Start

가변인자?

```
#include <stdio.h>
#include <stdarg.h>

void printvar(char* types, ...);

int main(void)
{
    printvar("scdi", "Hello, World", '!', 4.5, 100); // 가변 인자 전달
    return 0;
}

// 가변 매개변수
void printvar(char* types, ...) {
    va_list argp; // 각 가변 인자를 저장하는데 쓰일 타입

    va_start(argp, types); // va_list를 초기화하는 매크로. types의 크기
    while (*types != '\0') {

        switch (*(types++)) {
        case 's':
            printf("%s ", va_arg(argp, char*));// argp가 가지고 있는 값 or 가리키는
주소를 참조한 값을 가져오는 매크로
            break;
        case 'c':
            printf("%c ", va_arg(argp, char));
            break;
        case 'd':
            printf("%.1lf ", va_arg(argp, double));
            break;
        case 'i':
            printf("%d ", va_arg(argp, int));
            break;
        default:
            break;
        }
    }
    va_end(argp); // 사용이 끝나고 인수 리스트를 정리하는 매크로
}
```

- 좋은 설명 블로그

타입 및 매크로	설명
va_list	포인터의 데이터형
va_start()	인수의 목록을 초기화하는 데 사용되는 매크로
va_arg()	변수 목록에서 차례대로 각각의 인수를 읽어들이는 데 사용되는 매크로
va_end()	모든 인수를 받아들이고 나서 정리 동작을 수행하는 데 사용되는 매크로

va_start()

```
#define va_start(ap, v) ( (ap) = (va_list)_ADDRESSOF(v) + _INTSIZEOF(v) )
```

va_arg()

```
#define va_arg(ap, t) (*(t*)((ap += _INTSIZEOF(t)) - _INTSIZEOF(T)))
```

va_end()

```
#define va_end(ap) (ap = (va_list)0);
```

printf 가 가장 대표적인, 흔한, 잘 알고있는 가변인자 함수

```
#include <stdio.h>
int printf(const char * restrict format, ...);

printf("%d %s %c .. ", 100, "string", 'A');
```

복습...

구조체

- 사용자가 자료형들을 모아 하나의 데이터 형식으로 정의해서 담는 상자 / **사용자 정의 자료형**

구조체 정의 및 선언, 사용

- typedef (자료형 정의)
- 구조체 멤버
- 멤버변수 접근 : . (도트 연산자)으로 접근

```
struct bookIF { // 기본적으로 자료형은 'struct bookIF' 키워드를 포함한 풀네임으로 써주거나
    int bookNumber;
    char *title;
    char author[10];
}

typedef struct bookInfo {  // typedef를 이용해 자료형을 Book 처럼 축약형으로 만들어줘도 된다.
    int bookNumber;
    char *title;
    char author[10];
} Book;



void main() {
    struct bookIF b1, b2; // 키워드를 포함한 구조체 선언
    Book book1;           // 축약형 이름을 이용한 구조체 선언
    struct bookInfo book2;

    ...

    book1.bookNum = 10; // 내장하고 있는 변수에 접근
    book1.bookTitle  // error! undeclare 정의되지 않은 변수입니다..
    book1.title = "Book name";

    1. scanf("%s", book1.author);       // 둘다
    2. scanf("%s", &book1.author[0]);   // 가능
}
```

구조체 배열

- 배열하고 똑같다. 자료형을 사용자 정의 자료형인 구조체로 가질 뿐.
    - ex) struct bookInfo books[10]
    - 또는 Book books[10]

선언 및 사용

```
typedef struct bookInfo {
    int bookNum;
    char *title;
    char author[10];
} Book;

// 배열하고 다를게 없다. 똑같은 '배열' 이다!!!
BooK b1[10]; // BOOK을 Type으로 가지는 10개짜리 배열

// b1이 의미하는 바는??
b1 == &b1[0];

// 접근 및 사용할 때
b1[0].bookNum = 300;
b1[0].title = "First Book";
strcpy(b1[0].author, "Uncnown");
b1[0].author[2] = 'k';
```

간단한 응용

- 간이 단어장 만들기

```
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#define SIZE 3

typedef struct word {
    char eng[20];
    char kor[20];
} WORD;

void main() {
    WORD dict[SIZE];

    for (int i = 0; i < SIZE; i++) {
        printf("영어 단어 입력 >> ");
        scanf("%s", dict[i].eng);
        printf("한글 단어 입력 >> ");
        scanf("%s", dict[i].kor);
    }

    for (int i = 0; i < SIZE; i++) {
        printf("영어 단어 : %s\n한글 단어 : %s\n", dict[i].eng, dict[i].kor);
    }
}
```

구조체 포인터

- **구조체 자료형의 주소**를 담는 포인터
    - 포인터 자료형이 각각 달랐듯, '구조체'의 주소를 담을 수 있는 변수이다.
- 마찬가지로 `자료형 * 이름` 형태로 선언한다.
- 멤버변수 접근시 도트연산자 `.` 을 썼었는데, 포인터에서는 `->` 를 사용

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

typedef struct bookInfo {
    int bookNum;
    char* title;
    char author[10];
} Book;

Book* bp; // 선언 포인터 size도 역시 4byte

int main() {
    Book bArr[2] = { {1, "first", "Park"}, {2, "second", "Kim"} };
    bp = bArr; // == &bArr[0] 구조체 배열의 이름이 주소

    // 구조체 배열 접근
    printf("%d, %s, %s\n", bArr[0].bookNum, bArr[0].title, bArr[0].author);
    printf("%d, %s, %s\n", bArr[1].bookNum, bArr[1].title, bArr[1].author);

    // 포인터 형식 접근
    printf("%d, %s, %s\n", bp->bookNum, bp->title, bp->author); // ->로 접근
    printf("%d, %s, %s\n", (bp + 1)->bookNum, (bp + 1)->title, (bp + 1)-
>author);
    // 인덱스를 더해서 해당 포인터 값에 ->로 접근

}
```

함수와 구조체

함수의 매개변수 인자로 쓰이는 구조체

```
#include<stdio.h>

typedef struct bookInfo {
    int bookNum;
    char* title;
    char author[10];
} Book;

// 구조체
void func1(struct bookInfo b1) {
```

```
    b1.bookNum  ...
    b1.title    ...
    b1.author   ...
}
void func2(Book b1) {
    b1.bookNum  ...
    b1.title    ...
    b1.author   ...
}
```

- 구조체는 value type. **Call by value**!!

함수의 매개변수 인자로 쓰이는 구조체 배열, 구조체 포인터

```
#include<stdio.h>

typedef struct bookInfo {
    int bookNum;
    char* title;
    char author[10];
} Book;

// 구조체 배열, 구조체 포인터
void func3(struct bookInfo books[10], int size) {
    for(int i=0; i<size; i++) {
        printf("%d, %s, %s\n", books[i].bookNum, books[i].title,
books[i].author);
    }
}
void func4(struct bookInfo* books, int size) {
    for(int i=0; i<size; i++) {
        printf("%d, %s, %s\n", (books+i)->bookNum, (books+i)->title, (books+i)-
>author);
    }
}
void func5(Book books[10], int size) {
    for(int i=0; i<size; i++) {
        printf("%d, %s, %s\n", books[i].bookNum, books[i].title,
books[i].author);
    }
}
void func6(Book* books, int size) {
    for(int i=0; i<size; i++) {
        printf("%d, %s, %s\n", (books+i)->bookNum, (books+i)->title, (books+i)-
>author);
    }
}
```

- 함수에 구조체 배열과 포인터를 넘겼을 때는 **Call by reference**
- 배열일 때는 `.` 도트 연산자, 포인터일 때는 `->` 를 이용해 접근해야 한다!

함수의 반환형으로 쓰이는 경우

```
#include<stdio.h>

typedef struct bookInfo {
    int bookNum;
    char* title;
    char author[10];
} Book;

// 구조체
struct bookInfo func1();
Book func2();

// 구조체 배열, 구조체 포인터
struct bookInfo* func3();
Book* func4();
```

구조체에서도 Call by value, Call by reference

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

typedef struct data{
    int num1;
    int num2;
} data;

void input(data* p); // 주소를 넘겨 직접 접근하는 call by reference
data swap(data t); // 변수의 값을 복사해서 넘기는 call by value
void print(data t); // call by value
void main(){
    data t;

    input(&t); // 인자로 주소 넘김
    printf("원래 값:");
    print(t);

    t=swap(t); // 반환 받음

    printf("바뀐 후:");
    print(t);
}

void input(data* p){
    printf("두 개의 정수를 입력:");
    scanf("%d %d",&p->num1,&p->num2);
}

data swap(data t){
    int temp;

    temp=t.num1;
```

```
    t.num1=t.num2;
    t.num2=temp;

    return t; // 값을 반환
}
void print(data t){
    printf("%d %d\n",t.num1,t.num2);
}
```

- 구조체는 하나의 '자료형'이므로, 기본적으로 값을 복사해서 넘기는 **값에 의한 호출**이다.
- 포인터로 넘기게 되면 당연히 주소를 넘겨 직접 접근하는 **주소에 의한 호출**이다.
- swap 함수로 지역변수의 '값을 반환' 하기 때문에 그 값을 기존 t에 대입해 swap이 된 것.

간단한 응용

- 사원 정보 구조체 profile을 정의하여 구조체 배열을 선언하고, 함수에 넘겨서 특정 조건에 맞는 사원 정보만 출력하는 프로그램

```
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#include<string.h>
#define SIZE 3

typedef struct profile {
    char name[20];
    double grade;
    int english;
}profile;

void input_data(profile* ps, int size);
void good_find(profile* ps, int size);

void main() {
    profile person[SIZE];
    puts("우수 사원 기준 : 학점 3.8 이상, 영어 점수 800 이상\n");

    input_data(person, SIZE);
    good_find(person, SIZE);
}
void input_data(profile* ps, int size) {
    int i = 0;

    while (i < size) {
        printf("이름: ");
        scanf("%s", ps->name);
        getchar();

        printf("학점: ");
        scanf("%lf", &ps->grade);
        getchar();

        printf("영어점수: ");
        scanf("%d", &ps->english);
```

```
        getchar();
        puts("");
        i++, ps++;
    }
}
void good_find(profile* ps, int size) {
    int i = 0;

    puts("우수사원은");
    while (i < size) {
        if (ps->grade >= 3.8 && ps->english >= 800)
            printf("이름: %s, 학점: %.1lf, 영어점수: %d\n", ps->name, ps->grade,
ps->english);
        i++, ps++;
    }
}
```

실습 문제

학생 정보 관리 프로그램을 구현하라.

정의

- 구조체 **student**는 다음과 같은 조건을 가진다.
  - **멤버변수**로 {
    - **학생 이름**을 담을 크기 10의 char형 배열 name
    - **학번**을 담을 int형 변수 std_num
    - **담당 교수** 정보를 담은 구조체 변수 professor

    } STD를 축약형 이름 (구조체 별칭) 으로 가진다.
- 구조체 **professor**는 다음과 같은 조건을 가진다.
  - **멤버변수**로 {
    - **교수 이름**을 담을 크기 10의 char형 배열 name
    - **담당 전공**을 담을 크기 20의 char형 배열 subject

    } PRO를 축약형 이름 (구조체 별칭) 으로 가진다.

main

- 3명의 학생 정보를 담을 구조체 배열을 선언하여, 학생 이름 / 학번 / 담당 교수 정보를 반복문으로 입력받는다.
- 입력받은 모든 학생 정보 내용을 출력하는 show_info() 함수를 호출한다.
- 학생 정보 중, **10학번 이상의 화석**들만 출력하는 oldStu_info() 함수를 호출한다.

show_info()

- 매개변수로는 구조체 포인터, 학생 수를 받는다.
- 포인터 접근 방식을 이용해 다음 예시와 같이 출력한다.

```
학생이름 : 마이콜
학번 : 2010
교수이름 : 고길동
담당전공 : Sing

학생이름 : 해리포터
학번 : 2000
교수이름 : 덤블도어
담당전공 : Magic

학생이름 : 이학생
학번 : 2021
교수이름 : 나교수
담당전공 : Education
```

oldStu_info()

- 매개변수로는 구조체 포인터, 학생 수를 받는다.
- 포인터 접근 방식을 이용해 다음 예시와 같이 출력한다.

```
화석들 목록은 다음과 같습니다.

학생이름 : 마이콜
학번 : 2010
교수이름 : 고길동
담당전공 : Sing

학생이름 : 해리포터
학번 : 2000
교수이름 : 덤블도어
담당전공 : Magic
```

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

typedef struct professor {
    char name[10];
    char subject[20];
} PRO;

typedef struct student {
    char name[10];
    int std_num;
    PRO professor;
} STD;

void show_info(STD*, int);
void oldStu_info(STD*, int);

void main() {
    STD students[3];
```

```
    for (int i = 0; i < 3; i++) {
        scanf("%s %d %s %s", students[i].name, &students[i].std_num,
students[i].professor.name, students[i].professor.subject);
    }
    puts("\n");

    show_info(students, 3);
    oldStu_info(students, 3);
}

void show_info(STD* students, int size) {
    puts("show_info 호출");

    for (int i = 0; i < size; i++) {
        printf("학생 이름 : %s\n", (students + i)->name);
        printf("학번 : %d\n", (students + i)->std_num);
        printf("교수이름 : %s\n", (students + i)->professor.name);
        printf("담당전공 : %s\n", (students + i)->professor.subject);
        puts("");
    }
    puts("\n");
}

void oldStu_info(STD* students, int size) {
    puts("oldStu_info 호출");
    for (int i = 0; i < size; i++) {
        if ((students + i)->std_num <= 2010) {
            printf("학생 이름 : %s\n", (students + i)->name);
            printf("학번 : %d\n", (students + i)->std_num);
            printf("교수이름 : %s\n", (students + i)->professor.name);
            printf("담당전공 : %s\n", (students + i)->professor.subject);
            puts("");
        }
    }
}
```

파일 입출력

기본 입력 stdin - 키보드 / 마우스 입력

기본 출력 stdout - 모니터 (콘솔) 출력

- 파일을 읽어오거나 저장하는 방법
- '문자열'을 기준으로 한다.

과정

- 파일 처리를 위해서는 기본적으로 세 가지 단계가 존재
    - 파일 열기 (f**open**)
    - 파일 작업 읽고 쓰기 등
    - 파일 닫기 (f**close**)

파일 열기

FILE* fopen(const char *name, const char *mode)

```
FILE *fp = fopen("hello.txt", "r");
```

- name : path (파일의 경로)
- mode : 작업 설정 (읽기, 쓰기, 추가 등등)

모드	설명
"r"	**읽기 모드로 파일을 연다.**
"w"	**쓰기 모드로 파일 생성, 파일이 존재하면 기존 내용이 지워진다.**
"a"	추가 모드로 파일을 생성. 파일이 있으면 데이터가 끝에 추가 된다.
"r+"	읽기와 쓰기 모드로 파일을 연다. 파일이 반드시 존재해야한다.
"w+"	읽기와 쓰기 모드로 파일을 생성, 파일이 존재하면 새 데이터가 기존 데이터를 덮어 쓴다.
"a+"	읽기와 추가 모드로 파일을 연다. 파일이 존재하면 데이터가 파일 끝에 추가된다. 읽기는 어떤 위치에서나 가능
"b+"	이진 파일 모드로 파일을 연다.

파일 열기 성공여부 확인

```
if(fp != NULL) {
    printf("fopen successful\n");
} else {
    printf("fopen failed\n");
    exit(0);
}
```

- 저수준 시스템 프로그래밍에서 '파일 기술자(파일 디스크립터)' 개념 (fp)
    - 표준 입력 0, 표준 출력 1, 표준 에러 2가 기본 할당
    - 가장 작은 정수인 3 부터 시작...

파일 작업

함수 종류

종류	입력	출력
문자 단위	int fgetc(FILE *fp)	int fputc(int c, FILE *fp)
문자열 단위	char *fgets(char *buf, int n, FILE *fp)	int fputs(const char *buf, FILE *fp)
타입지정 입출력	int fscanf(FILE *fp, ...)	int fprintf(FILE *fp, ...)
이진 데이터	fread(char *buf, int size, int count, FILE *fp)	fwrite(char *buf, int size, int count, FILE *fp)

- get / put
- scan / print
    - 어디로부터 받아오는가? 어디로 넣는가??
    - 어디로부터 입력받는가? 어디로 출력 하는가??
        - 파일 디스크립터 개념으로 보면.. 기본 입력은 키보드(0), 기본 출력은 콘솔(1), fopen() 했으면 할당 가능한 작은 정수(3)부터..
        - 키보드(0)로부터 입력 받는다 / 파일(3)로부터 입력 받는다(== 파일을 읽어들인다)
        - 콘솔(1)로 출력한다 / 파일(3)로 출력한다. (== 파일에 쓴다)

int fgetc(FILE *fp)

- 인자로 읽어올 파일의 파일 포인터를 준다.
- 반환형은 int로, 아스키코드에 따라 읽어 문자로 인식한다.
- 파일의 끝에서는 EOF(-1)를 반환한다.
- fp는 파일에서 '커서'에 해당하는 개념으로 보면 될 듯

int fputc(int c, FILE *fp)

- 인자로 출력할 문자, 출력할 곳(파일)의 파일 포인터를 준다.
- ASCII CODE 문자 c를 fp가 가리키는 파일에 출력한다. (파일에 c를 쓴다)
- 반환형은 int로, 성공시 해당 문자를 반환하고 실패시 EOF(-1)를 반환한다.

char *fgets(char *buf, int n, FILE *fp)

- 인자로 읽어올 문자열을 저장할 버퍼와, 읽어올 크기, 파일 포인터를 준다.
- 반환형은 char*, 가지고 온 문자열을 반환하거나 파일의 끝에 도달했을때 널 포인터 반환.
- 파일의 끝에서는 EOF(-1)를 반환한다.
- fp는 파일에서 '커서'에 해당하는 개념으로 보면 될 듯

int fputs(const char *buf, FILE *fp)

- 인자로 출력할 문자열, 출력할 곳(파일)의 파일 포인터를 준다.
- 문자열 buf를 fp가 가리키는 파일에 출력한다. (파일에 buf를 쓴다)
- 반환형은 int로, 성공시 buf의 길이를 반환하고 실패시 EOF(-1)를 반환한다.

간단한 응용

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

void main() {
    FILE* fpw = fopen("./testDir/test.txt", "r"); // 경로 개념 - 절대경로, 상대경로
    // C:/workspace/c/STP_C_Study/21-2-C_STUDY/testDir/test.txt

    if (fpw != NULL) {
        puts("fopen successful");
    } else {
        puts("fopen failed");
        return;
    }

//  fputs("Hello world!!~!!", fpw);

    char buffer[30];
    fgets(buffer, 20, fpw); // 파일로부터 입력 받아오기

    puts(buffer); // 표준 출력
    fputs(buffer, stdout); // 표준 출력

    fclose(fpw);
}
```

int fscanf(FILE *fp, ...)

int fprintf(FILE *fp, ...)

여기서도 나오는 파일 디스크립터 개념의 접근... stdin, stdout

간단한 응용

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

void main() {
    int num;

    FILE* fp = fopen("./test.txt", "w+");

    fscanf(fp, "%d", &num); // 파일에서 읽어오기
    fprintf(fp, "%d", num); // 파일에 출력(저장)

    fscanf(stdin, "%d", &num); // 표준입력에서 읽어오기 (키보드)
    fprintf(stdout, "%d", num); // 표준출력 저장 (콘솔)

}
```

파일 닫기

int fclose(FILE *stream)

```
fclose(fp);
```

- 파일을 열었으면, 닫아줘야 한다.
- 안해주면 프로세스 계속 실행, 불필요한 자원 낭비

파일 경로

- 절대경로
    - /c/~~~ 컴퓨터의 최상단 root 경로부터 시작해 절대적인 경로
- 상대경로
    - 현재 위치를 기준으로 하는 경로