5장 같은 일 한곳에서 처리하기 : 함수

1. 이 장에서 만드는 프로그램
2. 함수란
3. 함수의 종류
4. 사칙연산 함수 만들기
5. 프로젝트: 구구단을 외자 게임

1. 이 장에서 만드는 프로그램

1단계부터 3단계까지 구구단을 외자 게임

2. 함수란

- 값을 입력받아 어떤 작업을 한 후 그 결과를 반환하는 것(반환하지 않을 수도 있음)
- printf()함수, rand()함수는 C표준 라이브러리에서 기본으로 제공하는 함수는 사용법만 익히면 쉽게 사용할 수 있다(stdio.h 해더파일에 포함)
- 사용자가 원하는 기능만 가진 함수를 사용하려면 사용자정의 함수를 만들어야 한다.

2. 함수란

(1) 수 누적하기

실행결과

```
1부터 100까지의 누적합은 5050입니다
10부터 100까지의 누적합은 5005입니다
```

```
int main(void){
    int a = 1, b=100;
    int tot = 0;
    for(int i=a ; i<=b ; i++){
        tot += i;
    }
    printf("%d부터 %d까지의 누적합은 %d입니다\n", a, b, tot);
    a = 10; b=100;
    tot = 0;
    for(int i=a ; i<=b ; i++){
        tot += i;
    }
    printf("%d부터 %d까지의 누적합은 %d입니다\n", a, b, tot);
}
```

2. 함수란

(2) 함수로 누적하기

- 함수 선언
  - 컴파일러에 '이런 함수를 쓸 거야'라고 알려 주는 것
  - main() 함수 위쪽에 작성
- 함수 정의
  - 함수가 어떤 일을 할지 정의하는 것
  - main() 함수 아래쪽에 작성
- 함수 호출
  - '함수에 어떤 값을 전달해 어떤 작업을 하라'고 명령하는 것

2. 함수란

(2) 함수로 누적하기

형식

```
void 함수명(전달값); // 함수 선언

int main(void) {
    함수명(전달값); // 함수 호출
}

void 함수명(전달값) {} // 함수 정의
```

2. 함수란

(2) 함수로 누적하기

함수를 사용하는 이유

- 첫째, 코드 중복을 방지하고 효율적으로 프로그래밍할 수 있다.
- 둘째, 다른 프로젝트에 재사용할 수 있다.

2. 함수란

(3) 사용자 정의 함수

- 표준 함수 : C 언어에 이미 만들어져 있는 함수. 프로그램을 작성할 때 형식에 맞춰 가져다 쓰기만 하면 된다.
- 사용자 정의 함수 : 사용자가 직접 만들어(정의해) 사용하는 함수. 어떤 작업을 반복해야 하는데 표준 함수가 없을 때 직접 만들 수 있다.

2. 함수란

(3) 사용자 정의 함수

그림 5-6 함수로 정의한 상자의 작동 원리

```
void box(int num) {
    return num + 4;
}
```

반환형 / 함수명 / 매개변수 / 반환값

그림 5-7 함수의 매개변수

```
void 함수명(int num, int num) { }  (×)
```

변수명이 같으면 안 됨

```
void 함수명(int num1, int num2, char c, float f) { } (○)
```

여러 자료형 가능

변수명 다르게

2. 함수란

(3) 사용자 정의 함수

그림 5-8 함수 선언과 함수 정의

```
// 함수 선언
void 함수명(int num1, int num2, char c, float f);

// 함수 정의
void 함수명(int num1, int num2, char c, float f) { }
```

반환형, 함수명, 매개변수의 종류와 개수가 같아야 함

2. 함수란(call by value)

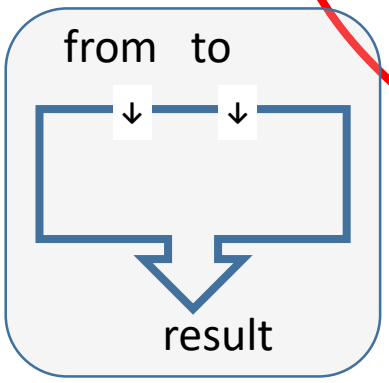
```
int sum(int from, int to); // 함수 선언(함수명 sum과 같은 이름의 변수명은 못 씀
int main(void){
    int tot = sum(1, 100); // 함수 호출
}
```

매개변수

매개변수

```
// 함수 정의
int sum(int from, int to){
    int result = 0;
    for(int i=from ; i<=to ; i++){
        result += i;
    }
    return result;
}
```

리턴값(반환값)



2. 함수란(call by reference)

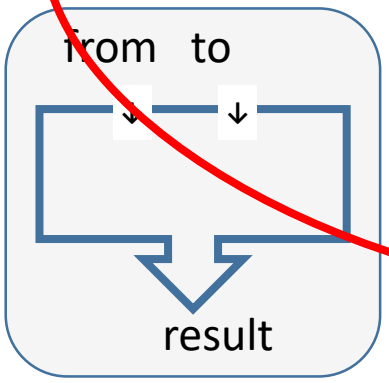
```
void sum(int from, int to, int* tot); // 함수 선언(함수명 sum과 같은 이름의 변수명은 못 씀
void main(void){
    int tot;
    sum(1, 100, &tot); // 함수 호출
}
```

매개변수

매개변수

```
// 함수 정의
void sum(int from, int to, int* tot){
    *tot = 0;
    for(int i=from ; i<=to ; i++){
        *tot += i;
    }
}
```

리턴값이 없으면 그냥 복귀



3. 함수의 종류

(1) 반환값이 없는 함수

- 반환형을 void로 선언한 함수

5.3.1 반환값이없는함수.c

```c
#include <stdio.h>

void function_without_return(); // 함수 선언

int main(void) {
    function_without_return(); // 반환값이 없는 함수 호출
    return 0;
}

void function_without_return() { // 함수 정의
    printf("반환값이 없는 함수입니다.\n");
}
```

실행결과

```
반환값이 없는 함수입니다.
```

3. 함수의 종류

(2) 반환값이 있는 함수

- int를 반환형으로 작성

5.3.2 반환값이있는함수.c

```c
int function_with_return(); // 함수 선언

int main(void) {
    int ret;
    ret = function_with_return(); // 반환값이 있는 함수 호출
    // int ret = function_with_return();
    printf("%d", ret);
    return 0;
}

int function_with_return() { // 함수 정의
    printf("반환값이 있는 함수입니다.\n");
    return 10;
}
```

함수의 반환값을 변수에 저장

실행결과

```
반환값이 있는 함수입니다.
10
```

function_with_return() 함수에서 출력

main() 함수에서 출력

3. 함수의 종류

(3) 매개변수(전달값)가 없는 함수

- 전달값
  - 함수를 호출할 때 함수에 전달하는 값
  - 함수에서는 이를 매개변수로 받는다.
- 전달값이 없는 함수도 만들 수 있다.
- 전달값이 없으면 매개변수도 필요 없으므로 함수를 선언할 때나 정의할 때 함수명 다음에 오는 소괄호에 아무것도 넣지 않는다.

3. 함수의 종류

(3) 매개변수(전달값)가 있는 함수

- 매개변수(parameter) : 함수를 호출할 때 전달되는 값이 저장되는 변수
- 인수(argument) : 함수를 호출할 때 전달하는 값이나 변수

```
void function(int a, int b); // 함수 선언
...
function(1, 2); // 함수 호출
```

→ a, b는 매개변수

→ 1, 2는 인수

3. 함수의 종류

(5) 반환값과 전달값이 있는 함수

⊖ 함수 선언(main() 함수 위)

```
int apple(int total, int ate);
```

⊖ 함수 정의(main() 함수 아래)

```
int apple(int total, int ate) {
    printf("전달값과 반환값이 있는 함수입니다.\n");
    return total - ate;
}
```

⊛ 함수 호출(main() 함수 안)

```
int ret = apple(5, 2);
```

4. 프로젝트: 1~3단계 구구단을 외자

(1) 문제 생성하기 : 3단계 문제를 모두 맞추어야 성공

㉠ 문제 풀기 3단계 반복 : for 문

㉡ rand() 함수로 난수 생성 (1리벨은 1~4/2레벨은 5~8/3레벨은 9~12)
- 전처리기 지시문에 time.h, stdlib.h 파일 추가
- 난수 초기화

㉢ 문제가 화면에 x × y = ? 구구단 출제 : printf() 문

④ 비밀번호가 맞추면 다음 단계로, 비밀번호가 틀리면 실패 종료

⑤ 3단계 모두 정담을 맞힐 경우 성공 메시지 출력

4. 프로젝트: 1~3단계 구구단을 외자

난수 생성 부분 함수로 구현하기

- ⊖ 난수 생성 : int getRandomNumber(int level) call by value함수
- ⊖ 함수 선언
  - 반환형 : int
  - 전달값 i
  - 매개변수 : int level로 선언
- ⊛ 함수 정의
  - 점점 어려운 문제가 나오도록 정의
    (1리벨은 1~4/2레벨은 5~8/3레벨은 9~12)
  - 생성한 난수를 return 문으로 반환

```
int getRandomNumber(int level){
    //1리벨은 1~4/2레벨은 5~8/3레벨은 9~12
    return (rand()% 4 +1) + (level-1)*4;
}
```

4. 프로젝트: 비밀번호 마스터

문제 출력 부분 함수로 구현하기

⊖ 문제 출력 함수 : void showQuestion(int level, int num1, int num2)
call by value 함수

⊖ 함수 선언
- 반환형 void

⊛ main() 함수 아래에 showQuestion() 함수 정의

④ 몇 번째 문제인지 printf() 문으로 출력

⑤ 문제 출력

⑥ printf() 문으로 행 구분, 비밀번호(문제의 정답) 입력 안내 문구 출력

```
void showQuestion(int level, int num1, int num2){
    printf("\n%d레벨 구구단을 외자~\n", level);
    printf("힌트 : %d x %d >>  ", num1, num2);
}
```

4. 프로젝트: 비밀번호 마스터

(2) 정답 입력받기

answer 변수에 저장될 답

- ⊖ 입력값이 난수로 생성한 두 수를 곱한 값과 똑같은 경우 : 정답 메시지 표시하고 다음 단계 문제 출제
- ⊖ 두 수를 곱한 값과도 같지 않은 경우 : 실패 메시지 출력하고 종료
- ⊛ 3단계 모두 정답을 맞힐 경우 성공

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
int getRandomNumber(int level);
void showQuestion(int level, int num1, int num2) ;
int main(void){
    srand((unsigned int) time(NULL)); // 난수 초기화

}
```

```
1레벨 구구단을 외자~
힌트 : 1 x 2 >>  2
1단계 정답입니다

2레벨 구구단을 외자~
힌트 : 6 x 6 >>  35
2단계 오답입니다. 실패 종료입니다
```

```
1레벨 구구단을 외자~
힌트 : 4 x 1 >>  4
1단계 정답입니다

2레벨 구구단을 외자~
힌트 : 6 x 7 >>  42
2단계 정답입니다

3레벨 구구단을 외자~
힌트 : 9 x 11 >>  99
3단계 정답입니다
비밀번호를 모두 맞췄습니다. 성공입니다
```