Lecture 04

제어 흐름 및 입출력

리뷰

- 복합문/블록
  - 중괄호로 묶인 여러 개의 선언문이나 문장
  - 블록이 중첩될 수 있음
  - 어떤 블록에서 선언된 변수가 그 블록과 내부의 블록에서만 통용됨

```
{
    int x=0;
    {
        int y=10; // x 및 y를 모두 사용 가능
    }
    // x만 사용 가능, y를 사용 못함
}
```

리뷰

- **if else** 문

```
if （수식 1）
    블록 1
else if （수식 2）
    블록 2
else if （수식 3）
    블록 3
else
    블록 4
```

- 수식은 **순서에 의해** 계산되며, 수식이 참일 경우 바로 그 밑에 블록이 수행되며, 그렇지 않을 경우 다음 수식으로 넘어가는 과정을 반복함
- 수식이 참일 경우가 하나도 없을 때 마지막 **else** 밑에 블록이 수행됨

리뷰

- **switch** 문

```
switch (수식) {
    case 상수:
        문장
        break;
    case 상수:
        문장
        break;
    default:
        문장
        break;
}
```

- 수식은 계산되며, 그 결과가 어떤 상수와 일치하면 바로 뒤의 문장은 수행됨
- **case** 다음에 오는 상수들은 반드시 달라야 하고, 정수 값을 가져야 함
- **default**는 각 경우가 만족되는 것이 없을 때 수행되고, 생략해도 됨

리뷰

- 순환문
  - **while** 문

    수식을 먼저 계산하며, 그 결과에 따라 **while** 문 뒤에 오는 블록이 수행될지를 결정함
  - **do while** 문

    블록이 수행된 후 수식이 계산되며, 그 결과에 따라 블록이 다시 수행될지를 결정함
  - **for** 문

    초기화, 루프 종료 조건 검토, 반복의 3가지를 포함함

제어 흐름

- **goto** 문이 같은 함수 내에서 다른 위치로 무조건으로 이동할 수 있게 함
- 이동하려는 위치를 레이블로 지정할 수 있음

```
start:
{
    if (수식)
        goto outside;
    …
    goto start;
}
outside:
…
```

제어 흐름

- **goto** 문을 많이 사용하면 코드 구조를 파악하기 어려움
  - 이를 스파게티(spaghetti) 코드라고 부름
- 일반적으로 **goto** 문을 사용하지 않으면 좋으나, **goto** 문이 유용할 때도 있음
  - 중첩된 루프에서 한 번에 모든 루프를 벗어날 수 있음

goto 사용

```
for (…) {
    for (…) {
        if (에러 발생)
            goto error;
    }
}
error:
```

goto 사용 안 함

```
int flag=1;
for (…) {
    for (…) {
        if (에러 발생) {
            flag=0;
            break;
        }
    }
    if (!flag) break;
}
```

표준 입력과 출력

- <stdio.h>란 표준 라이브러리의 함수를 사용하여 텍스트의 입력과 출력의 간단한 동작을 실행할 수 있음
- 문자 stream의 여러 개의 행으로 구성되며, 각 행의 끝에는 행바꿈('\n') 문자가 있음
  - 표준 라이브러리의 함수는 행바꿈('\n') 문자 또는 리턴키('\r') 등에 적절히 대응하는 동작을 해야 함

표준 입력과 출력

- `int putchar(int)`
  - `putchar(c)`는 문자 `c`를 표준 출력인 스크린에 보냄
  - 출력된 문자를 리턴하는데, 에러 발생 시에는 `EOF`를 리턴함
  - `prog >outfile`
    - 출력을 표준 출력 대신 `outfile`에 보냄
  - `prog1 | prog2`
    - `prog2`의 표준 입력에 `prog1`의 표준 출력을 집어넣음
- `int getchar()`
  - 다음의 입력 문자를 넘겨줌
  - 파일의 끝을 만나거나 에로 발생 시 `EOF`를 넘겨줌
  - `prog <infile`
    - `infile`로부터 `prog`가 문자를 읽어들임
  - `prog1 | prog2`
    - `prog1`의 표준 출력을 `prog2`의 표준 입력으로 되게 함

표준 입력과 출력

- 다음의 코드가 무슨 기능을 수행하는가?

```
int main() {
   char c;
   while ((c=getchar()) != EOF) {
      if (c>='A' && c<='Z')
         c = c-'A'+'a';
      putchar(c);
   }
   return 0;
}
```

형식화된 출력: printf


```
int printf(char *format,  arg1, arg2, …)
```

- 출력형식 제어 문자열(format)에 따라
  - 보통 문자 또는 문자열 출력

    ```
    printf("hello world!")
    ```

  - arg1, arg2, … 등의 매개변수를 format에서 지정된 형태로 출력

    ```
    printf("integer: %d", 20)
    ```

- 일반적인 출력형식을 정해주는 문자열

```
%[flags][width][.precision][modifier]<type>
```

형식화된 출력: printf


```
%[flags][width][.precision][modifier]<type>
```

type	의미	예
d, i	정수	printf("%d",10); // 10 출력
x, X	정수(16진수)	printf("%x",10); // 0xa 출력
u	무부호형 정수	printf("%u",10); // 10 출력
c	문자	printf("%c",'W'); // W 출력
s	문자열	printf("%s","hello"); // hello 출력
f	float	printf("%f",9.23); // 9.23 출력
g,G	double	printf("%g",9.23); // 9.23 출력
e, E	float(exp)	1e3, 1.2E3, 1E-3
%	퍼센트	printf("%d %%",10); // 10% 출력

형식화된 출력: printf

%[flags]**[width]**[.precision][modifier]<type>

포맷	출력
printf("%d",10);	10
printf("%4d",10);	[space][space]10
printf("%s","hello");	hello
printf("%7s","hello");	[space][space]hello

형식화된 출력: printf

%**[flags]**[width][.precision][modifier]<type>

포맷	출력
printf("%d,%+d,%+d",10,10,-10);	10,+10,-10
printf("%04d",10);	0010
printf("%7s","hello");	[space][space]hello
printf("%-7s","hello");	hello[space][space]

형식화된 출력: printf

%[flags][width]**[.precision]**[modifier]<type>

포맷	출력
printf("%.2f,%.0f",1.254,1.254);	1.25,1
printf("%.2e,%.0e",1.254,100.00);	1.25e+00,1e+02
printf("%.4s","hello");	hell
printf("%.1s","hello");	h

형식화된 출력: printf

%[flags][width][.precision]**[modifier]**<type>

modifier	의미
h	short라는 의미이며, i, d, o, u, x와 사용함
l	long이라는 의미이며, i, d, o, u, x와 사용함
L	double이라는 의미이며, e, f, g와 사용함

문자열

- 문자열 또는 스트링은 문자의 배열이며 끝이는 0 또는 '\0' 임

```
char str[] = "hello"; // 컴파일러가 크기를 정함
char str[10] = "hello"; // 충분한 크기를 할당해야 함
char str[] = {'h','e','l','l','o',0};
```

- 문자열 속에 "을 원하면 '\"' 특수 문자를 사용하면 됨

```
char str[] = "\"hello\"";
printf("%s", str); // "hello" 출력
```

문자열

- 문자열 비교
  - <string.h> 헤더파일에서 문자열 2개를 사전 순으로 비교해주는 strcmp 함수는 제공됨

```
int strcmp(char s[], char t[])
```

    - 문자열이 같으면 0을 리턴함
    - s가 t보다 사전 순으로 앞서면 음수를 리턴함
    - s가 t보다 사전 순으로 뒤에 있으면 양수를 리턴함

```
strcmp("A","a"); // 음수 리턴
strcmp("ironman","batman"); // 양수 리턴
strcmp("aA","aA"); // 0 리턴
strcmp("aA","a"); // 양수 리턴
```

형식화된 입력: scanf


```
int scanf(char *format, …)
```

- printf와 사용방법이 유사한 입력함수임
- 표준 입력으로부터 문자를 읽어 들여서 format의 변환형식에 따라 그 들을 해석하고 매개변수에 결과를 저장함
- 매개변수는 입력되는 문자가 위치해야 하는 곳을 가리키는 **포인터**여야 함
- 리턴값
  - 성공리에 대응된 것들의 개수와 할당된 입력 항목의 개수를 리턴함
  - 파일의 끝에 가면 EOF를 리턴함
  - 입력 문자가 변환형식 지정 문자열과 대응되지 않는 입력이 들어왔을 때 0을 리턴함

형식화된 입력: scanf


```
int scanf(char *format, …)
```

printf	scanf
printf("%d",x);	scanf("%d",&x);
printf("%10d",x);	scanf("%d",&x);
printf("%f",y);	scanf("%f",&y);
printf("%s",str);	scanf("%s",str);
printf("%s",str);	scanf("%20s",str);
printf("%s %s",a,b);	scanf("%20s %20s",a,b);

* 문자열 입력 경우 & 연산자가 필요 없음

형식화된 입력: scanf

- 예,

```
int day, year;
char month[20];

scanf("%d %s %d", &day, month, &year);
```

스트링 입력/출력

- 표준 입력/출력 대신 문자열로부터 문자를 읽어 들이거나 출력할 수 있음

- `int sprint(char str[], char format[], arg1, arg2, …)`
  - `printf`가 행하는 같은 변환을 행하나 출력을 문자열(`str`)에 저장함
  - 매개변수를 `printf`와 같은 방법으로 형식화시킴
  - `str`에 저장한 문자의 개수를 리턴하거나 에로 발생 시 음수를 리턴함

- `int sscanf(char str[], char format[], arg1, arg2, …)`
  - `scanf`가 행하는 같은 변환을 행하나 입력을 표준 입력 대신 문자열(`str`)로부터 읽어 들임
  - 매개변수를 `scanf`와 같은 방법으로 형식화시킴
  - `str`로부터 읽어 들인 입력 항목의 개수를 리턴하거나 에로 발생 시 음수를 리턴함

파일 액세스

- <stdio.h> 표준 라이브러리에서 FILE이라는 구조체 정의(structure definition)가 있음
  - 파일에 관한 정보를 포함함
  - 구조체에는 버퍼(buffer)의 위치와 버퍼 내에서의 문자의 위치 또한 파일이 읽기 위한 것인지 쓰기 위한 것인지, 에러나 EOF를 만나지 않았는지 등의 정보가 들어 있음
- 읽거나 쓰기 전에 파일을 열어야 함

```
FILE *fopen(char name[], char mode[])
```

  - 첫째 매개변수는 파일의 이름이며, 둘째 매개변수는 모드(r – 입력, w – 출력, a – 첨가, b – 2진수 파일 첨가)임
  - 성공리에 FILE을 가리키는 포인터를 리턴하고, 에러 발생 시 NULL을 리턴함

파일 액세스

- C 프로그램이 처음 수행될 때, 다음의 3개 파일이 자동 열려 있고, 포인터가 배정됨
  - `stdin` 표준입력 포인터
  - `stdout` 표준출력 포인터
  - `stderr` 표준에러출력 포인터
- `stdin, stdout, stderr`는 `FILE *` 형으로 되어 있으며, `<stdio.h>`에서 선언됨
- `fopen`의 역기능을 행하는 함수

```
int fclose(FILE *fp)
```

  - 파일을 닫고, `FILE` 포인터를 재사용할 수 있음
  - C 프로그램이 정상적으로 끝났을 때 `fclose`는 호출되고 각자의 열린 파일에 대해 자동적으로 닫힘

파일 입력

- `int getc(FILE *fp)`
  - `fp`에 의해 지정된 파일로부터 입력을 한 번 호출에 한 문자씩 리턴함
  - 파일의 끝이거나 에러가 발생하면 `EOF`를 리턴함
- `getchar`는 단순히 표준입력으로부터 문자를 읽어 들임

```
#define getchar() getc(stdin)
```

- `char[] fgets(char line[], int maxlen, FILE *fp)`
  - 다음 입력행(`'\n'`을 포함한)을 파일 `fp`로부터 문자열인 `line`에 읽어 들임
  - 최대 `maxlen - 1` 문자를 읽을 수 있음
  - `line`을 리턴함
  - 파일의 끝이거나 에러를 만날 때 `NULL`을 리턴함

파일 출력

- `int putc(int c, FILE *fp)`
  - 문자 `c`를 `fp` 파일로 출력함
  - 출력된 문자를 리턴하거나 에러가 발생할 때 `EOF`를 리턴함
- `putchar`는 단순히 표준출력으로 문자를 보냄

```
#define putchar(c) putc((c), stdout)
```

- `int fputs(char line[], FILE *fp)`
  - `'\0'`을 포함하지 않는 문자열을 파일에 출력함
  - 성공리에는 `0`을 리턴하고, 에러 발생 시에는 `EOF`를 리턴함

fscanf 및 fprintf

```
int fscanf(FILE *fp, char format[], arg1, arg2, …)
int fprintf(FILE *fp, char format[], arg1, arg2, …)
```

- 첫째 매개변수가 입력과 출력을 나타내는 파일 포인터를 제외하면 `scanf`나 `printf`와 동일함