출력(Output)

2020년 12월 10일 목요일 오후 2:31

출력(Output)

- 프로그램에서 프로그램 외부로 데이터를 보여주는 행위
- 프로그램이 어떤 데이터를 출력할지 아니까 괴상한 데이터들이 없다
  - 입력에 경우 괴상한 데이터들이 입력될 수도 있다

2 지정(formatted) 출력

- C에서 출력을 논할 때 가장 기본이 되는 함수
- 세 가지 종류
  - printf( ): 콘솔창(stdout)에 출력
  - fprintf( ): 스트림에 출력
    - 첫번째 매개변수로 stdout을 넣어주면 printf와 같다
  - sprintf( ): 문자열에 출력
- fprintf( )와 sprintf( )는 printf( )하고 작동법 동일
  - 첫 번째 매개변수로 '출력할 곳'을 넣어주는게 유일한 차이점

printf( )의 첫 번째 매개변수는 문자열

  - C#은 Console.WriteLine( )에 인자로 문자열을 넣든 int를 넣든 출력해줬음
  - C는 첫 번째 인자로 문자열만 받음
    - C에는 함수 오버로딩이 없기 때문

C#

```
int score = 10;
char ch = 'j';
float pi = 3.14f;
string name = "Caty";

Console.WriteLine("Hello POCU");
Console.WriteLine(score);
Console.WriteLine(ch);
Console.WriteLine(pi);
Console.WriteLine(name);
```

C

```
int score = 0;
char ch = 'j';
float pi = 3.14f;
const char* name = "Caty";

printf("Hello POCU"); /* 컴파일 */
printf(score);        /* 컴파일 오류 */
printf(ch);           /* 컴파일 오류 */
printf(pi);           /* 컴파일 오류 */
printf(name);         /* 컴파일 */
```

서식 문자열(format string)

```
printf("Hello POCU");                                  /* 일반 문자열 */
printf("%d", score);                                   /* 서식 문자열: 정수 출력 */
printf("%c", ch);                                      /* 서식 문자열: 문자 출력 */
printf("%f", pi);                                      /* 서식 문자열: 부동소수점 출력 */
printf("%s", name);                                    /* 서식 문자열: 문자열 출력 */
printf("Hello, %s\nYour score is %d\n", name, score); /* 서식 문자열: 혼합 출력 */
```

- printf( )는 일반 문자열 혹은 서식 문자열을 매개변수로 받음
- 서식 문자열
  - %로 시작하는 문자열
  - 소수점 이하 자리수, 자릿수 정렬, 어떤 데이터(숫자, 문자)를 출력할지 등을 알려주는 문자열
  - 서식 문자열에는 하나 이상의 데이터가 들어갈 수 있음
    - 서식 지정자의 순서와 동일한 순서로 데이터들을 printf( )의 추가 매개변수로 전달
      - %s <- name %d <- score
- 일반적인 서식 문자열 형식

```
%[플래그][너비][.숫자 정밀도 | .문자열 최소/최대 출력 개수][길이]서식 지정자
```

  - 선택사항으로 넣을 것
    1. 플래그
    2. 너비
    3. 정밀도
    4. 길이
    5. 서식 지정자
       - 서식 지정자는 필수로 넣어야 함
  - 반드시 순서를 지켜서 작성해야 함
- 서식 지정자(format specifier)

%	'%'를 출력	printf("%%\n");
c	문자(char) 출력	printf("%c\n", 'D');
s	문자열(char[]) 출력	printf("%s\n", "Lulu");

- 서식 문자라고도 함
- '%"는 서식 문자열에서 사용 중이므로 출력하려면 어쩔수 없이 반복
  - 이스케이프 문자 때문에 ₩출력하려면 ₩₩ 를 넣었던 것과 마찬가지
- %s 쓸 바엔 그냥 문자열을 곧바로 printf( ) 해도 됨
  - 단, 문자열 두 개 합치거나 문자열 + 숫자 합쳐서 출력할 때 유용

```
printf(name); /* const char* name = "Lulu"; */
printf("%s", name);
printf("Hello, %s\nYour score is %d\n", name, score);
```

d	부호있는 정수 출력	printf("%d\n", -10);
u	부호없는 정수 출력	printf("%u\n", 10);
o	부호없는 정수를 8진수로 출력 숫자 앞에 '0'은 안 붙여줌	printf("%o\n", 10);
x	부호없는 정수를 16진수(소문자)로 출력 숫자 앞에 '0x'는 안 붙여줌	printf("%x\n", 10);
X	부호없는 정수를 16진수(대문자)로 출력 숫자 앞에 '0X'는 안 붙여줌	printf("%X\n", 10);

- %u에 부호있는 수를 넣을 경우, 해당 수의 비트패턴에 해당하는 부호없는 수가 출력

```
printf("%u\n", -10); /* 4294967286 출력*/
```

- %X는 있는데 %O는 없다
  - 8진수는 숫자로만 이루어져 있어서

f	부동소수점 출력	printf("%f\n", 3.14);
e/E	부동소수점을 지수표기법으로 출력	printf("%e\n", 3.14); printf("%E\n", 3.14);
p	포인터값을 출력	printf("%p\n", (void*)name);

- %p 는 주소를 출력하는데 void*만 받음
  - 모든 주소는 어차피 길이가 같으니 어떤 포인터를 void*로 캐스팅해도 안전

② 서식 문자열

② 너비

정수	출력 너비 (너비보다 작으면 공백)	printf("%5d\n", number);

- 기본적으로 오른쪽 정렬이 됨

```
printf("%d\n", number);   1 0
printf("%5d\n", 10);          1 0
```

- 그 밖에 너비 옵션은 링크에서 찾아볼 것
  - https://en.cppreference.com/w/c/io/fprintf

② 플래그

-	왼쪽 정렬	printf("%-5d\n", number);
0	빈 공백을 0으로 채워줌	printf("%05d\n", number);
+	항상 부호(+, -)를 표시	printf("%+5d\n", number);
공백	양수인 경우에도 부호칸을 비워둠	printf("% d\n", number);

- '-'를 넣지 않으면
  - 기본으로 오른쪽 정렬
- ' '공백과 '+'를 같이 넣으면 서로 상충됨
  - '+'가 우선순위라서 ' '가 무시된다
- '+' 안 넣으면
  - 음수 기호만 출력
- '0'와 '-'를 같이 넣으면 서로 상충
  - '0'가 무시됨

```
printf("%-5d\n", 10);   1 0
printf("%05d\n", 10);   0 0 0 1 0
printf("%-05d\n", 10);  1 0
printf("%+d\n", 10);    + 1 0
printf("%+5d\n", 10);       + 1 0
printf("%+05d\n", 10);  + 0 0 1 0
printf("% d\n", 10);      1 0
printf("% 5d\n", 10);         1 0
printf("% 05d\n", 10);      0 0 1 0
printf("%+ 05d\n", 10);   + 0 0 1 0
```

② 플래그2

#	o	부호 없는 정수를 8진수로 출력 숫자 앞에 항상 '0'이 붙음	printf("%#o\n", number);
	x	부호 없는 정수를 16진수로 출력(소문자) 숫자 앞에 항상 '0x'가 붙음	printf("%#x\n", number);
	X	부호 없는 정수를 16진로 출력(대문자) 숫자 앞에 항상 '0X'가 붙음	printf("%#X\n", number);

```
printf("%#o\n", 10);          0 1 2
printf("%#x\n", 10);          0 x a
printf("%#X\n", 10);          0 X A
printf("% #o\n", 10);         0 1 2
printf("%5#o\n", 10);         #   o
printf("%+#o\n", 10);         0 1 2
printf("%-#o\n", 10);         0 1 2
```

컴파일 경고 발생

- x나 X말고 다른 곳에 붙일 수도 있음
  - 링크 참고: https://en.cppreference.com/w/c/io/fprintf
- #x/X/o 를 쓸 때는 다른 플래그나 서식지정자 'd'를 붙이지 않음

정밀도 (1)

- 서식 지정자 'f'와 함께 사용
  - 최소너비.소수점 아랫자리 수
  - (소수점 포함) 원래 숫자의 너비보다 최소 너비가 크면 공백으로 채움
  - (소수점 포함하지 않음) 원래 숫자의 소수점 아랫자리 수보다 소수점 아랫자리수가 크면 0으로 채움
  - 기본 소수점 아랫자리 수: 6

```
printf("%f\n", 3.14f);      3 . 1 4 0 0 0 0
printf("%3.3f\n", 3.14f);   3 . 1 4 0
printf("%6.3f\n", 3.14f);     3 . 1 4 0
```

② 정밀도 (2)

- 서식 지정자 's'와 함께 사용
  - 최소 너비.최대 너비
  - 출력할 문자열의 길이가 최소 너비보다 작으면 왼쪽을 공백으로 채움
  - 출력할 문자열의 길이가 최대 너비보다 크면 자름

```
printf("%5.7s\n", "Hi");          H i
printf("%5.7s\n", "Hello")        H e l l o
printf("%5.7s\n", "Hello, POCU"); H e l l o ,
```

② 길이 수정자(length modifier)

- 인자의 바이트 크기를 지정해준다
- 몇 가지 있는데 'l' 과 'L'만 그나마 유용할 지도
  - 하지만 최근 플랫폼에선 별 의미가 없음.
    int == long int, double == long double인 경우가 보통
  - 설사 위의 경우가 아니더라도 long double은 잘 안 씀

서식 문자열이 필요한 이유

- 일단 C에는 오버로딩이 없어 printf(int), printf(char) 불가능
- 그리고 임시 문자열 등을 자동으로 생성 안 해줌
  - C#처럼 매개변수로 문자열 + 문자열 + 문자열 이런식으로 안 됨

  - strcat( )을 이용해서 프로그래머가 직접 임시 문자열을 관리할 수 있음

- 즉, 서식 문자열은 추가 메모리 할당 없이 있는 자료형을 출력
  스트림에 문자들로 출력해줌
- C#의 문자열 포맷과 비슷하다
  - C에서 내려온 스타일

- fprintf( )도 똑같음
  - 단 여기에는 스트림을 사용

  - 스트림이 뭘까?
    보통 프로그램이 실행할 때 기본적으로 3개의 스트림을 줌
    - stdout (콘솔 출력)
    - stdin (콘솔 입력)
    - stderr (콘솔 출력임. 하지만 오류 메시지를 출력하는 스트림)
    - 이 3개 스트림을 표준 스트림(standard stream)이라 함
- stdout
  - 우리가 계속 봐왔던 콘솔 스트림
  - stdout은 보통 라인 버퍼링(line buffering)을 사용
  - 버퍼링
    - 출력할 내용이 있어도 곧바로 출력하지 않고 쌓아 둠
    - 어느정도 버퍼가 차면 그제서야 출력
  - 라인 버퍼링: 다음과 같은 경우에 버퍼를 비움
    1. 버퍼가 꽉 차거나
    2. 버퍼에 '₩n'가 들어올 때
  - 강제로 버퍼를 비우고 싶다면 fflush(stdout); 을 호출하면 됨

② 버퍼링의 종류

- 풀 버퍼링(full buffering)
  - 버퍼가 가득 차면 비움. 라인 버퍼링과 마찬가지로 fflush( )로 강제로 비울 수 있음
  - 코드 슬라이드에서 본 거
- 라인 버퍼링(line buffering)
  - 버퍼가 꽉 차거나, 버퍼에 '₩n'가 들어오면 버퍼를 비움
- 버퍼링 없음(no buffering)
  - 버퍼를 사용하지 않음
- 표준에서 stdout, stderr, stdin의 버퍼링 종류를 지정하지 않음
  - 구현따라 다를 수 있음
  - 일반적으로 stdout은 라인 버퍼링을 사용

- 다른 스트림은 뭐가 있을까?
  - 모든 출력 스트림에 fprintf( ) 사용 가능
    - 첫 인자로 들어가는 스트림만 달라지고 나머지 매개변수는 그대로
  - 파일 스트림
    - C에 있음.
  - 문자열 스트림
    - C에는 없음
    - 그 대신 sprintf( ) 가 있다

- sprintf( )

```
int sprintf(char* buffer, const char* format, ...);
```

  - 어디에 출력?
    - char 배열에
  - 정말 많이 쓰는 함수
    - C++에 string 클래스가 있는데도 이걸 쓰기도 한다
    - 이유는 속도가 빠르기 때문
  - 다만, 프로그래머가 충분히 큰 버퍼를 잡아주지 않으면 위험

```
char buffer[100];
int score = 100;
const char* name = "Rachel";

sprintf(buffer, "%s: %d", name, score);

printf("%s\n", buffer);
```

```
Rachel: 100
```

```c
sprintf(buffer, "%s: %d", name, score);

printf("%s\n", buffer);
```

```
Rachel: 100
```

- sprintf( )는 안전할까?
  - 안전하지 않음
    - strcpy( ), strcat( )과 같은 이유
  - snprintf( )함수가 있음
    - C99에
  - 표준은 아니지만 컴파일러마다 다르게 제공하는 함수가 있음
    - 마이크로소프트 비주얼 c의 _snprintf( )
      - 앞에 _ 가 붙인 함수는 보통 표준이 아닌경우가 많았음
      - 최근에는 아닌 경우도 있음
      - snprintf( )와 _snprintf( )가 작동하는 방법이 달라서 호환 문제가 있었다

기타 출력 함수

- putchar( )

```c
int puts(const char* str);
int fputs(const char* str, FILE* stream);
```

  - 문자열을 stdout에 출력
  - 마지막에 줄도 바꿔줌: '₩n'
  - fputs(str, stdout)과 같음
- putchar( )

```c
int putchar(int ch);
int fputc(int ch, FILE* stream);
```

  - 문자를 stdout에 출력
  - fputc(ch, stdout)하고 같음