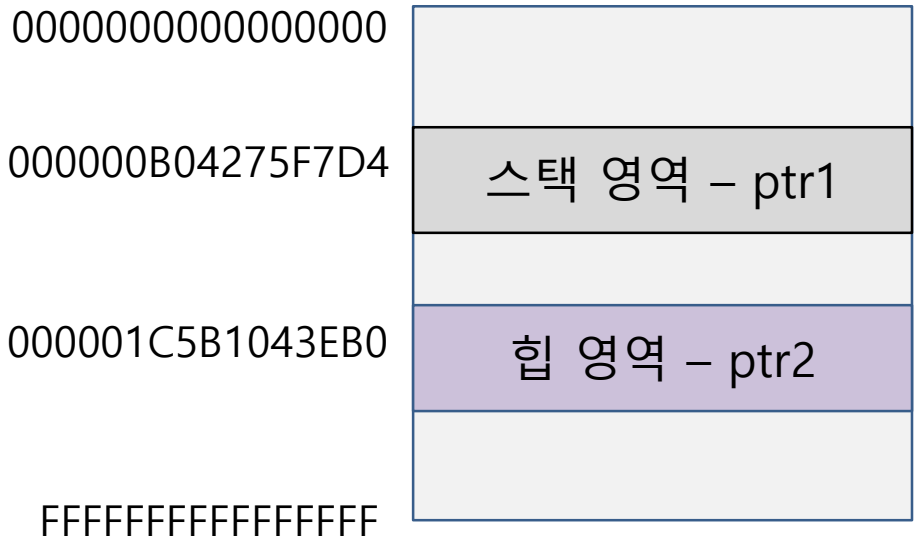
동적 메모리 할당

- 동적 할당 사용

```
    /*pn[0] = 2;
    pn[1] = 4;*/

    //2의 배수 저장
    for (int i = 0; i < 5; i++){
        pn[i] = (i + 1) * 2;
    }

    for (int i = 0; i < 5; i++){
        printf("%d ", pn[i]); //2 4 6 8 10
    }

    free(pn); //메모리 반납

    return 0;
}
```

동적 메모리 할당

- 동적 할당 사용



동적 메모리 할당

- 스택 영역과 힙 영역

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

//정적 할당(메모리 스택영역)
ptr1 = &num1;

//동적 할당(메모리 힙영역)
ptr2 = (int* )malloc(sizeof(int) * 3);
if (ptr2 == NULL) {
    printf("동적 메모리 할당에 실패했습니다.\n");
    exit(1);
}
```

동적 메모리 할당

- 스택 영역과 힙 영역

```
//값 저장
ptr2[0] = 11;
ptr2[1] = 12;
ptr2[2] = 13;

//ptr1의 값과 주소
printf("%d %p\n", *ptr1, ptr1);
printf("\n");

//ptr2의 값과 주소
for (int i = 0; i < 3; i++) {
    printf("%d %p\n", *(ptr2 + i), ptr2 + i);
}

free(ptr2);  //메모리 해제
```

```
10 000000B6D9CFFAD4

11 00000208724712B0
12 00000208724712B4
13 00000208724712B8
```

동적 메모리 할당

- 동적 할당 배열의 연산

```c
int len;
int* p;
int i, sum = 0;
float avg;

printf("*** 점수의 평균 계산 ***\n");
printf("입력할 정수의 개수: ");
scanf("%d", &len);

p = (int*)malloc(sizeof(int) * len); //동적 할당

//점수 입력
for (i = 0; i < len; i++) {
    printf("%d번째 점수: ", i + 1);
    scanf("%d", &p[i]);
}
```

동적 메모리 할당

- 동적 할당 배열의 연산

```
//합계 계산
for (i = 0; i < len; i++) {
    sum += p[i];
}
//printf("%d\n", sum); //합계

//평균 계산
avg = (float)sum / len;
printf("평균 점수: %.1f\n", avg);

free(p);  //메모리 반납
```

```
*** 점수의 평균 계산 ***
입력할 정수의 개수: 3
1번째 점수: 75
2번째 점수: 80
3번째 점수: 92
평균 점수: 82.3
```

동적 메모리 할당 예제

- 공백 문자 제거

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

void delBlank(char a[]);
int main()
{
    char arr[] = "A B c D e F !";

    delBlank(arr); //공백문자 제거 함수 호출

    printf("%s\n", arr);

    return 0;
}
```

동적 메모리 할당 예제

- 공백 문자 제거

```
void delBlank(char a[])
{
    int len = strlen(a);
    char* str;  //포인터(동적 할당)
    int i, k = 0;

    str = (char*)malloc(sizeof(char) * len);
    if (str == NULL) {
        printf("동적 메모리 할당에 실패함\n");
        return 1;
    }
```

```
ABcDeF!
```

동적 메모리 할당 예제

- 공백 문자 제거

```
//공백문자 제거후 저장
for (i = 0; i < len; i++)
{
    if (a[i] != ' ') //a[i] 공백문자가 아니면
        str[k++] = a[i];
}
str[k] = '\0'; //문자열 맨뒤 널문자

//strcpy(a, str);  //str을 a에 복사
strcpy_s(a, sizeof(a), str);

free(str);
```

성적 관리

- **성적 관리 프로그램**

1. 학생수 – 동적 할당(포인터)
2. 점수 입력 – 학생수 만큼 점수를 입력함
3. 점수 리스트 – 점수 리스트 출력
4. 분석 – 평균, 최고 점수 계산 및 출력
5. 종료 – 프로그램 종료

※ 학생수를 입력하지 않고 2, 3, 4, 메뉴를 선택하면
"학생수를 먼저 입력하세요" 메시지 출력

성적 관리

- 기능별 출력 화면

```
-------------------------------------------------------
1.학생수 | 2.점수입력 | 3.점수리스트 | 4.분석 | 5.종료
-------------------------------------------------------
선택> 1
학생수: 3
-------------------------------------------------------
1.학생수 | 2.점수입력 | 3.점수리스트 | 4.분석 | 5.종료
-------------------------------------------------------
선택> 2
score[0]=70
score[1]=85
score[2]=90
-------------------------------------------------------
1.학생수 | 2.점수입력 | 3.점수리스트 | 4.분석 | 5.종료
-------------------------------------------------------
선택> 3
score[0]=70
score[1]=85
score[2]=90
-------------------------------------------------------
1.학생수 | 2.점수입력 | 3.점수리스트 | 4.분석 | 5.종료
-------------------------------------------------------
선택> 4
평균 점수: 81.67
최고 점수: 90
-------------------------------------------------------
1.학생수 | 2.점수입력 | 3.점수리스트 | 4.분석 | 5.종료
-------------------------------------------------------
선택> 5
프로그램 종료!
```

성적 관리

- **성적 관리 프로그램**

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h> //true/false 사용

int main()
{
    bool run = true;   //토글(상태) 변수 - 실행/종료
    int studentNum, choice;  //학생수, 메뉴 선택
    int* score = NULL;  //점수 (동적 배열)
    int sum, max;       //합계, 최고점수
    float avg;          //평균

    while (run) {
        printf("-----------------------------------------------------\n");
        printf("1.학생수 | 2.점수입력 | 3.점수리스트 | 4.분석 | 5.종료\n");
        printf("-----------------------------------------------------\n");
        printf("선택> ");
        scanf("%d", &choice); //메뉴 입력
```

성적 관리

- **성적 관리 프로그램**

```
switch (choice) {
case 1:
    printf("학생수 입력: ");
    scanf_s("%d", &studentNum);
    score = (int*)malloc(sizeof(int) * studentNum);
    if (score == NULL) {
        printf("동적 메모리 할당에 실패했습니다.\n");
        return 1;
    }
    break;
case 2:
    for (int i = 0; i < studentNum; i++) { //점수 입력
        printf("score[%d]=", i);
        scanf("%d", &score[i]);
    }
    break;
case 3:
    for (int i = 0; i < studentNum; i++) { //점수 리스트
        printf("score[%d]=%d\n", i, score[i]);
    }
    break;
```

성적 관리

- **성적 관리 프로그램**

```
    case 4:
        sum = 0; //합계 초기화
        max = score[0]; //최대값 설정
        for (int i = 0; i < studentNum; i++) {
            sum += score[i];  //합계
            if (score[i] > max)
                max = score[i]; //최대값
        }
        avg = (float)sum / studentNum;
        printf("평균 점수: %.1f\n", avg);
        printf("최고 점수: %d\n", max);
        break;
    case 5:
        printf("프로그램 종료!");
        run = false;  //실행 종료
        break;
    default:
        printf("잘못된 선택입니다. 다시 선택하세요\n");
        break;
    }
}
free(score); //메모리 반납
```

성적 관리 – 함수 사용

- **전역변수, 학생수**

```
//전역 변수 선언
bool run = true;  //상태 변수
int studentNum, choice; //학생수, 메뉴 선택
int* score;   //점수 (포인터)
int sum, max; //합계, 최고점수
float avg;    //평균

void getStudentNum() { //학생수
    if (score != NULL) {
        free(score); //메모리 해제
        score = NULL;
    }

    printf("학생수: ");
    scanf("%d", &studentNum);
    score = (int*)malloc(sizeof(int) * studentNum); //동적 생성
    if (score == NULL) {
        printf("동적 메모리 할당에 실패했습니다.\n");
        return 1;
    }
}
```

성적 관리 – 함수 사용

- 점수 입력 / 점수 리스트

```
void inputScore() { //점수 입력
    if (score == NULL) {
        puts("학생수를 먼저 입력하세요.");
        return;
    }
    for (int i = 0; i < studentNum; i++) {
        printf("score[%d]=", i);
        scanf("%d", &score[i]);
    }
}

void getScoreList() { //점수 리스트
    if (score == NULL) {
        puts("학생수를 먼저 입력하세요.");
        return;
    }
    for (int i = 0; i < studentNum; i++) {
        printf("score[%d]=%d\n", i, score[i]);
    }
}
```

성적 관리 – 함수 사용

- 점수 분석

```
void calculate() { //통계 분석
    if (score == NULL) {
        puts("학생수를 먼저 입력하세요.");
        return;
    }

    sum = 0; //합계 초기화
    max = score[0]; //최대값 설정
    for (int i = 0; i < studentNum; i++) {
        sum += score[i];  //합계
        if (score[i] > max)
            max = score[i]; //최대값
    }
    avg = (float)sum / studentNum;
    printf("평균 점수: %.2f\n", avg);
    printf("최고 점수: %d\n", max);
}
```

성적 관리 – 함수 사용

- **main() 함수**

```
int main()
{
    while (run) {
        printf("-----------------------------------------------------\n");
        printf("1.학생수 | 2.점수입력 | 3.점수리스트 | 4.분석 | 5.종료\n");
        printf("-----------------------------------------------------\n");
        printf("선택> ");

        scanf_s("%d", &choice); //메뉴 입력
        switch (choice) {
        case 1:
            getStudentNum();
            break;
        case 2:
            inputScore();
            break;
```

성적 관리 – 함수 사용

- **main() 함수**

```
            case 3:
                getScoreList();
                break;
            case 4:
                calculate();
                break;
            case 5:
                printf("프로그램 종료!");
                run = false;
                break;
            default:
                printf("잘못된 선택입니다. 다시 선택하세요\n");
                break;
            }
        }

        return 0;
}
```

영어 타이핑 게임 2

- **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
```

영어 타자 게임

```c
#define MAX_WORDS 10    //단어의 개수
#define MAX_LENGTH 20   //단어의 길이
int main()
{
    // 1. 단어 분리 및 저장
    char words[] = "ant bear chicken cow dog elephant monkey lion tiger horse";
    char* wordList[MAX_WORDS]; // 분리된 단어 저장용 배열
    int idxOfWords = 0;

    char* ptr = strtok(words, " "); //공백을 구분기호로 문자열 자르기
    while (ptr != NULL && idxOfWords < MAX_WORDS){
        wordList[idxOfWords++] = ptr;
        ptr = strtok(NULL, " ");
    }
    // 2. 타자 게임 준비
    char* question;
    char* answer = (char*)malloc(MAX_LENGTH * sizeof(char)); //동적 배열로 할당
    int n = 1;
    clock_t start, end;
    double elapsedTime;
    srand(time(NULL));
```

영어 타자 게임

```
printf("영어 타자 게임, 준비되면 엔터> ");
getchar();
start = clock();
while (n <= 10){
    printf("\n문제 %d\n", n);
    int rndIdx = rand() % idxOfWords; //실제 단어 개수 사용
    question = wordList[rndIdx];  //랜덤한 단어 추출
    printf("%s\n", question); //문제 출제
    scanf("%s", answer);  //사용자 입력
    if (strcmp(question, answer) == 0){
        printf("통과!\n");
        n++;
    }else{
        printf("오타! 다시 도전!\n");
    }
}
end = clock();
elapsedTime = (double)(end - start) / CLOCKS_PER_SEC;
printf("게임 소요 시간: %.2f초\n", elapsedTime);
free(answer); // 메모리 해제
```

이중 포인터

◈ 이중 포인터 사용

```
int** pp;  //이중 포인터 선언
int i, j;
int k = 0;

pp = (int **)malloc(sizeof(int*) * 2); //16byte
if (pp == NULL){
    printf("동적 메모리 할당에 실패했습니다.\n");
    exit(1);
}

//동적 할당
for (i = 0; i < 2; i++){
    pp[i] = (int*)malloc(sizeof(int) * 2);
    if (pp[i] == NULL){
        printf("행 %d의 메모리 할당 실패\n", i);
        exit(1);
    }
}
```

이중 포인터

◈ 이중 포인터 사용

```
//값 저장
for (i = 0; i < 2; i++){
    for (j = 0; j < 2; j++){
        k++;
        pp[i][j] = k;
    }
}

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

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

```
1 2
3 4
```