

C Programming

Ch.5 주요 라이브러리 함수들

Contents

1. 표준 라이브러리 함수

1. math.h : 수학 함수들

2. stdlib.h : rand, srand, system, atoi, atof

3. time.h : clock

2. 비표준 라이브러리 함수 (콘솔 제어 중심)

1. 입력을 위한 _getch와 _kbhit 함수

2. 콘솔 화면 제어 함수

3. 실행 제어 함수

3. 종합 예제

포인터, 구조체 등 아직 배우지 않은 내용들이 있으므로 100% 이해는 어려움.
현재는 원리 이해보다는 활용에 초점을 맞추어 학습

본 자료에서 소개할 표준 라이브러리 함수들

C 표준 라이브러리	헤더 파일	함수 프로토타입	기능
<assert.h> <complex.h> (since C99) <ctype.h> <errno.h> <fenv.h> (since C99) <float.h> <inttypes.h> (since C99) <iso646.h> (since C95) <limits.h> <locale.h> <math.h> <setjmp.h> <signal.h> <stdalign.h> (since C11) <stdarg.h> <stdatomic.h> (since C11) <stdbool.h> (since C99) <stdckdint.h> (since C23) <stddef.h> <stdint.h> (since C99) <stdio.h> <stdlib.h> <stdnoreturn.h> (since C11) <string.h> <tgmath.h> (since C99) <threads.h> (since C11) <time.h> <uchar.h> (since C11) <wchar.h> (since C95) <wctype.h> (since C95)	math.h	double fabs(double x);	절대값을 반환함
		double round(double x);	반올림 결과를 반환함
		double sin(double x);	sin 값을 반환함
		double log(double x);	자연로그 값을 반환함
		double pow (double x, double y);	x의 y승 값을 반환함
	stdlib.h	int rand(void);	0에서 RAND_MAX까지의 값들 중 무작위 값을 생성함
		void srand (unsigned int seed);	rand 함수를 사용하기 전에 랜덤 넘버 생성기를 초기화함
		int system (const char *string);	string 문자열에 해당하는 커맨드 명령어를 실행함
		int atoi(const char *nptr);	문자열에 포함된 정수값을 반환함
		double atof (const char *nptr);	문자열에 포함된 실수값을 반환함
	time.h	clock_t clock(void);	프로그램이 실행된 이후의 CPU time(clock) 값을 반환. 미리 정의된 CLOCKS_PER_SEC 문자열로 나눴으로써 초 단위 값을 알 수 있음

fabs, sin, log, pow, round 함수 (1)

- ▶ `double fabs(double x)` : x의 절대값
- ▶ `double sin(double x)` : x의 sin 값 (입력 : 라디안)
- ▶ `double log(double x)` : x의 자연로그 값
- ▶ `double pow(double x, double y)` : x의 y승 값
- ▶ `double round(double x)` : x의 반올림 결과
 - C11 표준에 포함, Visual C++ 12.0(2013)부터 지원

fabs, sin, log, pow, round 함수 (2)

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

```
int main(void)
{
```

```
    double num;
```

```
    printf("실수 1개 입력 : ");
    scanf("%lf", &num);
```

```
    printf("절대값      : %10.6f \n", fabs(num));
    printf("반올림값    : %10.6f \n", round(num));
    printf("sin 값       : %10.6f \n", sin(num));
    printf("log 값       : %10.6f \n", log(num));
    printf("3승 값       : %10.6f \n", pow(num, 3));
```

```
}
```

```
실수 1개 입력 : 2.5
절대값      : 2.500000
반올림값    : 3.000000
sin 값      : 0.598472
log 값      : 0.916291
3승 값      : 15.625000
```

```
실수 1개 입력 : -2.5
절대값      : 2.500000
반올림값    : -3.000000
sin 값      : -0.598472
log 값      : -1.103170
3승 값      : -15.625000
```

rand, srand 함수 (1)

▶ rand 함수

- 0 ~ RAND_MAX(=32,767) 범위의 정수 중 하나를 난수로 선택
- 예 : 10 ~ 20 사이의 난수를 5개 생성하고 합계 출력

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

int main(void)
{
    int num;
    int result = 0;

    for (int i = 0; i < 5; i++)
    {
        num = rand() % 11 + 10; // 10~20 사이의 정수값 생성
        printf("%d ", num);
        result += num;
    }

    printf("\n합계 : %d\n", result);
}
```

18 19 19 11 17
합계 : 84

- 몇 개의 정수가 있는가? $(20 - 10) + 1$
 - 최소값은 무엇인가? 10
 - 따라서, $\text{rand()} \% ((20-10) + 1) + 10$
 - 이를 일반화하면,
 $\text{rand()} \% (\text{최대값} - \text{최소값} + 1) + \text{최소값}$

// 10~20 사이의 정수값 생성

문제점 : 실행할 때마다 난수 발생 순서가 동일 → srand 함수 사용

rand, srand 함수 (2)

- ▶ **srand 함수** : 난수 발생기 초기화
 - **srand(정수값)** : 정수값이 동일하면 매번 생성되는 난수 순서 동일
 - 정수값 : **time(NULL)** 함수 사용 - 프로그램 실행 시 마다 다름

```
#include <stdio.h>
#include <time.h>           // time 함수
#include <stdlib.h>
```

```
int main(void)
{
```

```
    int num;
    int result = 0;
```

```
    srand(time(NULL)); // 랜덤 넘버 생성기 초기화
```

```
    for (int i = 0; i < 5; i++)
    {
```

```
        num = rand() % 11 + 10;
        printf("%d ", num);
        result += num;
    }
```

```
    printf("\n합계 : %d\n", result);
```

```
}
```

12 10 11 17 19
합계 : 69

15 13 16 11 19
합계 : 74

system 함수

- ▶ 프로그램 내에서 커맨드 명령어(도스 명령어) 실행
- ▶ 도스 명령어 : 도스창(cmd.exe)을 통해 실행 가능
 - dir, cd, copy, cls, ...

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

int main(void)
{
    system("dir"); //파일 목록 출력
    printf("안녕하세요\n");

    return 0;
}
```

Microsoft Visual Studio 디버그 × + ▾

D 드라이브의 볼륨 : EV0860_1TB
볼륨 일련 번호 : 7EBF-652D

D:\My workspace\C\test_c 디렉터리

날짜	시간	타입	이름	크기
2025-09-29	월 오후 03:49	<DIR>	.	
2025-05-22	목 오후 03:32	<DIR>	..	
2025-09-29	월 오후 03:49		139 main.c	
2025-05-22	목 오후 04:45	<DIR>	test_c	
2025-05-22	목 오후 03:32		1,438 test_c.sln	
2025-09-23	화 오후 08:32		6,456 test_c.vcxproj	
2025-05-22	목 오후 04:45		980 test_c.vcxproj.f	
2025-05-22	목 오후 03:32		168 test_c.vcxproj.u	
2025-09-23	화 오후 08:32	<DIR>	x64	
		5개 파일	9,181 바이트	
		4개 디렉터리	321,029,091,328 바이트	남음

안녕하세요

D:\My workspace\C\test_c\x64\Release\test_c.exe(프로세스 282)
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션]
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...

atoi, atof 함수

- ▶ atoi : 문자열 내에 포함된 숫자들을 int 값으로 반환
- ▶ atof : 문자열 내에 포함된 숫자들을 double 값으로 반환
- ▶ 문자열이 숫자로 시작하지 않는 경우 0 반환

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

int main(void)
{
    int num1 = atoi("123cde456");
    double num2 = atof("123.456cde789.012");
    int num3 = atoi("ab123cde");    // 숫자로 시작 x => 0

    printf("num1 : %d\n", num1);
    printf("num2 : %f\n", num2);
    printf("num3 : %d\n", num3);
}
```

```
num1 : 123
num2 : 123.456000
num3 : 0
```

clock 함수 (1)

- ▶ 프로그램 실행 시 내부 clock 값이 0으로 초기화됨
- ▶ 이 값은 초당 일정 횟수만큼(CLOCKS_PER_SEC, 1000) 증가함 → 두 시점 사이의 경과 시간을 알아오는 데 활용
- ▶ 예 : 총 1,000만 번 동안 0.0001을 더한 후 출력
 - 실행 시간 출력

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

```
int main(void)
{
```

```
    double result = 0;
```

```
    clock_t current_clock;
```

```
    clock_t initial_clock = clock();
```

```
    printf("초기 clock : %d\n", initial_clock);
```

```
// 덧셈 결과 저장
```

```
// 현재 clock 저장
```

```
// 초기 clock
```

clock_t는 long 타입과 동일
- typedef 문 사용 : 교재 11주차

clock 함수 (2)

```

for (int i = 1; i <= 10000000; i++)
    result = result + 0.0001;
printf("합산 결과 : %f\n", result);

current_clock = clock();           // 현재 clock
printf("현재 clock : %d\n", current_clock);

printf("실행 시간 : %f초\n",       // 실행 시간 출력
       (double)(current_clock - initial_clock) / CLOCKS_PER_SEC);
}

```

초기 clock : 0
 합산 결과 : 1000.000000
 현재 clock : 47
 실행 시간 : 0.047000초

CLOCKS_PER_SEC는 1000
 - #define으로 미리 정의되어 있음

정수 사이의 연산 → 정수 결과
 - 실수 결과를 위해서는 강제
 형변환 필요

본 자료에서 소개할 비표준 라이브러리 함수들

▶ Windows 운영체제에서 제공하는 함수들

헤더 파일	함수 프로토타입 및 기능
conio.h	<code>int _getch(void);</code> 키보드로부터 문자를 입력받음. <code>scanf("%c", &var)</code> 함수를 통한 문자 입력과는 달리 입력 즉시 문자를 읽어들이며 해당 문자가 화면에 나타나지도 않음
	<code>int _kbhit(void);</code> 키 입력 여부를 알아냄. 키가 입력된 경우 0이 아닌 값이 반환됨
Windows.h	<code>void *GetStdHandle(unsigned long nStdHandle);</code> 지정한 장치(표준 입력, 표준 출력, 표준 에러)에 대한 핸들(포인터)이 반환됨
	<code>int SetConsoleCursorPosition(void *hConsoleOutput, COORD dwCursorPosition);</code> 실행 도스창의 커서 위치를 이용시킴. 출력 내용은 현재 커서 위치에 출력되므로 임의의 위치에 데이터를 출력하고자 할 때 활용할 수 있음
	<code>int SetConsoleCursorInfo(void *hConsoleOutput, const CONSOLE_CURSOR_INFO *lpConsoleCursorInfo);</code> 커서의 크기를 설정할 수 있고 아울러 커서의 숨김 여부도 설정할 수 있음
	<code>int SetConsoleTextAttribute(void *hConsoleOutput, unsigned short wAttributes);</code> 도스창에 출력되는 문자의 속성을 설정함. 문자의 전경색, 배경색 등을 설정할 수 있음
	<code>void Sleep(unsigned long dwMilliseconds);</code> <code>dwMilliseconds</code> 에 해당하는 밀리초 만큼 프로그램의 실행이 중단됨

_getch 함수를 이용한 문자 입력

- ▶ _getch 함수 : 문자 하나 입력
 - 비버퍼형 입력 : 키 입력 즉시 해당 문자를 읽어들이
 - 입력 문자가 화면에 나타나지 않음
 - cf) scanf("%c", &ch); // 엔터키를 눌러야 됨. 입력값이 화면에 나타남
- ▶ 예제 : 입력받은 문자를 그대로 화면에 출력
 - 엔터키 → '\r'로 입력됨 → '\n'으로 출력해야 다음 줄로 이동

_getch, scanf 공통점 :
입력을 위해 대기(멈춤)

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

int main(void)
{
    int ch = _getch();           // 문자 입력

    while (ch != 'q')
    {
        if (ch == '\r')         // Enter키 입력
            printf("\n");
        else
            printf("%c", ch);    // 읽어들이 문자를 화면에 출력
        ch = _getch();
    }
}
```

Hi C Programming
_getch 연습

문자는 주로
int 형으로 처리

_getch 함수를 이용한 방향키 및 ESC 키 감지

- ▶ 방향키가 눌러졌음을 인식하고 싶다!
 - 문제점 : 방향키의 아스키 코드 값이 다른 문자의 값과 중복됨
 - 예 : UP키의 아스키 코드 값 : 72 → 'H'와 동일
 - 다행히 방향키 입력 시 해당 값이 입력되기 전에 0 또는 224의 값이 입력됨

// 1110 0000

 - → 0 또는 224가 입력되었다면 그 다음 입력 문자를 통해 방향키 인식

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

```
#define KEY_ESC    27
#define KEY_UP     (256 + 72)
#define KEY_DOWN   (256 + 80)
#define KEY_LEFT   (256 + 75)
#define KEY_RIGHT  (256 + 77)
```

#define : 문자열 상수와 동일
프로그램 내에서 KEY_ESC는 27과 동일함

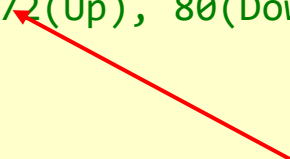
UP키는 원래 72임. 이 값에 256을 더해(아스키 범위를 벗어나는 값) UP키를 'H'와 구별하기 위해 사용

```
int GetKey(void) {
    int ch = _getch();

    if (ch == 0 || ch == 224)
        // 방향키의 경우 0 또는 224의 값이 먼저 입력됨
        ch = 256 + _getch();
    // 그 다음에 해당 방향키에 따라 72(Up), 80(Down), 75(Left), 77(Right) 값이 입력됨
    return ch;
}

int main(void) {
    int ch;

    while ((ch = GetKey()) != KEY_ESC) {
        switch (ch) {
            case KEY_RIGHT:
                printf("[Key Right] ");
                break;
            case KEY_LEFT:
                printf("[Key Left] ");
                break;
            case KEY_UP:
                printf("[Key Up] ");
                break;
            case KEY_DOWN:
                printf("[Key Down] ");
                break;
        }
    }
}
```



UP키가 눌러진 경우 UP키를 의미하는 KEY_UP(256+72)가 반환됨

_kbhit 함수를 이용한 키 입력 감지

- ▶ 어떤 키인지에 관계없이 키 입력 여부 감지
 - 키 입력이 없으면 0, 키가 입력되었다면 0이 아닌 값 반환
- ▶ 예제 : 정수값이 1씩 100,000번 증가할 때마다 “안녕!” 출력

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

int main(void)
{
    int count = 0;

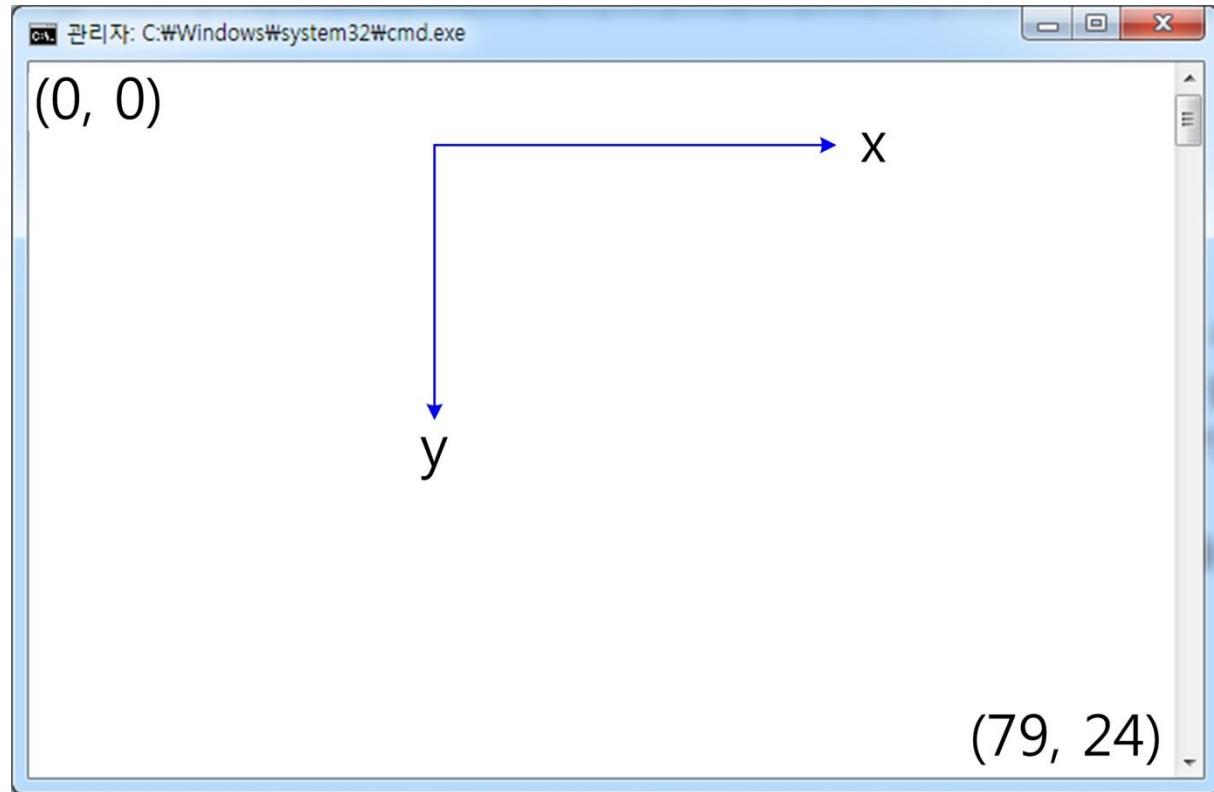
    while (1) {
        count++;
        if (count == 100000) {
            printf("안녕!");
            count = 0;
        }

        if (_kbhit()) { // 키 입력이 있다면 0이 아님
            if (_getch() == 'q')
                break;
        }
    }
}
```

- 'q' 문자 입력 시 종료
- 키 입력을 위해 count 증가를 멈추면 안됨

도스창의 좌표 체계

- ▶ 도스창의 기본 크기
 - 알파벳 문자 단위 : 가로 80, 세로 25
- ▶ 도스창의 좌표 체계
 - 가로 x, 세로 y
 - (0, 0) ~ (79, 24)



커서 이동 (1)

- ▶ 출력 데이터는 커서가 위치한 곳에 출력됨
 - 출력 후 커서는 한 칸씩 오른쪽으로 이동
 - 커서를 이동시킬 수 있다면 원하는 곳에 데이터 출력 가능
- ▶ 커서 이동 관련 함수
 - <Windows.h>
 - GetStdHandle : 표준 입출력 핸들 반환
 - GetStdHandle(STD_OUTPUT_HANDLE) : 표준 출력 장치 핸들
 - SetConsoleCursorPosition : 커서 이동
 - Coord pos = { 5, 7 }; // Coord는 구조체 : 교재 11주차
 - SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), pos);

사용 방법만 알면 얼마든지 활용 가능!

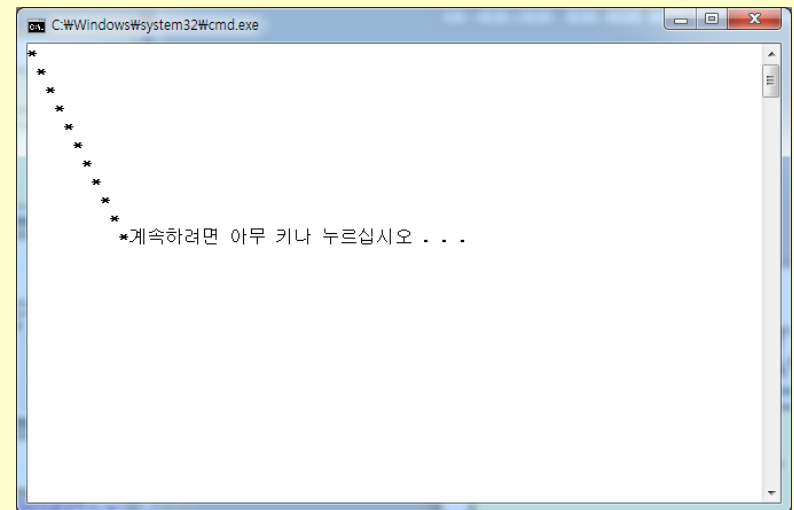
커서 이동 (2)

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

```
void GotoXY(int x, int y)
{
    // COORD 구조체 변수를 통해 이동할 위치 설정
    COORD pos = { x, y };
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), pos);
}
```

```
int main(void)
{
    for (int i = 0; i <= 10; i++)
    {
        GotoXY(i, i);
        printf("*");
    }
}
```

```
COORD pos;
pos.X = x;
pos.Y = y;
pos = { x, y }; //불가능
```



커서 크기 등 커서 속성 설정 (1)

▶ SetConsoleCursorInfo 함수

- 커서 크기 설정, 커서 숨기기/보이기
- 두 번째 매개변수로 `CONSOLE_CURSOR_INFO` 구조체 변수의 주소 전달

- ```
struct CONSOLE_CURSOR_INFO {
 DWORD dwSize;
 BOOL bVisible;
};
```

## ▶ 예제 : 2개의 값 입력

- 첫 번째 : 커서 크기 100(꼭 찬 사각형, 기본 25)
- 두 번째 : 커서 숨기기

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

```
int main(void)
{
```

```
 int num1;
 int num2;
```

```
 // 커서 크기(100), 커서 출력 여부(TRUE)
 CONSOLE_CURSOR_INFO ci = { 100, TRUE };
 // 커서 정보 설정
```

```
 SetConsoleCursorInfo(GetStdHandle(STD_OUTPUT_HANDLE), &ci);
```

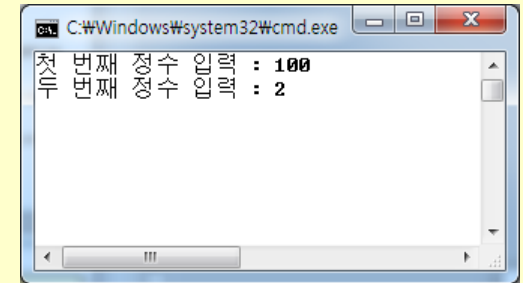
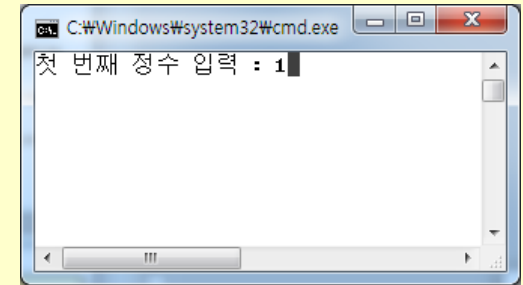
```
 printf("첫 번째 정수 입력 : ");
 scanf("%d", &num1);
```

```
 ci.bVisible = FALSE; // 커서 숨김. 구조체(11주차)
 // 커서 정보 설정
```

```
 SetConsoleCursorInfo(GetStdHandle(STD_OUTPUT_HANDLE), &ci);
```

```
 printf("두 번째 정수 입력 : ");
 scanf("%d", &num2);
```

```
}
```



# 출력 문자의 색깔 등 문자 속성 지정 (1)

- ▶ **SetConsoleTextAttribute** 함수
  - 출력 문자의 전경색(문자의 색)과 배경색 변경
  - 기본색 : 흰색 문자, 검정색 배경
- ▶ **SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 색)**
  - FOREGROUND\_BLUE, FOREGROUND\_GREEN, FOREGROUND\_RED
  - BACKGROUND\_BLUE, ..... // 포어그라운드와 동일
  - FOREGROUND\_INTENSITY, BACKGROUND\_INTENSITY // 연한색으로 만들
  - 검정색 : 색 지정 생략 (0을 지정)
  - 흰색 : 세 가지 색을 비트 단위 OR(|)로 연결

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

int main(void)
{
 // 파란색 문자, 검정색 배경
 SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), FOREGROUND_BLUE);
 printf("Hello! C World.\n");

 // 검정색 문자, 초록색 배경
 SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), BACKGROUND_GREEN);
 printf("Programming is fun.\n");

 // 빨간색 문자, 파란색 배경
 SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
 FOREGROUND_RED | BACKGROUND_BLUE);
 printf("Nice to meet you!\n");

 // 빨간색 문자, 흰색 배경
 SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
 FOREGROUND_RED |
 BACKGROUND_RED |
 BACKGROUND_GREEN |
 BACKGROUND_BLUE |
 BACKGROUND_INTENSITY);
 printf("Good bye.\n");
}
```

# 실행 시간 지연 : Sleep 함수

- ▶ Sleep(1000) : 1000밀리초(1초) 동안 실행 멈춤
- ▶ 예제 : 1초 간격으로 1부터 10까지 출력

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

int main(void)
{
 int i;

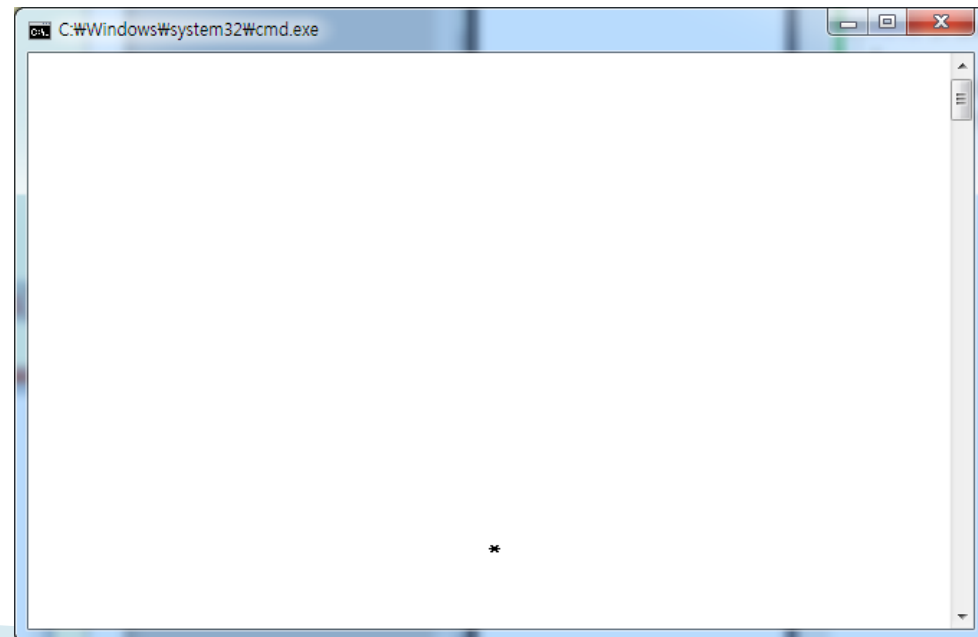
 for (i = 1; i <= 10; i++)
 {
 printf("%d ", i);
 Sleep(1000); // 1000밀리초(1초) 동안 실행 멈춤
 }
}
```

1초 동안 다른 작업을 할 수 없음.  
clock 함수를 사용하여 구현한다면?



# 종합 예제 (1)

- ▶ 프로그램이 시작되면 (x, 0) 위치에 '\*' 문자 하나가 나타난다. x의 값은 rand 함수를 사용하여 (0 ~ 79) 사이의 난수로 설정한다. 그리고 '\*' 문자는 0.5초 간격으로 한 칸씩 아래로 떨어진다. 즉, y 좌표의 값이 1씩 증가하는 것이다. 그런데 방향키 중 왼쪽키 또는 오른쪽 키를 누르면 해당 방향으로 한 칸만큼 이동해야 한다. '\*' 문자가 (x, 24) 위치에 도달하면 프로그램은 종료된다.
- ▶ 문자를 이동하려면(=이동하는 것처럼 보이려면)
  - 기존 위치로 이동하여 삭제(=공백 문자 출력)
  - 새 위치로 이동하여 출력
- ▶ 사용 함수들
  - srand, rand
  - clock
  - GotoXY
  - GetKey



# 종합 예제 (2)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>
#include <Windows.h>

#define KEY_LEFT (256 + 75)
#define KEY_RIGHT (256 + 77)

int GetKey(void);
double GetElapsedTime(clock_t initial_clock, clock_t current_clock);
void GotoXY(int x, int y);
void Erase(int x, int y); // (x, y)로 이동하여 공백 문자 출력
void Draw(int x, int y); // (x, y)로 이동하여 '*' 문자 출력
```

```
int main(void)
{
 srand(time(NULL));

 int x = rand() % 80;
 int y = 0;
 Draw(x, y);

 clock_t initial_clock = clock();

 while (1)
 {
 clock_t current_clock = clock();

 if (GetElapsedTime(initial_clock, current_clock) > 0.5)
 {
 // 0.5초 경과
 Erase(x, y);
 y++;
 Draw(x, y);

 if (y == 24)
 break;

 initial_clock = current_clock; // 기준 시각 재설정
 }
 }
}
```

# 종합 예제 (4)

```
 if (_kbhit())
 {
 int key = GetKey();

 if (key == KEY_LEFT)
 {
 Erase(x, y);
 x--;
 Draw(x, y);
 }
 else if (key == KEY_RIGHT)
 {
 Erase(x, y);
 x++;
 Draw(x, y);
 }
 }
} // end of while
}
```

# 종합 예제 (5)

```
int GetKey(void)
{
 int ch = _getch();

 if (ch == 0 || ch == 224)
 // 방향키의 경우 0 또는 224의 값이 먼저 입력됨
 ch = 256 + _getch();
 // 그 다음에 해당 방향키에 따라 72(Up),
 // 80(Down), 75(Left), 77(Right) 값이 입력됨
 return ch;
}

double GetElapsedTime(clock_t initial_clock, clock_t current_clock)
{
 return (double)(current_clock - initial_clock) / CLOCKS_PER_SEC;
}
```

# 종합 예제 (5)

```
void GotoXY(int x, int y)
{
 // COORD 구조체 변수를 통해 이동할 위치 설정
 COORD pos = { x, y };
 SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), pos);
}

void Erase(int x, int y)
{
 GotoXY(x, y);
 printf(" ");
}

void Draw(int x, int y)
{
 GotoXY(x, y);
 printf("*");
}
```

# 이번 장에서 배운 것

- 표준 C에서는 다양한 라이브러리 함수들을 제공하고 있으며, 운영 체제에서도 많은 라이브러리 함수들을 제공하고 있다.
- sin, log 등 수치 계산을 위한 라이브러리 함수들이 있다.
- rand 함수를 통해 난수를 발생시킬 수 있다.
- clock 함수를 통해 두 지점 사이의 경과 시간을 알아낼 수 있다.
- \_getch 함수는 비버퍼형 문자 입력 함수이다.
- \_kbhit 함수를 통해 키 입력 여부를 알아낼 수 있다.
- SetConsoleCursorPosition 함수를 사용하여 커서를 임의의 위치로 이동할 수 있으며, rand, \_getch, \_kbhit, clock 함수 등을 함께 사용하여 도스창 기반의 게임 등 동적 프로그램을 작성할 수 있다.
- 표준, 비표준 라이브러리에는 매우 많은 함수들이 있다. 도움말 등을 통해 필요할 때 찾아서 사용할 수 있는 능력을 향상시킬 필요가 있다.

