2023/06/17

# APPDONG
# C언어 멘토링

보강 주차
멘토 : 김민수

# Contents

# 함수와 포인터의 활용

함수의 인자 전달 방식

값에 의한 호출 (Call by Value)
-> 함수 호출 시 실인자 값이 형식인자에 복사되어 저장된다

참조에 의한 호출 (Call by Reference)
-> 함수로 전달된 실인자의 주소를 이용하여 변수를 참조한다

# 함수와 포인터의 활용

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

void increase_CallByValue(int origin, int increment);
void increase_CallByReference(int* origin, int increment);

int main() {
    int amount = 10;

    increase_CallByValue(amount, 20);
    printf("Call by Value : %d\n", amount);

    increase_CallByReference(&amount, 20);
    printf("Call by Reference : %d", amount);

    return 0;
}
```

# 함수와 포인터의 활용

```
19
20  void increase_CallByValue(int origin, int increment) {
21      origin += increment;
22  }
23
24  void increase_CallByReference(int* origin, int increment) {
25      *origin += increment;
26  }
```

Call by Value의 경우 함수 내부의 변수 값은 함수 외부에 영향을 끼치지 못한다

반면에 Call by Reference의 경우 해당 변수의 주소를 참조하기 때문에 값이 변경되면 해당 변수의 값도 변경된다

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

void swap1(int x, int y);
void swap2(int* x, int* y);

int main() {
    int a = 100, b = 200;

    printf("a= %d, b= %d\n\n", a, b);

    // Call by Value
    swap1(a, b);
    printf("After swap1() a= %d, b= %d\n\n", a, b);

    // Call by Reference
    swap2(&a, &b);
    printf("After swap2() a= %d, b= %d\n\n", a, b);

    return 0;
}
```

```
23
24  void swap1(int x, int y) {
25      int temp;
26
27      temp = x;
28      x = y;
29      y = temp;
30
31      printf("swap1() a= %d, b= %d\n", x, y);
32  }
33
34  void swap2(int* x, int* y) {
35      int temp;
36
37      temp = *x;
38      *x = *y;
39      *y = temp;
40
41      printf("swap2() a= %d, b= %d\n", *x, *y);
42  }
```

함수 내에서만 x값, y값이 바뀌지
실제 a값과 b값에 영향을 줄 수 없다

a와 b의 주소를 참조하는 값이 변경되면
a값과 b값도 영향을 받는다

# 함수와 포인터의 활용

배열을 함수의 인자로 전달하는 방법

```
double sum(double arr[], int n);
...
double data[] = {2.1, 3.3, 5.5};
sum(data, 3);
```

-> 배열이름으로 배열인자를 명시한다

배열을 함수 인자로 전달할 때 배열 크기도 함께 전달하는 것이 효율적이다
-> 전달하지 않으면 정해진 상수를 함수 내부에서 사용해야하는 데
배열 크기가 변경될 경우 코드를 수정해야하므로 비효율적이다

# 함수와 포인터의 활용

sum(data, size);  -> 배열 이름으로 함수 인자를 전달한다
+ 배열 크기도 함께 전달한다

배열을 인자로 가지는 함수 선언 방법

double sum(double data[], int n); 또는
double sum(double *data, int n); 으로 함수를 선언하면 된다

# 함수와 포인터의 활용

배열 크기 구하는 방법

sizeof(배열명) / sizeof(배열의 자료형 또는 배열원소)

ex)

```
int arr[] = {1.2, 3.3, 5.5, 7.7, … };
int arr_size1 = sizeof(arr) / sizeof(int);
int arr_size2 = sizeof(arr) / sizeof(arr[0]);
```

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

double sum(double ary[], int n);

int main() {
    double data[] = { 2.3, 3.4, 4.5, 5.6, 7.8 };
    int size = 0;

    size = sizeof(data) / sizeof(data[0]);

    printf("Total= %lf\n", sum(data, size));

    return 0;
}
```

배열 크기를 계산해서

배열 이름과 배열 크기를 함수 인자로 전달

```
17
18  double sum(double ary[], int n) {
19      int i = 0;
20      double total = 0.0;
21
22      for (i = 0; i < n; i++) {
23          total += ary[i];
24      }
25
26      return total;
27  }
```

total += *(ary + i); 로 작성 가능

# 함수와 포인터의 활용

배열을 전달할 때, 함수에서는 인자를 배열 대신 포인터로 받아도 된다

ex)

```
int sum(int arr[], int size);    <->    int sum(int *arr, int size);
...                                     ...
sum += arr[i];                          sum += *(arr + i);
or                                      or
sum += *arr++;                          sum += *(arr++);
```

# 함수와 포인터의 활용

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

void input_arr(double* pa, int size);
double find_max(double* pa, int size);

int main() {
    double arr[5];
    double max;

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

    input_arr(arr, size);
    max = find_max(arr, size);

    printf("배열의 최댓값 : %lf", max);

    return 0;
}
```

# 함수와 포인터의 활용

```
21
22  void input_arr(double* pa, int size) {
23      int i;
24
25      printf("%d개의 실수값 입력 : ", size);
26      for (i = 0; i < size; i++) {
27          scanf("%lf", pa + i);
28      }
29  }
30
```

```
31  double find_max(double arr[], int size) {
32      double max;
33      int i;
34
35      max = arr[0];
36      for (i = 1; i < size; i++) {
37          if (arr[i] > max) max = arr[i];
38      }
39
40      return max;
41  }
```

# 함수와 포인터의 활용

포인터를 반환하는 함수

int* add(int *, int, int);
-> 반환값 : 포인터(int* )
-> 호출 예시 : *add(&sum, m, n);

반환되는 값(주소값)을 다시 간접 참조하여 저장된 데이터 값 접근 가능

단, 반환값은 함수 내부의 지역변수의 주소값이면 안됨.
함수가 종료되는 시점에 메모리에서 제거 되는 변수이기 때문에 문제가 발생

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

int* add(int*, int, int);

int main() {
    int m = 0, n = 0, sum = 0;

    scanf("%d %d", &m, &n);

    printf("두 정수의 합 : %d\n", *add(&sum, m, n));

    return 0;
}

int* add(int* psum, int a, int b) {
    *psum = a + b;

    return psum;
}
```

함수 반환 값을 참조해서
정수의 합 출력

포인터를 반환(주소 값을 반환)

# 문자열

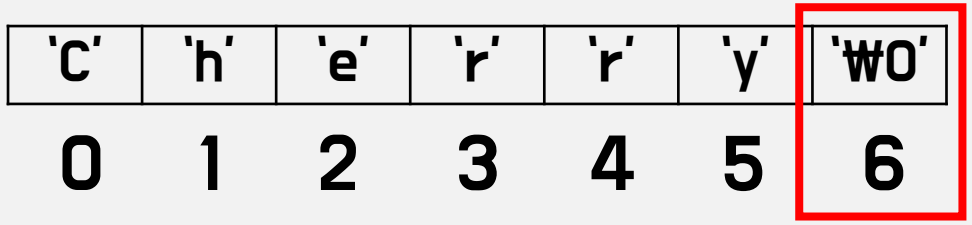
'C' -> 문자 "Cherry" -> 문자열

```
char c = 'C';
char str[7] = "Cherry";
```

C언어에서는 character 배열을 사용해서 문자열을 나타낸다.
문자열이 저장될 때 실제 문자열의 길이보다 1이 더 큰 공간을 차지한다.
-> 이유는?

# 문자열

char str[7] = "Cherry";




문자열 끝에는 항상 '₩0' (종료 문자, null)이 존재한다.
문자열을 출력 -> null이 나올 때까지 출력

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

int main() {
    char str[7] = "Cherry";

    printf("%s", str);

    return 0;
}
```

문자열을 출력할 때는 %s 사용

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

int main() {
    char word[7] = "Cherry";
    char* str = word;

    printf("%s", str);

    return 0;
}
```

포인터를 사용해서 출력 가능

word는 word[]의 맨 앞의 주소
이를 str로 참조해서 출력

# 문자열

문자열을 표현하는 방법

1. 배열 표현
char str[7] = "Cherry";

2. 포인터 표현
char* str = "Cherry";

-> 차이점은?

배열 표현의 경우 str[0] = 'A'; 처럼 각 자리의 문자를 변경할 수 있음.
그러나, 포인터 표현의 경우 값을 변경할 수 없음.

# 문자열

## 문자열 길이를 구하는 법

### 1. 소스 코드를 직접 작성

```
int str_length(char *str) {
    int i = 0;
    while(str[i]) {
        i++;
    }

    return i;
}
```

### 2. strlen() 함수 사용

#include <string.h> 헤더파일 포함.

```
char *str = “hello”;

printf(“%d”, strlen(str));
```

```c
#include <stdio.h>
#include <string.h>

int str_length(char* s);

int main() {
    char* str = "Hello, World!";

    printf("%d₩n", strlen(str));
    printf("%d", str_length(str));

    return 0;
}

int str_length(char* s) {
    int i = 0;

    while (s[i]) {
        i++;
    }

    return i;
}
```

strlen(): 함수 정도는 기억하는 게 좋음

# 문자열

문자열을 입력 받는 법

```
char str[20];

scanf("%s", str);
```

scanf(); 를 사용해서 간단하게 입력 받을 수 있다.
단, scanf("%s", str);는 공백문자들을 모두 무시하기 때문에 한 단어가 아닌 긴 문자열을 입력 받을 때 사용할 수 없다.

# 문자열을 입력 받는 법 3가지 - 첫번째. scanf("%[^₩n]s", str);

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

int main() {
    char str[20];

    scanf("%[^₩n]s", str);

    printf("%s", str);

    return 0;
}
```

[^₩n] 서식 지정자를 사용해서
공백을 포함하여 입력

# 문자열을 입력 받는 법 3가지 - 두번째. gets(str, sizeof(str));

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

int main() {
    char str[20];

    gets(str, sizeof(str));

    printf("%s", str);

    return 0;
}
```

안정성의 이유로 현재 gets();는 사용 권장 안함.

# 문자열을 입력 받는 법 3가지 - 세번째. fgets(str, sizeof(str), stdin);

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

int main() {
    char str[20];

    fgets(str, sizeof(str), stdin);

    printf("%s", str);

    return 0;
}
```

gets();를 대체하여 사용.
stdin은 표준 입력이라는 뜻.

# 문자열

문자열을 복사하는 함수 strcpy():

#include <string.h> 헤더 파일 포함.
strcpy(s2, s1); -> s1의 문자열을 s2로 복사

```
char s1[10] = “Hello”;
char s2[10];

strcpy(s2, s1);
```

```
char* s1 = “Hello”;
char* s2;

strcpy(s2, s1);
```

-> 과연 결과는?

# 문자열

문자열을 합치는 함수 strcat();

#include <string.h> 헤더 파일 포함.
strcat(s2, s1); -> s2 뒤에 s1을 붙임.

```
char s1[10] = “world”;
char s2[20] = “hello”;

strcpy(s2, s1);
```

```
char* s1 = “world”;
char* s2 = “hello”;

strcpy(s2, s1);
```

-> 과연 결과는?

# 문자열

문자열을 비교하는 함수 strcmp();

== (비교 연산자)로는 문자열을 비교할 수 없다.
문자열을 비교할 경우 strcmp();를 사용하자.
strcmp(s1, s2); -> s1과  s2가 같은지 비교 (-1, 0, 1 반환)

반환 기준
-1 : ASCII 코드 기준으로 s2가 클 때
 0 : ASCII 코드 기준으로 두 문자열이 같을 때
 1 : ASCII 코드 기준으로 s1가 클 때

```
char s1[10] = “hello”;
char s2[20] = “hello”;

int re = strcmp(s1, s2);
```

# 구조체

## 구조체

정수, 문자, 실수, 포인터, 배열 등을 묶어 하나의 자료형으로 이용하는 것

서로 관련 있는 정보들을 하나로 처리할 때 유용하게 사용할 수 있다

# 구조체

구조체 정의 방법

1. 구조체 템플릿 정의 "struct" + 구조체 이름
ex) struct lecture {  };

2. 구조체 멤버(필드) 정의 -> 구조체를 구성하는 항목
ex) struct lecture {
        char name[20];
        int credit;
        int hour;
    };

# 구조체

## 구조체 사용 방법

구조체를 정의한 후, 사용하려는 함수나 main에서
변수로 선언하면 된다.

ex) struct lecture data_structure;

# 구조체

구조체 변수 선언 후, 중괄호를 이용해서 각 멤버 값을 초기화 할 수 있다.

초기화 전의 값은 0, 0.0, ₩0 등의 값으로 초기화 되어있다.

ex) struct lecture data_structure = {"자료구조", 3, 4};

# 구조체

구조체 멤버에 접근해서 값을 처리하는 방법

접근 연산자 . 를 사용해서 멤버를 참조

ex) data_structure.hour = 6;

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <string.h>

struct account {
    char name[12];  // 계좌주 이름
    int actnum;     // 계좌번호
    double balance; // 잔고
};

```

```c
int main() {
    struct account mine = { "홍길동", 1001, 300000 };
    struct account yours;

    strcpy(yours.name, "김민수");

    yours.actnum = 1002;
    yours.balance = 500000;

    printf("%s %d %.2lf\n", mine.name, mine.actnum, mine.balance);
    printf("%s %d %.2lf\n", yours.name, yours.actnum, yours.balance);

    return 0;
}
```

# 구조체

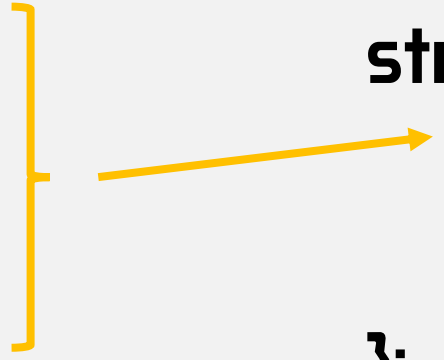
## 구조체 멤버로 사용되는 구조체

구조체 1

```
struct s1 {
    int a;
    int b;
};
```

구조체 2

```
struct s2 {
    struct s1 new_s1
    int c;
    int b;
};
```



구조체 1의 멤버를
구조체 2에서 참조할 수 있다

# 구조체

구조체 멤버로 사용되는 구조체

```
struct s2 new_s2;

new_s2.new_s1.a = 12;
new_s2.new_s1.b = 120;
```

접근 연산자를 2번 사용해서 값을 처리할 수 있다.

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <string.h>

struct date {
    int year;
    int month;
    int day;
};

struct account {
    struct date open_date;  // 계좌 개설일자
    char name[12];  // 계좌주 이름
    int actnum;     // 계좌번호
    double balance; // 잔고
};

```

```c
int main() {
    struct account me = { {2022, 12, 19} ,"김민수", 1001, 100000 };
    me.open_date.year = 2023;
    me.open_date.month = 1;
    me.open_date.day = 1;

    printf("%d %d %d\n", me.open_date.year, me.open_date.month, me.open_date.day);
    printf("%s %d %.2lf\n", me.name, me.actnum, me.balance);

    return 0;
}
```

# 구조체

## 공용체

서로 다른 자료형의 값을 동일한 저장공간에 저장하는 자료형

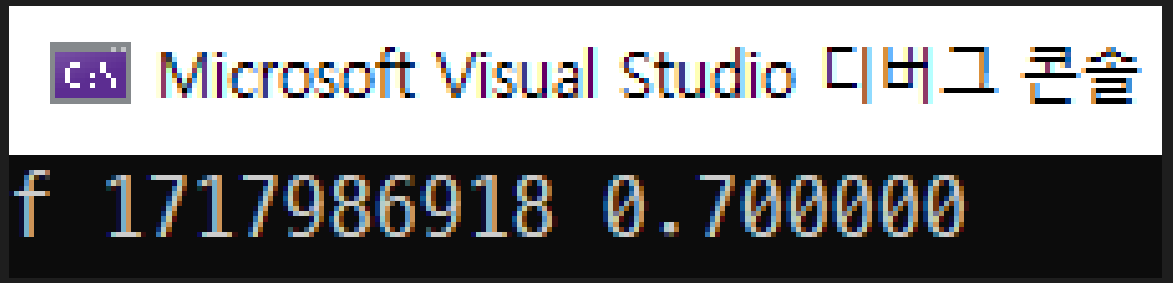
선언 방식은 struct 대신
union으로 사용하는 것을 제외하면 구조체와 동일

# 구조체

## 공용체 사용 시 주의할 점

항상 마지막에 저장한 멤버에 접근 했을 때 정확한 값을 얻을 수 있다.
그 외의 값들은 출력할 때 정확한 값을 구할 수 없음.

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

union ex1 {
    char ch;
    int i;
    double real;
};

int main() {
    union ex1 union_example;

    union_example.ch = 'A';
    union_example.i = 10;
    union_example.real = 0.7;

    printf("%c %d %lf\n", union_example.ch, union_example.i, union_example.real);

    return 0;
}
```



# 구조체

자료형 재정의 typedef

이미 사용되는 자료 유형을 다른 새로운 자료형 이름으로
재정의할 수 있도록 하는 키워드

typedef를 사용해서 구조체를 정의할 때
구조체를 새로운 자료형으로 재정의 해서 단순화할 수 있음

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <string.h>

struct date {
    int year;
    int month;
    int day;
};

typedef struct date date;

typedef struct {
    date open_date; // 계좌 개설일자
    char name[12];  // 계좌주 이름
    int actnum;     // 계좌번호
    double balance; // 잔고
} account;

```

struct date 대신 date로 자료형 사용 가능

struct account가 아닌 account로 자료형 사용

```c
int main() {
    account me = { {2022, 12, 19} ,"김민수", 1001, 100000 };
    me.open_date.year = 2023;
    me.open_date.month = 1;
    me.open_date.day = 1;

    printf("%d %d %d\n", me.open_date.year, me.open_date.month, me.open_date.day);
    printf("%s %d %.2lf\n", me.name, me.actnum, me.balance);

    return 0;
}
```

# 구조체와 포인터의 활용

구조체 포인터는 구조체의 주소값을 저장하는 변수

ex)

```
typedef struct {
    char name[20];
    int type;
    int credit;
    int hour;
} lecture;
```

```
lecture os = {"운영체제",2, 3, 3};

lecture *p = &os; // 구조체 포인터
```

-> p로 구조체 변수 os 멤버 참조 가능

# 구조체와 포인터의 활용

구조체 멤버 접근 연산자 "->"

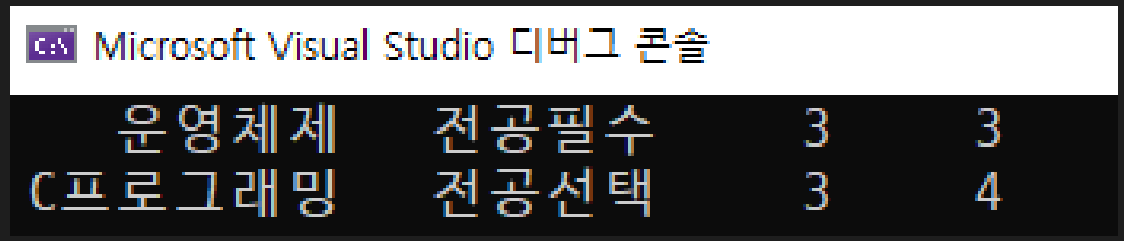
ex)

| | |
|---|---|
| p->name | os.name |
| p->type | os.type |
| p->credit | os.credit |
| p->hour | os.hour |

(*p).name도 가능

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

typedef struct {
    char name[20];
    int type;
    int credit;
    int hour;
} lecture;

char lectype[4][100] = {"교양", "일반선택", "전공필수", "전공선택"};

```

```c
int main() {
    lecture os = { "운영체제", 2, 3, 3 };
    lecture c = { "C프로그래밍", 3, 3, 4 };
    lecture* p = &os;

    printf("%12s %10s %5d %5d\n", p->name, lectype[p->type], p->credit, p->hour);
    printf("%12s %10s %5d %5d\n", c.name, lectype[c.type], c.credit, c.hour);

    return 0;
}
```



# 구조체와 포인터의 활용

## 구조체 배열

구조체 자료형의 정보를 연속적인 공간에 저장
각 저장공간(인덱스)마다 구조체 데이터를 담고 있음

# 구조체와 포인터의 활용

```
lecture c[] = { {"프로그래밍기초", 2, 6, 4},
                {"자바프로그래밍", 3, 3, 3},
                {"논리와비판적사고", 1, 3, 2} };
```

c[0] c[0].name : "프로그래밍기초" c[0].type : 2 c[0].credit : 6 c[0].hour : 4

c[1] c[1].name : "자바프로그래밍" c[1].type : 2 c[1].credit : 6 c[1].hour : 4

c[2] c[2].name : "논리와비판적사고" c[2].type : 2 c[2].credit : 6 c[2].hour : 4

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

typedef struct {
    char name[20];
    int type;
    int credit;
    int hour;
} lecture;

char lectype[4][100] = { "교양", "일반선택", "전공필수", "전공선택" };
char head[4][100] = { "강좌명", "강좌구분", "학점", "시수" };

```

```
15  int main() {
16      lecture course[] = { {"프로그래밍기초", 2, 6, 4},
17                           {"자바프로그래밍", 3, 3, 3},
18                           {"논리와비판적사고", 1, 3, 2} };
19
20      int size, i;
21
22      size = sizeof(course) / sizeof(course[0]);
23
24      printf("%14s %12s %6s %6s\n", head[0], head[1], head[2], head[3]);
25      printf("=========================================\n");
26
27      for (i = 0; i < size; i++) {
28          printf("%16s %10s %5d %5d\n", course[i].name, lectype[course[i].type], course[i].credit, course[i].hour);
29      }
30
31      return 0;
32  }
```

Microsoft Visual Studio 디버그 콘솔

```
          강좌명       강좌구분    학점    시수
=========================================
    프로그래밍기초     전공필수       6       4
    자바프로그래밍     전공선택       3       3
  논리와비판적사고     일반선택       3       2
```

# 동적 메모리 할당

동적 메모리 할당 (dynamic memory allocation)

프로그램 실행 중에 필요한 메모리를 할당하는 방법이다.
메모리 사용 예측이 정확하지 않고 실행 중에 메모리 할당이 필요하다면
동적 메모리 할당 방식이 적합하다.

동적 메모리 할당 함수를 호출하여 힙(heap) 영역에 메모리를 확보한다.
메모리는 사용 후 free() 함수를 사용하여 해제한다.

# 동적 메모리 할당

동적 메모리 할당 함수 3가지

void* malloc(size_t);
-> 인자 만큼의 메모리 할당 후 기본 주소 반환

void* calloc(size_t, size_t);
-> 뒤 인자 만큼의 메모리 크기로 앞 인자 수 만큼 할당 후 기본 주소 반환

void* realloc(void*, size_t);
-> 앞 인자의 메모리를 뒤 인자 크기로 변경 후 기본 주소 반환

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <stdlib.h>

int main() {
    int n = 0;

    printf("입력할 점수의 개수를 입력 >> ");
    scanf("%d", &n);

    int* ary = NULL;
    ary = (int*)malloc(sizeof(int) * n);

    if (ary == NULL) {
        printf("메모리 할당에 문제가 있습니다.");
        exit(1);
    }

```

malloc(): 은 할당된 공간이
모두 쓰레기 값으로 채워진다.

메모리 할당 실패시 NULL 반환

```c
    // 표준입력과 처리
    printf("%d개의 점수 입력 >> ", n);

    int sum = 0;
    for (int i = 0; i < n; i++) {
        scanf("%d", (ary + i));
        sum += *(ary + i); // sum += ary[i];
    }

    // 표준입력 자료와 결과 출력
    printf("입력 점수: ");

    for (int i = 0; i < n; i++)
        printf("%d ", *(ary + i));
    printf("₩n");

    printf("합: %d 평균: %.1f₩n", sum, (double)sum / n);

    free(ary);

    return 0;
}
```

메모리 해제

```
Microsoft Visual Studio 디버그 콘솔
입력할 점수의 개수를 입력 >> 3
3개의 점수 입력 >> 30 40 50
입력 점수: 30 40 50
합: 120 평균: 40.0

C:\Users\minsu\Desktop\studyC\Project1\x6
개).
이 창을 닫으려면 아무 키나 누르세요...
```

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <stdlib.h>

int main() {
    int n = 0;

    printf("입력할 점수의 개수를 입력 >> ");
    scanf("%d", &n);

    int* ary = NULL;
    ary = (int*)calloc(n, sizeof(int));

    if (ary == NULL) {
        printf("메모리 할당에 문제가 있습니다.");
        exit(1);
    }

```

calloc(); 은 할당된 공간이
모두 0 으로 초기화된다.

메모리 할당 실패시 NULL 반환

```c
    // 표준입력과 처리
    printf("%d개의 점수 입력 >> ", n);

    int sum = 0;
    for (int i = 0; i < n; i++) {
        scanf("%d", (ary + i));
        sum += *(ary + i); // sum += ary[i];
    }

    // 표준입력 자료와 결과 출력
    printf("입력 점수: ");

    for (int i = 0; i < n; i++)
        printf("%d ", *(ary + i));
    printf("₩n");

    printf("합: %d 평균: %.1f₩n", sum, (double)sum / n);

    free(ary);

    return 0;
}
```

메모리 해제

```
Microsoft Visual Studio 디버그 콘솔
입력할 점수의 개수를 입력 >> 3
3개의 점수 입력 >> 30 40 50
입력 점수: 30 40 50
합: 120 평균: 40.0

C:\Users\minsu\Desktop\studyC\Project1\x6
개).
이 창을 닫으려면 아무 키나 누르세요...
```

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

void myprintf(int* ary, int n);

int main(void) {
    int* reary, * cary;

    if ((cary = (int*)calloc(3, sizeof(int))) == NULL) {
        printf("메모리 할당이 문제가 있습니다.₩n");
        exit(EXIT_FAILURE);
    }

    if ((reary = (int*)realloc(cary, 4 * sizeof(int))) == NULL) {
        printf("메모리 할당이 문제가 있습니다.₩n");
        exit(EXIT_FAILURE);
    }

    myprintf(reary, 4); //앞 3개는 기본 값인 0 출력, 마지막 하나는 쓰레기 값이 저장

    free(reary);

    return 0;
}
```

realloc(): 은 할당된 공간이
모두 쓰레기 값으로 채워진다.

```
26  void myprintf(int* ary, int n) {
27      for (int i = 0; i < n; i++)
28          printf("ary[%d] = %d ", i, *(ary + i));
29      printf("₩n");
30  }
```

```
Microsoft Visual Studio 디버그 콘솔
ary[0] = 0 ary[1] = 0 ary[2] = 0 ary[3] = -842150451

C:\Users\minsu\Desktop\studyC\Project1\x64\Debug\2023_
개).
이 창을 닫으려면 아무 키나 누르세요...
```

realloc(); 은 기존에 동적 할당한 크기를 수정해서
다시 공간을 재할당할 때 사용한다.

끝