

9장. 파일 입출력, 전처리기

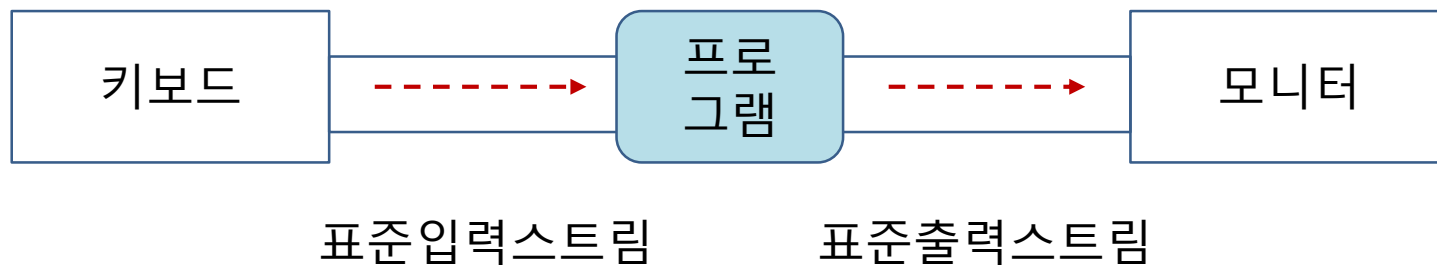
Visualstudio 2019



파일 입출력

스트림이란(Stream)?

데이터를 입력하고 출력하기 위한 다리(연결 통로)이다.



기본개방 스트림파일 – 메모리에 구성한 논리적 파일(물리적 x)
운영체제가 제어 및 관리함

스트림	설명	장치
stdin	표준 입력을 담당	키보드
stdout	표준 출력을 담당	모니터
stderr	표준 에러를 담당	모니터



파일 입출력

버퍼와 버퍼링

버퍼(Buffer)는 처리할 데이터를 임시로 저장하는 저장소

입력버퍼는 데이터를 저장하기 위한 버퍼이며, 출력 버퍼는 데이터를 출력하기 위한 버퍼이다.

콘솔 입출력 함수 및 파일 입출력 함수

콘솔은 키보드나 모니터와 같은 표준 입출력 장치이다.

콘솔 입출력 함수

getchar(), putchar(), gets(), puts(), printf(), scanf() 등

파일 입출력 함수

fgetc(), fputc(), fgets(), fputs(), fscanf(), fprintf() 등



파일 입출력

◆ 파일 입출력의 필요성

프로그램 실행 중에 메모리에 저장된 데이터는 프로그램이 종료되면 사라진다. 데이터를 프로그램이 종료된 후에도 계속해서 사용하려면 파일에 저장하고 필요할때 파일을 읽어서 데이터를 사용할 수 있다.

파일을 이용한 입출력 과정

1. 파일 스트림을 생성한다 -> 파일 포인터 생성
2. 파일을 연다. -> `fopen()` 함수
3. 파일 입출력을 수행한다. -> `fgetc()`, `fputc()`, `fgets()`, `fputs()`,
`fprintf()`, `fscanf()`
1. 파일을 닫는다. -> `fclose()`



파일 입출력

➤ 파일 쓰기

fopen(파일이름, "w") – "w"는 쓰기 모드

```
FILE* fp;  
fp = fopen("hello.txt", "w");  
  
fputc('H', fp);  
fputc('E', fp);  
fputc('L', fp);  
fputc('L', fp);  
fputc('O', fp);  
fputc('\n', fp);  
  
fputs("hello", fp);  
  
fclose(fp);
```

fputc(문자, 파일포인터)함수 – 한 문자 입력,
fputs(문자열, 파일포인터)함수 – 문자열 입력

파일(F) 편집(E) 서식(O)

HELLO
hello



파일 입출력

➤ 파일 읽기 1

fopen(파일이름, "r") – "r"는 읽기 모드

```
FILE* fp = fopen("hello.txt", "r");
if (fp == NULL) {
    printf("해당 파일이 없습니다.\n");
    return 1; //에러가 있을 때 반환(1 또는 -1)
}
int ch;

while ((ch = fgetc(fp)) != EOF) { // EOF = -1
    //putchar(ch); 콘솔에 출력
    printf("%c", ch);
}
```

fgetc(파일포인터) – 파일에서 한 문자씩 읽기, 성공하면 읽은 문자열의 포인터를 반환, 실패하면 NULL 반환

```
/*while (1) {
    int ch = fgetc(fp);
    if (ch == EOF) break;
    //printf("%c", ch);
    putchar(ch);
}*/

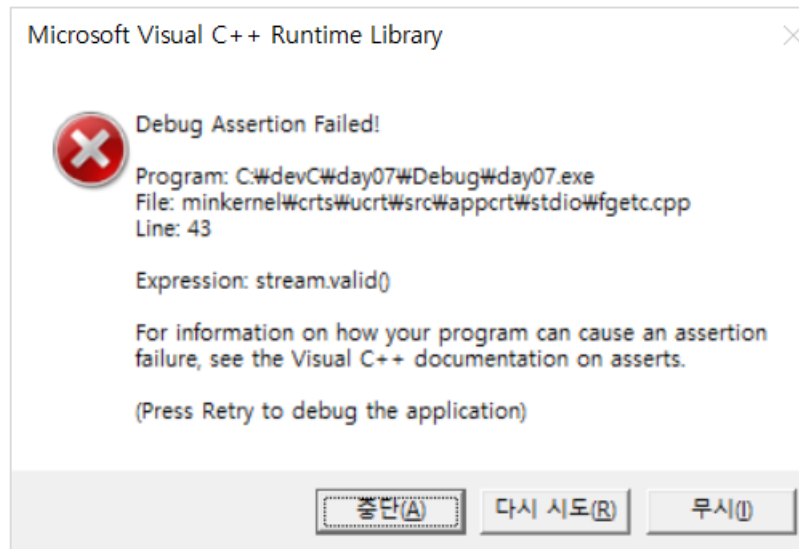
fclose(fp);
```



파일 입출력

➤ 파일 읽기 1

파일 이름에 오류가 있거나, 해당 파일이 없을 때 오류 발생



파일 입출력

➤ 파일 읽기 2

fgets(버퍼, 버퍼크기, 파일포인터) – 문자열 읽기

성공하면 읽은 문자열의 포인터를 반환, 실패하면 NULL 반환

```
char buffer[20];  
//int buffer;  
  
FILE* fp = fopen("hello.txt", "r");  
  
/*fgets(buffer, sizeof(buffer), fp);  
printf("%s\n", buffer);*/  
  
//여러줄 읽을때  
while (fgets(buffer, sizeof(buffer), fp) != NULL) {  
    printf("%s", buffer);  
};  
  
fclose(fp);
```



파일 입출력

➤ 파일 쓰기

fopen(파일이름, "a") – "a"는 추가 쓰기 모드

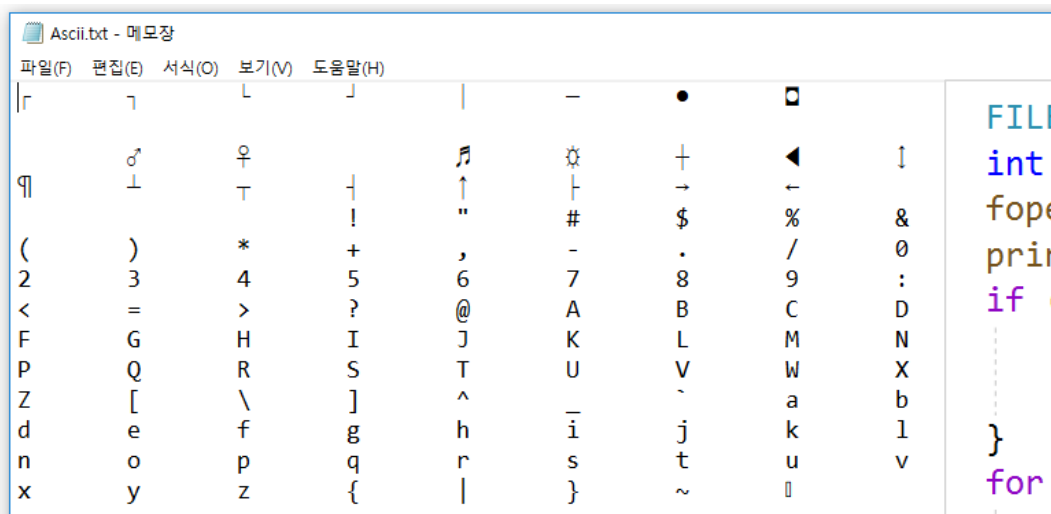
fprintf(파일포인터, 문자열 포맷) – 파일에 쓰기 함수

```
FILE* fp;  
fp = fopen("hello2.txt", "a");  
  
//fprintf(fp, "안녕하세요~.\n"); //추가로 저장됨  
fprintf(fp, "좋은 하루되세요!\n");  
  
fclose(fp);
```



파일 입출력

➤ 파일 쓰기(저장) 예제 - 아스키 파일 쓰기



```
FILE* fp;
int i = 0;
fopen_s(&fp, "Ascii.txt", "w");
printf("ASCII 테이블을 작성합니다.");
if (fp == NULL) {
    printf("파일을 생성할 수 없습니다.");
    return -1;
}
for (i = 1; i < 128; i++) {
    if (i % 10 == 0) {
        fputc('\n', fp);
    }
    fputc(i, fp);
    fputc('\t', fp);
}

fclose(fp);
```



파일 입출력

➤ 파일 쓰기(저장) 예제 - 구구단 파일 생성

```
2 x 1 = 2  
2 x 2 = 4  
2 x 3 = 6  
2 x 4 = 8  
2 x 5 = 10  
2 x 6 = 12  
2 x 7 = 14  
2 x 8 = 16  
2 x 9 = 18
```

```
int i, j;  
FILE* fp;  
fp = fopen("gugudan.txt", "w");  
  
for (i = 2; i < 10; i++) {  
    for (j = 1; j < 10; j++) {  
        fprintf(fp, "%d x %d = %d\n", i, j, (i * j));  
    }  
    fprintf(fp, "\n");  
}  
  
fclose(fp);
```

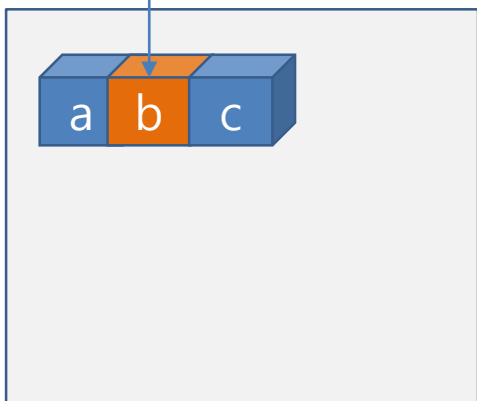


파일 입출력

➤ 파일 복사 – 파일 읽고 쓰기

`fopen_s(&fin, 원본파일, "r")`

`fin`



`fgetc()`

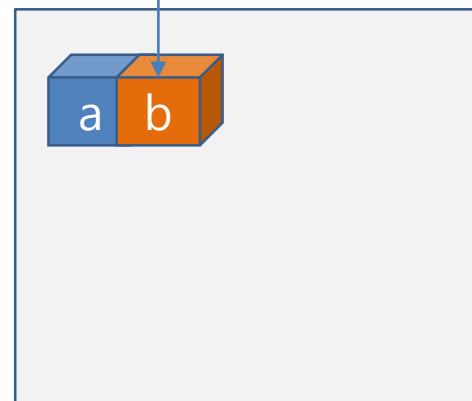


`fputc()`



`fopen_s(&fout, 복사파일, "w")`

`fout`

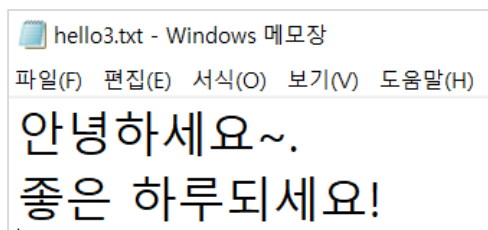
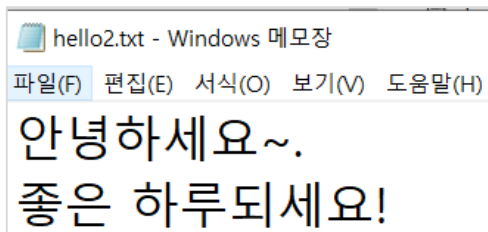


`fin`은 원본파일을 가리키는 파일 포인터이며, `fout`은 새로 생성할 파일을 가리키는 파일 포인터로 두 개의 파일을 오픈하게 된다.



파일 입출력

➤ 파일 복사 - 읽고 쓰기



```
FILE* fin;  
FILE* fout;  
int input = 0;
```

```
fopen_s(&fin, "hello2.txt", "r");  
fopen_s(&fout, "hello3.txt", "w");
```

```
if (fin == NULL) {  
    printf("파일 열기에 실패했습니다.\n");  
    return 1;  
}
```

```
if (fout == NULL) {  
    printf("파일 열기에 실패했습니다.\n");  
    return 1;  
}
```

```
puts("*** 파일에 데이터를 입력 ***");  
while (input != EOF) {  
    input = fgetc(fin); //파일에서 읽어오기  
    fputc(input, fout); //파일에 쓰기  
    fputc(input, stdout); //모니터에 쓰기  
}
```

```
fclose(fin);  
fclose(fout);
```



파일 입출력

➤ 입력받아 저장하기

fscanf_s() - 파일에 표준 입력함수

fprintf() - 파일에 쓰기 함수

```
이름 입력 : 장그래  
국어 점수 입력 : 90  
영어 점수 입력 : 85
```



score.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
장그래 90 85 87.5



파일 입출력

➤ 입력받아 저장하기

```
FILE* fp;
char name[10];
int kor, eng;
double avg;

printf("이름 입력 : ");
fscanf_s(stdin, "%s", name, sizeof(name));
```

```
printf("국어 점수 입력 : ");
fscanf_s(stdin, "%d", &kor);

printf("영어 점수 입력 : ");
fscanf_s(stdin, "%d", &eng);
```

```
fopen_s(&fp, "score.txt", "w");
if (fp == NULL) {
    printf("파일을 생성할 수 없습니다.\n");
    return -1;
}

avg = ((double)kor + (double)eng) / 2;
fprintf(fp, "%s %d %d %3.1lf\n", name, kor, eng, avg);

fclose(fp);
```



파일 입출력

➤ 입력 버퍼의 문제

```
int age;
char name[10];

printf("나이를 입력 : ");
scanf_s("%d", &age);

printf("이름을 입력 : ");
//while (getchar() != '\n');
fgets(name, sizeof(name), stdin);

printf("나이 : %d\n", age);
printf("이름 : %s\n", name);
```

키보드



2	1	␣ n		
---	---	-----	--	--

scanf() 함수가 나이를 입력받은 후 버퍼에 남겨진 개행 문자가 다음에 호출되는 fgets() 함수가 데이터로 읽어드려 문자열을 입력할 기회를 상실한다.



파일 입출력

➤ 성적 리스트 만들기

번호(0이면 종료): 1
이름 입력 : 강하늘
국어 영어 수학 점수 입력 : 95 90 80

번호(0이면 종료): 2
이름 입력 : 이우주
국어 영어 수학 점수 입력 : 85 85 85

번호(0이면 종료): 3
이름 입력 : 정은하
국어 영어 수학 점수 입력 : 70 75 60

번호(0이면 종료): 0

scorelist.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
1 강하늘 95 90 80
2 이우주 85 85 85
3 정은하 70 75 60
```



파일 입출력

➤ 성적 리스트 만들기

```
while (1)
{
    printf("\n번호(0이면 종료): ");
    scanf_s("%d", &no);
    if (no <= 0) break;

    printf("이름 입력 : ");
    while (getchar() != '\n');
    gets(name);
    //gets 함수는 '\n'을 읽어오면 자동 개행되어 입력할 수 없다.

    printf("국어 영어 수학 점수 입력 : ");
    scanf_s("%d %d %d", &kor, &eng, &math);

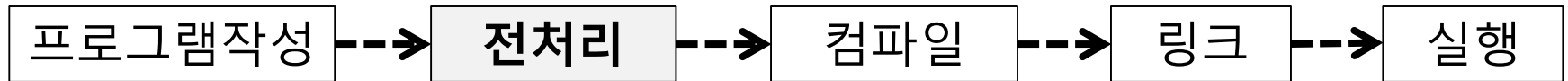
    //파일에 쓰기
    fprintf(fp, "%d %s %d %d %d\n", no, name, kor, eng, math);
}

fclose(fp);
```



전처리기 (preprocessor)

◆ 전처리기(preprocessor)의 목적



C컴파일러가 실제 컴파일 작업을 하기 전에 먼저 처리하는 작업

전처리기가 특정 과정을 미리 처리해 두도록 하면, 이 특정 과정에 해당하는 부분을 여러 개의 파일로 나누어 여러 사람이 공동 개발할 수 있다.

디버깅을 위한 코드와 릴리즈(배포)를 위한 코드를 분리할 수도 있다.



전처리기 (preprocessor)

◆ 소스파일과 헤더파일

소스파일은 확장자가 "*.c"인 파일이며, 헤더 파일은 확장자가 "*.h"인 파일이다.

소스파일과 헤더파일은 전처리 과정을 통해 하나로 합쳐 진다.

- 헤더파일에 두면 좋은 것들
 - 함수의 프로토타입 선언 → main()아래 있을때 원형만 선언
 - 새로운 데이터형의 정의 → 구조체



전처리기 (preprocessor)

```
typedef struct _Student {  
    int num;  
    char name[20];  
}Student;
```

Student.h

```
#include <stdio.h>  
#include "Student.h"  
int main()  
{  
    Student st = { 15, "장그래" };  
    printf("학번 : %d, 이름 : %s \n", st.num, st.name);  
    return 0;  
}
```

main.c



매크로(Macro)

◆ 매크로 상수

#define으로 시작되는 전처리 문장을 매크로라고 한다.

매크로 상수와 매크로 함수로 나누어 짐.

#define PI 3.14

- #define : 전처리기 지시자
- PI : 매크로 상수 이름
- 3.14 : 치환값

1. define문으로 정의한 문자의 뒤에는 세미콜론(;)을 붙이지 않는다.
2. #define 문에 의해 정의되는 식별자는 영어 대문자 사용한다.
3. #define문으로 연산을 하는 경우에는 되도록 괄호를 사용하여 묶어준다.



매크로(Macro)

◆ 매크로 상수

```
#include <stdio.h>
#define PI 3.14

int main() {

    int radius;
    double area, circum;

    printf("반지름을 입력하세요: ");
    scanf_s("%d", &radius);

    area = PI * radius * radius;
    circum = 2 * PI * radius;

    printf("원의 넓이 : %lf\n", area);
    printf("원의 둘레 : %lf\n", circum);
    return 0;
}
```



매크로 상수



메이저리그는 점점 더 빠른 구속을 추구하고 있다. 올 시즌 메이저리그 포심 패스트볼 평균 구속은 시속 93.2마일(150.0km)에 달한다. 이제는 100마일(160.9km)이 넘는 공도 어렵지 않게 볼 수 있게 됐다.

투수에게 있어 구속이 가장 중요한 요소는 아니다. 구종, 제구, 구위 등 수 많은 요소들이 어우러져야 비로소 뛰어난 투구를 할 수 있다. 하지만 같은 조건이라면 당연히 구속이 빠를수록 유리하다. 구속이 빠를수록 타자들이 공에 대처할 수 있는 물리적인 시간이 줄어들기 때문이다.



매크로 상수

◆ 매크로 상수 예제

```
#include <stdio.h>
#define _CRT_SECURE_NO_WARNINGS
#define RATE_KPH_MPH 1.609344

int main() {
    //KPH(킬로미터)를 MPH(마일)로 변환
    int kph;
    double mph;
    //const double RATE_KPH_MPH = 1.609344;

    printf("당신의 구속을 입력하세요[km/H]: ");
    scanf_s("%d", &kph);
    mph = kph / RATE_KPH_MPH;
    printf("당신의 구속은 %.21f[MPH]입니다.\n", mph);

    return 0;
}
```



매크로(Macro)

◆ 매크로 함수

함수처럼 인자를 설정할 수 있는 매크로를 의미함.

매크로 함수라고 부르지만 단순히 치환하기만 하므로 실제로는 함수가 아님

#define SUB(x, y) (x - y)

- #define : 전처리기 지시자
- **SUB(x, y)** : 매크로 함수 이름
- **x - y** : 함수의 기능



매크로 함수

매크로 함수는 함수를 호출하는 것이 아니라 컴파일 할때 코드를 바꾸는 작업이기 때문에 CPU의 함수 호출 처리를 생략할 수 있다.

```
#include <stdio.h>
#define EXPONENTIAL(num) (num) * (num) //매크로 함수

int exponetial(int num) { //일반 함수
    return num * num;
}

int main() {
    int result = 0;

    result = EXPONENTIAL(4);
    //result = exponetial(4);

    printf("결과는 %d입니다.\n", result);

    return 0;
}
```



매크로 함수

◆ 매크로 함수 예제

```
#include <stdio.h>
#define SUM(x, y) (x) + (y)
#define MUL(x, y) (x * y)

int main() {
    int a = 10, b = 20;
    int result;

    printf("a + b = %d\n", SUM(a, b)); //(a) + (b)
    result = 30 / MUL(2, 5);    // result = 30 / 2 * 5(괄호 생략)
    printf("result : %d\n", result);

    return 0;
}
```



조건부 컴파일

- 조건부 컴파일 지시자

- 조건에 따라 소스코드를 선택적으로 컴파일 하므로 보다 이식성 있는 코드를 개발할 수 있음
- 매크로 상수의 존재 유무 또는 매크로 상수값을 검사하여 수행함
- #if, #else, #elif / #ifdef, #ifndef, #endif 등 사용

#if 조건식

컴파일할 수행문1

#else

컴파일할 수행문2

#endif

#if defined 치환값

컴파일할 수행문1

#else

컴파일할 수행문2

#endif



조건부 컴파일

◆ 조건부 컴파일 예제

```
#define VER 10 //치환값이 있는 매크로명 정의
//#define BIT16 //치환값이 없는 매크로명 정의
int main() {

    int max;

    #if VER >= 10
        printf("버전 %d입니다.\n", VER);
    #endif

    printf("===== \n");

    #ifdef BIT16 //매크로명 BIT16이 정의되어 있으면
        max = 32767;
    #else //매크로명 BIT16이 정의되어 있지 않으면
        max = 2147483647;
    #endif

    printf("int형 변수의 최댓값 : %d\n", max);
    return 0;
}
```



조건부 컴파일

◆ 조건부 컴파일 예제

```
#if defined EN
    #define HELLO_MESSAGE "Hello"
#else defined KO
    #define HELLO_MESSAGE "안녕하세요"
#endif
```

message.h

```
#include <stdio.h>
#define KO
#include "message.h"

int main() {
    printf("%s\n", HELLO_MESSAGE);

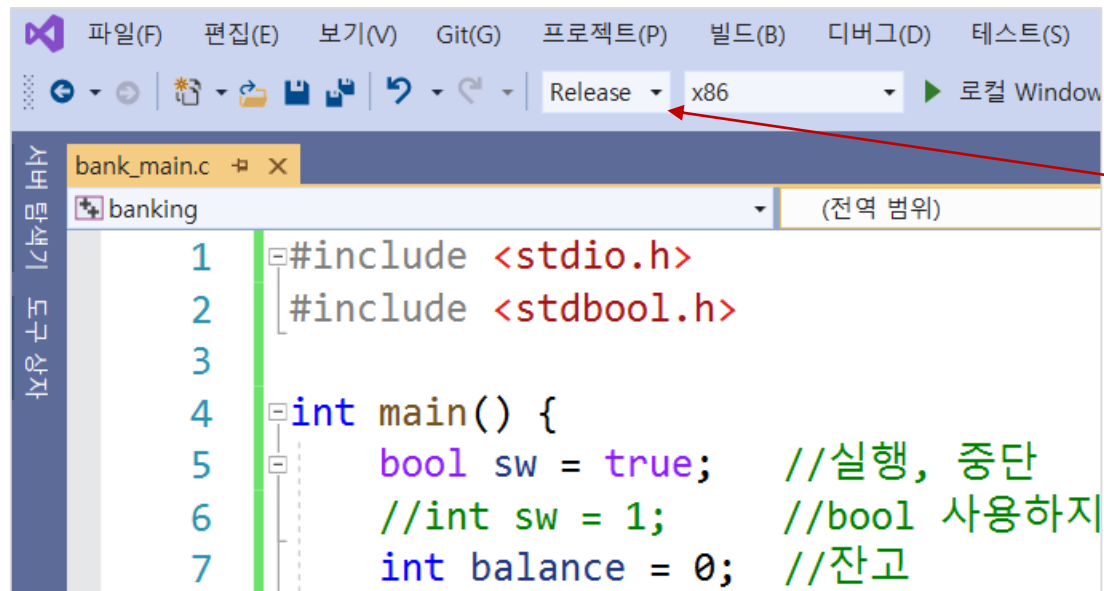
    return 0;
}
```



파일 배포

◆ 파일 배포

파일 배포란 c언어 소스파일을 .exe 실행 파일로 만들어 공개 및 서비스 하는 것을 말한다.



```
1 #include <stdio.h>
2 #include <stdbool.h>
3
4 int main() {
5     bool sw = true;    //실행, 중단
6     //int sw = 1;      //bool 사용하지
7     int balance = 0;   //잔고
```

Debug 모드를
Release 모드로 바꾼다.

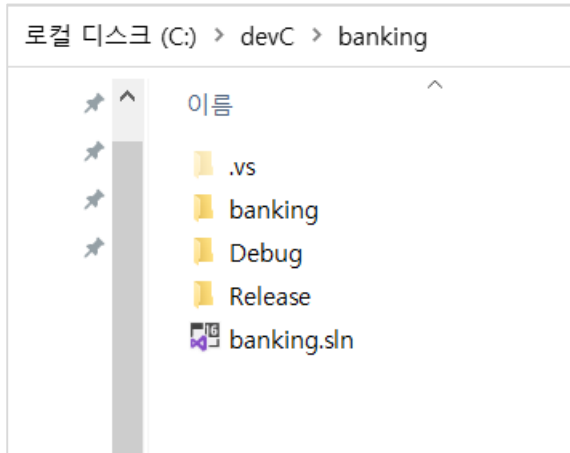


Ct기 + F5로 실행

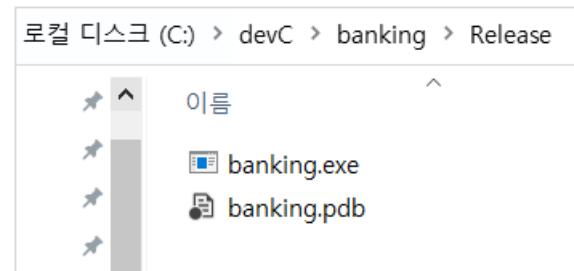


파일 배포

◆ 파일 배포



Release 폴더 생성됨



Banking.exe 파일 생성



파일 배포

- ◆ exe 파일이 실행되지 않는 문제 해결

system("pause") 를 명시함

```
int i, j, seat;
for (i = 0; i < rowNum; i++) {
    for (j = 1; j <= columnNum; j++) {
        seat = i * columnNum + j;
        printf("좌석%d ", seat);
        if (seat == customerNum)
            break;
    }
    printf("\n");
}

system("pause");

return 0;
```

seats.c



◆ 윈도우 터미널(명령 프롬프트)에서 실행하기

```
C:\Users\W김기용>cd C:\WdevCWseatsWRelease

C:\WdevCWseatsWRelease>dir
C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: 4A81-5207

C:\WdevCWseatsWRelease 디렉터리

2021-11-01 오후 09:05 <DIR>      .
2021-11-01 오후 09:05 <DIR>      ..
2021-11-02 오후 02:55              9,216 seats.exe
2021-11-02 오후 02:55          421,888 seats.pdb
                2개 파일              431,104 바이트
                2개 디렉터리 36,521,902,080 바이트 남음

C:\WdevCWseatsWRelease>seats.exe
입장객 수 입력 : 16
좌석 열의 수 입력 : 3
좌석1 좌석2 좌석3
좌석4 좌석5 좌석6
좌석7 좌석8 좌석9
좌석10 좌석11 좌석12
좌석13 좌석14 좌석15
좌석16
계속하려면 아무 키나 누르십시오 . . .
```

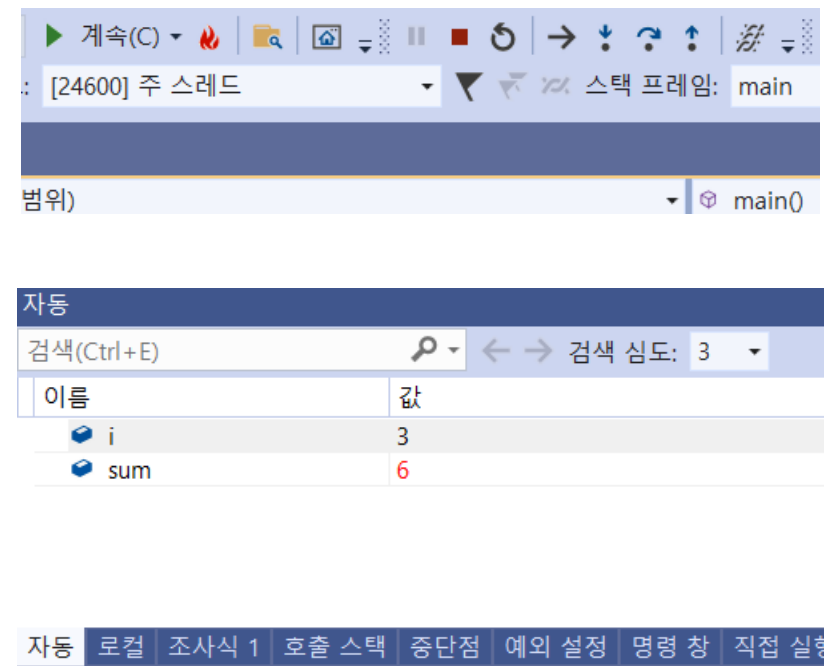


디버깅(Debugging)

◆ 디버깅 작업

- 중단점 설정(F9) : 디버그 > 중단점 설정
- 실행(F10) : 디버그 > 프로시저 단위 실행

```
3 int main() {  
4  
5     int i = 0, sum = 0;  
6  
7     for (i = 1; i <= 10; i++) {  
8         sum = sum + i;  
9     } 경과 시간 1ms 이하  
10  
11     printf("합 : %d", sum);  
12  
13     return 0;  
14 }
```



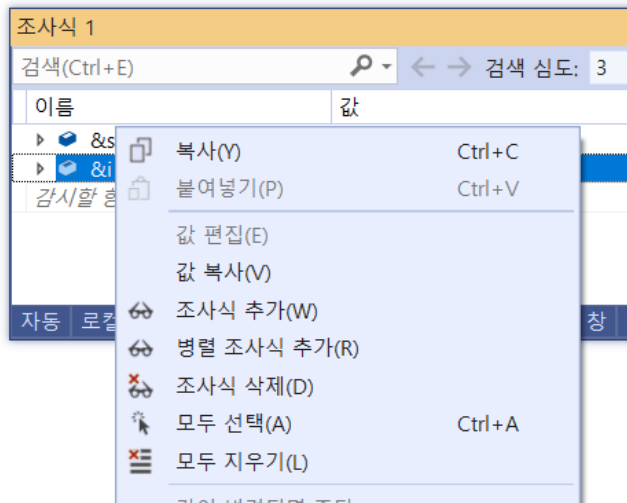
Debugger interface showing the execution of a C program. The main window displays the source code with a breakpoint set at line 7. The right sidebar shows the 'Stack' window with the current frame 'main' and the 'Variables' window showing the values of 'i' (3) and 'sum' (6). The bottom status bar indicates the current state is '자동' (Automatic).

이름	값
i	3
sum	6



디버깅(Debugging)

◆ 디버깅 작업



단축메뉴 > 조사식 추가

&sum : 주소값 확인

조사식 1		
검색(Ctrl+E) 🔍 < > 검색 심도: 3		
이름	값	형식
▶ &sum	0x001df974 {3}	int *
▶ &i	0x001df980 {2}	int *
감시할 항목 추가		
자동 로컬 조사식 1 호출 스택 중단점 예외 설정 명령 창 직접 실행 창 출력 오류		

