

6

함수

- 학습목표**
- 함수의 개념과 종류, 필요성을 이해하고 함수를 정의하는 방법을 설명할 수 있다.
 - 사용자 정의 함수를 호출하고 결과값을 반환하는 프로그램을 작성할 수 있다.

생각 펼치기

산업이 발달하기 전에는 필요한 물건이 있으면 각자가 만들어 사용했다. 떡을 만들고 싶으면 농사를 지어 쌀을 만들고, 쌀을 찌서 방아에 넣고 찼어서 떡을 만들었다. 그러나 현대 사회에서는 일이 분업화되면서 그 일이 직업이 되었다. 방앗간은 계속 쌀을 빻고, 버스 기사는 운전을 하고, 자동차 공장의 도색하는 팀은 계속 색을 칠하고, 문을 끼우는 팀은 계속 문을 장착한다.

이와 같이 효율을 높이기 위해 각자가 맡은 역할별로 일을 구분하는 경우가 많은데, 프로그래밍에서도 이런 기능을 수행하는 것이 있을까?



핵심 질문

프로그램을 작성할 때 같은 역할을 하는 코드를 반복하여 사용하지 않고, 효율적으로 작성하려면 어떻게 해야 할까?

미션

이 단원을 학습하면서 해결해 보자.



보통 학급 전체의 성적으로 표시한 학급별 성적 일람표에는 표와 같이 학생별로 여러 과목의 성적, 총점, 평균 등이 기록되어 있다.

번호 \ 과목	국어	영어	수학	정보	총점	평균
1	87	92	75	91	345	86.25
2	68	95	76	88	327	81.75
3	77	88	91	87	343	85.75
4	78	81	96	76	331	82.75

위의 자료를 이용하여 우리 반 학생 4명의 총점과 평균을 구하는 프로그램을 사용자 정의 함수를 이용하여 작성해 보자.

1 함수에 대해 알아보자

프로그래밍에서 반복하는 작업이 필요할 경우 함수의 이름만 적어 주면 함수 안에 들어 있는 명령들이 실행된다. 이와 같이 함수는 프로그램 코드를 재사용하는 유용한 도구이다. 함수는 특정 작업을 수행하는 코드의 집합으로, 필요한 데이터를 전달받고 작업이 완료되면 그 결과를 함수를 호출한 곳으로 반환할 수 있다.

다음 프로그램과 같이 학생별 총점을 구하려면 총점을 구할 때마다 반복문을 계속 작성해 주어야 한다. 이처럼 총점을 구할 때마다 코드를 작성한다면 상당히 번거롭고 코드의 길이가 길어지고, 무엇보다도 총점을 구하는 공식을 바꾸면 사용한 반복문을 일일이 수정하여야 한다.

프로그램

```
01 #include <stdio.h>
02 int main( )
03 {
04     int stScore1[5] = { 87, 92, 95, 91, 0};
05     int stScore2[5] = { 68, 95, 76, 88, 0};
06     int stScore3[5] = { 77, 88, 91, 87, 0};
07     int stScore4[5] = { 78, 81, 96, 76, 0};
08     int countSubject = 0;
09     for (countSubject = 0; countSubject<4; countSubject++) {
10         stScore1[4] = stScore1[4] + stScore1[countSubject];
11     }
12     printf("1번 학생의 총점은 %d", stScore1[4]);
13     return 0;
14 }
```

마지막 칸에는 총점 입력

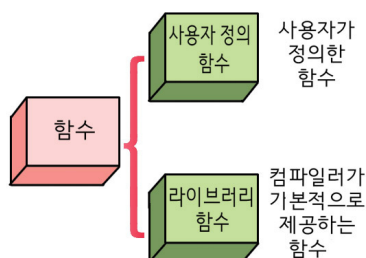
첫 번째 학생의 총점 구하기

실행 결과

1번 학생의 총점은 365

각 학생들의 총점을 필요할 때마다 구하려면 반복문을 학생마다 반복해 주어야 함.

함수는 사용자 정의 함수와 라이브러리 함수로 구별할 수 있다. 사용자 정의 함수는 프로그래머가 정의하는 함수이고, 라이브러리 함수는 컴파일러가 기본적으로 제공하는 함수이다. 위 프로그램에서 총점을 계산하는 9~11번 줄의 코드는 사용자 정의 함수로 작성할 수 있다.



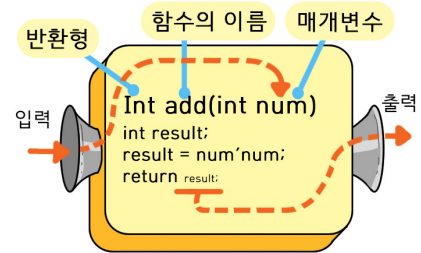
[그림 II-6] 함수의 구분

함수의 장점

- 프로그램에서 중복되는 코드를 제거할 수 있다.
- 복잡한 프로그램을 간단한 작업들로 분해하여 처리할 수 있다.
- 한 번 만들어진 함수를 다른 프로그램에도 재사용할 수 있다.
- 여러 사람이 작업할 때 일을 분담하기 쉬워진다.
- 프로그램을 수정하기 쉬워진다.

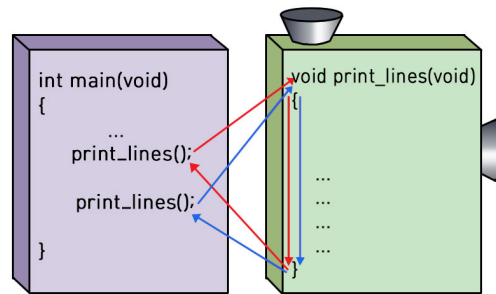
2 사용자 정의 함수를 만들어 보자

함수를 사용하기 위해서는 먼저 함수를 작성하는 것이 필요하다. 함수를 작성하는 것을 C 언어에서는 함수를 ‘정의(Definition)한다’라고 하며, 함수를 사용하는 것을 함수를 ‘호출(Call)한다’고 한다.



1 함수의 호출

함수를 호출(Function Call)하기 위해서는 프로그램 코드 중에 함수를 호출할 위치에 함수 이름을 쓰면 된다. 함수가 호출되면 함수 안의 문장이 순차적으로 실행되며, 실행이 끝나면 호출한 원래 위치로 되돌아간다.



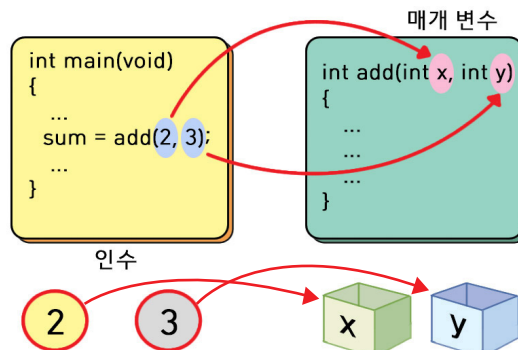
2 매개 변수와 인수

Q & A

Q 매개 변수와 인수는 같은 기억 공간을 사용할까? 즉 같은 변수일까?

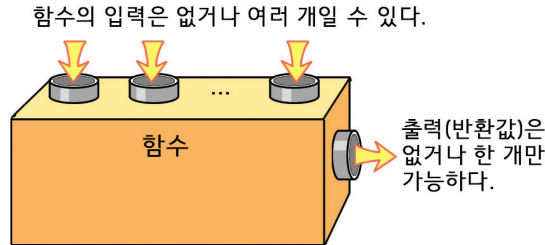
A 다른 값이며 다른 메모리 공간을 사용한다. 만일 인수로 변수를 사용하면 그 변수가 사용하는 메모리 공간과 함수의 매개 변수가 사용하는 기억 공간은 다르다.

매개 변수(Parameter)는 함수 이름 뒤 소괄호 안에 넣는 변수들이다. 함수는 필요에 따라서 매개 변수를 이용해서 호출 시에 자료를 전달받을 수 있다. 인수(Argument)는 호출 프로그램에 의하여 함수에 실제로 전달되는 값이고, 함수가 호출될 때마다 인수는 함수의 매개 변수에 전달된다.



3 반환

함수는 필요에 따라 자신을 호출한 곳으로 값을 반환(Return)할 수 있다. 값을 반환하기 위해서는 함수를 정의할 때 함수 이름 앞에 반환할 값의 자료형을 제시한다. 정수 2개를 입력받아 합한 결과를 정수로 반환하려면 함수 이름 앞에 정수형을 기록하고, 함수 안에서 return문을 사용하여 반환할 정수나 정수식을 넣는다.



다음 프로그램에서 함수 totalScore는 정수 배열을 받아 배열의 요소를 합한 정수 값을 반환한다.

프로그램

```
01 #include <stdio.h>
02 int totalScore(int grade[ ]) {
03     int countSubject, sum=0;
04     for(countSubject=0; countSubject<4; countSubject++) {
05         sum = sum + grade[countSubject];
06     }
07     return sum;
08 }
09 int main( )
10 {
11     int stScore1[5] = { 87, 92, 95, 91, 0};
12     int stScore2[5] = { 68, 95, 76, 88, 0};
13     int stScore3[5] = { 77, 88, 91, 87, 0};
14     int stScore4[5] = { 78, 81, 96, 76, 0};
15     stScore1[4] = totalScore(stScore1);
16     printf("1번 학생의 총점은 %d", stScore1[4]);
17     return 0;
18 }
```

int는 함수 totalScore의 반환형 매개 변수는 배열

sum은 함수 totalScore의 반환할 값

총점을 구할 때마다 함수를 호출하면 됨.

실행 결과

1번 학생의 총점은 365

함수의 반환 타입

- 함수에 반환 타입이 없을 경우에는 void라는 키워드를 사용하여 함수의 몸체에서 기능적인 처리를 하고, 작업을 끝낸다.
- 만약 반환 타입이 void가 아니라면, 함수의 몸체에서는 적절하게 반환 타입에 맞게 return해 주어야 한다.
- 함수는 매개 변수는 없고 반환값만 있는 경우도 있고, 반대로 매개 변수는 있지만 반환이 없을 수도 있다.
- 매개 변수가 없는 경우에는 괄호 안을 비워두거나 void라고 쓴다.

4 함수의 선언

Q & A

❓ 함수의 선언과 정의만 다른 파일에 한다면, main() 함수 안에서 그 함수를 사용할 수 있을까? 만일 가능하다면 어떻게 할까?

ⓐ 가능하다. 함수의 선언과 정의는 다른 파일에 한 후, 그 파일의 위치와 이름을 #include로 포함해 주면 된다. 실제 많은 프로그램들은 함수의 선언과 정의만 따로 모은 파일을 만들어 사용한다.

main() 함수 안에서 사용자 정의 함수를 사용하기 위해서는 사용하기 전에 컴파일러에게 함수에 대한 정보를 미리 알려 주어야 한다. 즉, main() 함수 위에 사용자 정의 함수를 정의하는 것이 필요하다.

함수를 미리 정의하고 사용하면 되지만 항상 그렇게 할 수 있는 것은 아니다. 따라서 다음 프로그램과 같이 함수 원형(Function Prototype)을 사용해서 함수를 사용하기 전에 미리 컴파일러에게 함수에 대한 정보를 알려 준다. 함수 원형은 함수가 사용되기 이전 즉, main() 함수 위에 선언해야 한다.

프로그램

```
01 #include <stdio.h>
02 int totalScore(int grade[ ]);
03 int main( )
04 {
05     int stScore1[5] = { 87, 92, 95, 91, 0};
06     int stScore2[5] = { 68, 95, 76, 88, 0};
07     int stScore3[5] = { 77, 88, 91, 87, 0};
08     int stScore4[5] = { 78, 81, 96, 76, 0};
09     stScore1[4] = totalScore(stScore1);
10     printf("1번 학생의 총점은 %d", stScore1[4]);
11     return 0;
12 }
13 int totalScore(int grade[ ] ) {
14     int countSubject, sum=0;
15     for(countSubject=0; countSubject<4; countSubject++) {
16         sum = sum + grade[countSubject];
17     }
18     return sum;
19 }
```

함수의 선언

총점을 구할 때마다 함수를 호출하면 됨.

함수 정의

실행 결과

1번 학생의 총점은 365

예제

미션에서 제시한 학생 성적 자료와 프로그램의 총점을 구하는 totalScore를 이용하여 배열에 들어 있는 4개 값의 평균을 구하는 사용자 정의 함수를 정의해 보자(단, 매개 변수는 score[], 반환형은 실수(double)로 한다).

프로그램

```
01 double average(double score[ ]){
02     int countSubject;
03     double result = 0;
04     for(countSubject = 0; countSubject < 4; countSubject++)
05         result = result + score[countSubject];
06     return result/4;
07 }
```

예제

사용자가 입력한 세 수가 삼각형의 세 변의 길이라고 가정하고, 사용자 정의 함수를 사용하여 입력한 세 수가 삼각형을 이룰 수 있는지 판별하는 프로그램을 작성해 보자.

알고리즘 설계

- ① 정수 3개를 매개 변수로 받는다.
- ② 조건문을 이용하여 세 수 중 가장 큰 수가 나머지 두 수의 합보다 큰지 판별한다.
- ③ 삼각형이 되면 1, 그렇지 않으면 0을 반환한다.

프로그램

```
01 #include <stdio.h>
02 int isTriangle(int a, int b, int c)
03 {
04     if ((a+b)<c || (a+c)<b || (b+c)<a)
05         return 0;
06     else
07         return 1;
08 }
09 int main( )
10 {
11     if (isTriangle(12, 20, 7))
12         printf("삼각형이 됩니다.");
13     else
14         printf("삼각형이 안 됩니다.");
15     return 0;
16 }
```

실행 결과

삼각형이 안 됩니다.

1. 우리 반 학생 4명의 총점과 평균을 구하는 프로그램을 사용자 정의 함수를 이용하여 작성해 보자.

총점을 구하는 사용자 정의 함수의 원형 - double totalScore(double score[])

평균을 구하는 사용자 정의 함수의 원형 - double average(double score[])

프로그램

```
01 #include <stdio.h>
02 double totalScore(double score[ ]);
03 double average(double score[ ]);
04 int main( )
05 {
06     double stScore1[6] = {87, 92, 95, 91, 0, 0};
07     double stScore2[6] = {68, 95, 76, 88, 0, 0};
08     double stScore3[6] = {77, 88, 91, 87, 0, 0};
09     double stScore4[6] = {78, 81, 96, 76, 0, 0};
10     stScore1[4] = totalScore(stScore1);
11     stScore2[4] = totalScore(stScore2);
12     stScore3[4] = totalScore(stScore3);
13     stScore4[4] = totalScore(stScore4);
14     stScore1[5] = average(stScore1);
15     stScore2[5] = average(stScore2);
16     stScore3[5] = average(stScore3);
17     stScore4[5] = average(stScore4);
18     printf("1번 학생의 총점 %lf 평균 %lf\n", stScore1[4], stScore1[5]);
19     printf("2번 학생의 총점 %lf 평균 %lf\n", stScore2[4], stScore2[5]);
20     printf("3번 학생의 총점 %lf 평균 %lf\n", stScore3[4], stScore3[5]);
21     printf("4번 학생의 총점 %lf 평균 %lf\n", stScore4[4], stScore4[5]);
22     return 0;
23 }
24 double totalScore(double score[ ]){
25     int countSubject=0;
26     double result;
27     for(countSubject=0;countSubject<4; countSubject++)
28         result = result + score[countSubject];
29     return result;
30 }
31 double average(double score[ ]){
32     int countSubject;
33     double result;
34     for(countSubject=0;countSubject<4; countSubject++)
35         result = result + score[countSubject];
36     return result/4;
37 }
```

함수 호출

총점을 구하는 함수

평균을 구하는 함수

실행 결과

```
1번 학생의 총점 365.000000
평균 91.250000
2번 학생의 총점 327.000000
평균 81.750000
3번 학생의 총점 343.000000
평균 85.750000
4번 학생의 총점 331.000000
평균 82.750000
```

3 지역 변수와 전역 변수에 대해 살펴보자

C 언어에서 변수는 적용 범위에 따라 지역 변수(Local Variable)와 전역 변수(Global Variable)로 나눌 수 있다.

1 지역 변수

지역 변수는 하나의 코드 블록 내부에서만 정의되는 변수를 말한다.

프로그램

```

01  #include <stdio.h>
02  void add( )
03  {
04      int num1;
05      int num2;
06      num1 = 10;
07      num2 = num1 * 2;
08  }
09  int main( )
10  {
11      int num2 = 5;
12      add( );
13      printf("%d", num2);
14      return 0;
15  }
```

add 함수용 지역 변수 선언

main 함수용 변수 num2 선언

사용 범위에 따른 C 언어 변수

- 지역 변수(Local Variable)
- 전역 변수(Global Variable)
- 정적 변수(Static Variable)
- 외부 변수(Extern Variable)

실행 결과

5

[그림 II-7]에서 보면 코드 블록 3에서 선언한 변수는 코드 블록 3 밖에서는 사용할 수 없다. 코드 블록 1에서 선언한 변수는 코드 블록 2와 3에서 사용할 수 있다. 왜냐하면, 코드 블록 2와 3은 코드 블록 1 안에 있기 때문이다.

코드 블록

C 언어에서 중괄호 { } 묶여진 부분을 말한다. 코드 블록 내에서는 변수를 선언할 수 있고, 그 코드 블록에서만 사용한다.

```

if(a>1)
{
    ...
    if(b>2)
    {
        if(c>3)
        {
            ...
        }
        ...
    }
    ...
}
```

코드 블록 3

코드 블록 2

코드 블록 1

[그림 II-7] 코드 블록과 변수

2 전역 변수

전역 변수는 전역 변수가 선언된 파일 및 프로그램 전체에서 사용할 수 있는 변수이다. 만일 여러 개의 소스 파일로 하나의 실행 파일을 만드는 경우의 전역 변수는 프로그램에 포함된 여러 가지 소스 파일에서 사용할 수 있다.

함께
해결하기

다음 프로그램을 파일 3개에 나누어 작성하고 실행해 보자.

프로그램 조건

- ① 두 개의 변수 num1과 num2의 선언은 myvar.h에 작성한다.
- ② 함수 add는 myfunc.h에 작성한다.
- ③ main 함수는 mymain.c에 작성하고, 파일 제일 위에 myvar.h와 myfunc.h를 포함한다.

프로그램

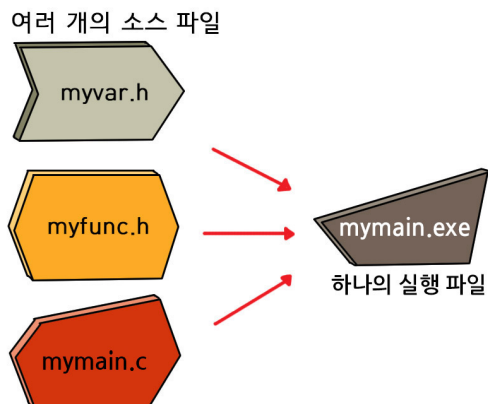
```
01 #include <stdio.h>
02 int num1;
03 int num2;
04 void add( )
05 {
06     num1 = 10;
07     num2 = num1 * 2;
08 }
09 int main( )
10 {
11     int num2 = 5;
12     add( );
13     printf("%d", num2);
14     return 0;
15 }
```

전역 변수 num1, num2 선언

num1과 num2의 값은 add 함수용

main 함수용 변수 num2 선언

프로그램 구성



4 되부름 함수에 대해 알아보자

되부름 함수란 사용자 정의 함수 안에서 자기 자신을 다시 호출하는 함수를 말한다. 되부름 함수는 재귀 호출(Recursive Call)이라고도 하는데, 이는 어떤 문제를 풀 때 큰 문제를 큰 문제와 동일한 규칙을 갖고 있는 반복되는 형태의 작은 문제로 만드는 방식이다.

다음 프로그램에는 2개의 사용자 정의 함수가 있는데, 하나는 재귀 호출을 이용한 것이고 다른 하나는 반복문을 이용한 것으로 두 함수의 결과는 동일하다.

프로그램

```
01 #include <stdio.h>
02 int factorial_r( int n ) ● 재귀 호출을 이용한 팩토리얼 계산
03 {
04     if ( n > 1 ) ● 재귀 조건
05     {
06         return ( n*factorial_r(n-1));
07     }
08     else
09     {
10         return 1; ● STOP 조건
11     }
12 }
13 int factorial_l( int n ) ● 반복문을 이용한 팩토리얼 계산
14 {
15     int i, p=1;
16     for ( i=1; i<=n; i++ )
17     {
18         p = p*i;
19     }
20     return p;
21 }
22 int main( )
23 {
24     int recursive, loop;
25     recursive = factorial_r(5);
26     loop = factorial_l(5);
27     printf("재귀 호출 %d\n", recursive );
28     printf("반복문 %d\n", loop);
29     return 0;
30 }
```

실행 결과

재귀 호출 120
반복문 120

Q & A

Q 재귀 호출을 사용한 코드를 반복문으로도 표현할 수 있다면 두 방법에는 어떤 차이가 있을까?







A 일반적으로 재귀 호출을 사용하면 코드가 간결하고 명확해진다. 대신 재귀 호출을 사용하면 함수를 여러 번 반복하면서 호출하여 함수 호출이 과도하게 많아지게 되어 CPU에 부담이 되며 함수의 반복 호출에 따른 실행 시간도 증가하게 된다. 더불어 함수 내에서 함수를 호출하면 현재 작업하던 내용을 스택이라는 메모리 공간에 기억해야 하므로 기억 장소를 낭비하게 된다. 이때, 반복문을 사용하면 코드가 다소 복잡해 보이지만, CPU에 부하를 적게 주고 처리 속도도 좀 더 빠르다.

예제

한 쌍의 토끼는 1개월 후부터 매달 한 쌍의 토끼를 낳는다. 만약, 토끼가 죽지 않는다고 할 때, 10개월 후에는 몇 쌍의 토끼가 낳게 되는지 구하는 프로그램을 작성해 보자.

피보나치 수열

피보나치 수열이란 앞의 두 수의 합이 바로 뒤의 수가 되는 수열이다. 이 수열을 처음 소개한 사람의 이름을 따서 피보나치 수열이라고 한다. 피보나치의 수열은 자연 속의 꽃잎의 수나 해바라기 씨앗의 개수에서 찾아볼 수 있다.

개월	처음	1개월 후	2개월 후	3개월 후	4개월 후	5개월 후	...
토끼							...
토끼 쌍의 수	1쌍	1쌍	2쌍	3쌍	5쌍	8쌍	...

알고리즘 설계

- ① 정수 1개를 입력받는다.
- ② 재귀 호출을 사용하여 입력받은 수보다 작을 때까지 반복하며 사용자 정의 함수를 호출한다.
사용자 정의 함수는 재귀 함수를 이용한다.

프로그램

```

01  #include <stdio.h>
02  int fibo(int num);
03  int main(void){
04      int num;
05      printf("\n개월 수 입력: " );
06      scanf("%d" , &num) ;
07      printf("총 토끼 쌍: %d", fibo(num+1));
08      return 0;
09  }
10  int fibo(int num){
11      if(num == 0) return 0;
12      else if(num == 1) return 1;
13      else return fibo(num-1) + fibo(num-2);
14  }

```

실행 결과

개월 수 입력: 10
총 토끼 쌍: 89

5 라이브러리 함수를 사용해 보자

라이브러리 함수는 사용자 정의 함수와는 다르게 C 언어 컴파일러가 기본적으로 제공하는 함수이다. 개발용 언어로 C 언어를 많이 사용하는 이유도 라이브러리 함수의 지원이 잘 되어 있기 때문이다. 다음은 C 언어에서 라이브러리 함수가 사용되는 경우이다.

- 표준 입출력(시스템 함수)
- 문자열 처리
- 오류 처리
- 수학 연산
- 시간 처리
- 데이터 검색과 정렬

1 난수 발생 함수

라이브러리 함수 중 많이 사용하는 함수가 난수 발생 함수이다. 난수는 규칙성 없이 임의로 생성되는 수이고, 암호학이나 게임 등에서 필수적으로 사용된다.

예제 난수 발생 함수를 이용하여 1에서 6사이의 난수 10개를 발생시키는 프로그램을 작성해 보자.

프로그램

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 #include <time.h>
04 int main(void)
05 {
06     int a;
07     srand((unsigned)time( NULL ));
08     for(a = 0; a < 10; a++)
09         printf("%d ", 1+rand( )%6);
10     return 0;
11 }
```

난수 생성, 동적 메모리 관리 등의 함수를 포함.

시간과 날짜를 얻거나 조작하는 함수를 포함.

1~6 사이의 정수를 만들기 위해서는 0을 없애기 위해 1을 더하고 % 연산을 이용해 6의 나머지만 취함.

난수 발생 함수

rand()는 0에서 32367 사이의 값을 임의로 발생시키는 함수이다. int a = rand() 형태로 사용하는데, 프로그램을 실행할 때마다 같은 수만 나오게 될 수 있다. 따라서 프로그램이 실행할 때마다 다른 난수를 발생시키기 위해 난수 발생 초기 수치를 지정하는 함수인 srand()를 쓴다. 이 함수에 매개 변수로 현재 시간을 넣으면, 현재 시각은 항상 변하므로 다른 난수를 발생시킬 수 있다.

실행 결과

4 6 3 2 4 1 5 2 3 1

2 수학 함수

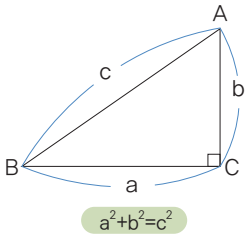
수학 함수의 종류는 아주 다양한데 그중 일부를 정리하면 다음 표와 같다. 수학 함수를 사용하기 위해서는 프로그램에 `math.h` 헤더 파일을 포함시켜야 한다.

[표 II - 11] 수학 함수의 종류

분류	함수	설명
삼각 함수	<code>double sin(double x)</code>	x의 사인값 계산
	<code>double cos(double x)</code>	x의 코사인값 계산
	<code>double tan(double x)</code>	x의 탄젠트값 계산
지수 함수	<code>double exp(double x)</code>	e^x 계산
	<code>double log(double x)</code>	x의 자연로그 계산
	<code>double log10(double x)</code>	x의 상용로그 계산
기타 함수	<code>int abs(int x)</code>	정수 x의 절댓값 계산
	<code>double fabs(double x)</code>	실수 x의 절댓값 계산
	<code>double sqrt(double x)</code>	x의 제곱근 계산

피타고라스의 정리

직각 삼각형에서 직각을 낀 두 변의 길이를 각각 a, b라 하고, 빗변의 길이를 c라하면 $a^2 + b^2 = c^2$ 이 성립한다.



예제

제곱근을 구하는 `double sqrt` 함수를 이용하여 피타고라스의 정리를 활용하는 프로그램을 작성해 보자. 두 수를 입력받고 그 수가 직각 삼각형의 넓이와 높이라 가정할 때, 빗변의 길이를 구해 보자.

프로그램

```

01 #include <stdio.h>
02 #include <math.h>
03 int main( )
04 {
05     double s1, s2;
06     printf("두 변의 길이를 입력하시오.\n");
07     scanf("%lf %lf", &s1, &s2);
08     printf("빗변의 길이는 %lf 입니다.", sqrt((s1*s1)+(s2*s2)));
09     return 0;
10 }

```

`sqrt()` 함수를 사용하기 위해서임.

피타고라스 정리를 활용해서 빗변의 길이를 구함.

실행 결과 예

두 변의 길이를 입력하시오.
3 4
빗변의 길이는 5.000000 입니다.

사용자 정의 함수를 이용하여, cm를 inch로 변환하는 프로그램을 작성해 보자.

프로그램

```
01 #include <stdio.h>
02 double cmtoinch(double cmv);
03 int main(void){
04     double cm, inch;
05     printf("cm값 입력: " );
06     scanf("%lf" , &cm) ;
07     inch = cmtoinch(cm);
08     printf("%lfcm는 %lfinch " , cm, inch);
09     return 0;
10 }
11 double cmtoinch(double cmv) {
12     double result;
13     result = cmv * 0.393701;
14     return result;
15 }
```

실행 결과 예

cm값 입력: 10
10.000000cm는 3.937010inch

응용하기

10개의 정수의 평균을 구하는 사용자 정의 함수를 작성하고, 이를 이용하여 평균을 구하는 프로그램을 작성해 보자.

프로그램

```
01 #include <stdio.h>
02 double average(int score[ ]);
03 int main( )
04 {
05     double result;
06     int numbers[10] = {87, 92, 95, 91, 16, 23, 45, 67, 98, 35};
07     result = average(numbers);
08     printf("%lf", result);
09     return 0;
10 }
11 double average(int score[ ]) {
12     int countSubject;
13     double result = 0;
14     for(countSubject = 0; countSubject < 10; countSubject++)
15         result = result + score[countSubject];
16     return result/10;
17 }
```

평균을 구하는 함수

실행 결과

64.900000



수행 평가

사용자 정의 함수를 이용한 소수 판별 프로그램

활동 목표 사용자 정의 함수를 이용하여 프로그램을 작성할 수 있다.

입력한 정수가 소수인지 판별하는 프로그램이다.

다음 프로그램 코드를 사용자 정의 함수를 활용하는 형태로 수정해 보자.

프로그램

```
01  #include <stdio.h>
02  int main( )
03  {
04      int n, count=0, a=2;
05      printf("2 이상 정수 입력: ");
06      scanf("%d",&n);
07      while(a<=n)
08      {
09          if(n%a==0)
10              count++;
11          a++;
12      }
13      if(count==1) {
14          printf("%d는 소수입니다.\n",n);
15      }
16      else {
17          printf("%d는 소수가 아닙니다.\n",n);
18      }
19      return 0;
20  }
```

사용자 정의 함수의
이름은 findprime으로 하고
매개 변수는 정수 1개,
반환값은 정수예요.



스스로 평가하기

평가 항목	구분		
	그렇다	보통이다	그렇지 않다
• 사용자 정의 함수를 목적에 맞게 정의할 수 있다.			
• 사용자 정의 함수를 이용하여 프로그램을 작성할 수 있다.			



내 실력 확인하기

내용을 이해했나요?

- **사용자 정의 함수:** 프로그래머가 만든 특정 작업을 수행하는 명령들의 집합이다. 같은 일을 반복하는 것을 사용자 정의 함수로 만들면 수정도 용의하고 코드의 길이도 줄일 수 있다.
- **라이브러리 함수:** 자주 사용되는 특정 기능의 명령들을 모아서 컴파일러에서 제공하는 함수이다. 사용자 정의 함수와 비슷하지만 프로그래머가 정의할 필요가 없고 컴파일러에 따라 다르다.

문제로 확인할까요?

1. 사용자 정의 함수를 사용할 때 장점은 무엇인지 적어 보자.
2. 다음 프로그램은 컴파일 오류가 발생한다. 프로그램이 실행되었을 때 '7'이 출력될 수 있도록 프로그램을 수정해 보자.

```

01  #include <stdio.h>
02  int z;
03  void f(int x)
04  {
05      x = 2;
06      z = z + x;
07      return z;
08  }
09  int main(void)
10  {
11      z = 5;
12      printf("%d", f(z));
13      return 0;
14  }
    
```

평가해 볼까요?

★ 다음 평가 항목에 따라 자신의 성취 척도를 스스로 점검해 보자.

영역	평가 항목	척도				
		1	2	3	4	5
이해	사용자 정의 함수의 기능과 형식을 설명할 수 있는가?					
적용	프로그램에서 반복되는 기능을 함수로 정의할 수 있는가?					
태도	협동 학습에서 자신의 역할을 충실히 이행하며, 모둠에 충분한 도움을 주었는가?					