제3장

함수의 기본

학습 목표

- 사전 정의 함수
  - 값을 리턴하는 함수와 리턴하지 않는 함수
- 프로그래머 정의 함수
  - 정의, 선언, 호출
  - 재귀 함수
- 영역 규칙
  - 지역 변수
  - 전역 상수와 전역 변수
  - 블록, 중첩된 영역

함수 소개

- 프로그램의 블록을 만드는 것
- 다른 언어에서 다른 용어 :
  - 프로시저, 서브프로그램, **메소드**
  - C++ 에서: 함수
- I-P-O
  - 입력 – 처리 – 출력
  - 프로그램의 기본적인 하위 조각
  - 이러한 “조각”들을 위해 함수를 사용

사전 정의 함수

- 라이브러리 : 프로그램에서 사용 가능한 함수들로 구성!
- 2가지 종류:
  - 값을 리턴하는 함수
  - 값을 리턴하지 않는(void) 함수
- 라이브러리를 적용하기 위해 반드시 “#include”를 사용
  - 예,
    - <cmath>, <cstdlib> (Original "C" libraries)
    - <iostream> (cout, cin을 위한)

사전 정의 함수 사용

- 수학 함수들은 매우 방대함
  - <cmath.h> 라이브러리에서 찾을 수 있다
  - 자바의 Math 클래스에 해당
  - 대부분 값(해답)을 리턴

- 예제 : theRoot = sqrt(9.0);
  - 자바
  - 구성 :

    sqrt : 라이브러리 함수의 이름
  - I-P-O에서 :
    - I = 9.0
    - P = “제곱근을 계산”
    - O = 3, 리턴된 것 & theRoot에 할당된 것

1개의 값을 리턴하는 사전 정의 함수(1 of 2)

디스플레이 3.1 1개의 값을 리턴하는 사전 정의 함수

```
1   //사용자의 예산으로 구입할 수 있는
2   //개집의 크기를 계산한다.
3   #include <iostream>
4   #include <cmath>
5   using namespace std;

6   int main( )
7   {
8       const double COST_PER_SQ_FT =
9       double budget, area, lengthSi

10      cout << "Enter the amount bud
11      cin >> budget;

12      area = budget/COST_PER_SQ_FT;
13      lengthSide = sqrt(area);
14      cout.setf(ios::fixed);
15      cout.setf(ios::showpoint);
16      cout.precision(2);
17          cout << "For a price of $" <<
18              << "I can build you a luxu
19              << "that is " << lengthSi
20              << " feet on each side.\n"

21      return 0;
22  }
```

Sample Dialogue

```
Enter the amount budgeted for your doghouse
For a price of $25.00
I can build you a luxurious square doghouse
that is 1.54 feet on each side.
```

더 많은 사전 정의 함수

- #include <cstdlib>
  - 이 라이브러리는 다음과 같은 함수를 포함 :
    - abs() // int형 절대값을 리턴
    - labs() // long형 절대값을 리턴
    - *fabs() // 부동소수점형 절대값을 리턴
  - *fabs()는 사실 <cmath> 라이브러리에 있다!
    - 혼란스러울 수 있다.
    - 기억하라 : 해당 라이브러리는 C++이 태어난 이후에 추가되었다
    - 자세한 건 부록/매뉴얼을 참고

더 많은 사전 정의 함수

- pow(x, y)
  - x에 y의 거듭제곱을 리턴

    $3.0^{2.0} = 9.0$의 결과로 9.0이 출력된다.

```
double result, x = 3.0, y = 2.0;
result = pow(x, y);
cout << result;
```

- 이 함수는 2개의 인자를 받는 것에 주목하라
  - 함수는 여러 개의 인자, 다양한 데이터 형을 가질 수 있다.

더 많은 수학 함수 (1 of 2)

디스플레이 3.2 사전 정의 함수 몇 가지

이 모든 사전 정의 함수는 include 지시자뿐만 아니라 using namespace std;가 필요하다.

이름	설명	인자의 형	리턴 값의 형	예	리턴 값	라이브러리
sqrt	Square root	double	double	sqrt(4.0)	2.0	cmath
pow	Powers	double	double	pow(2.0,3.0)	8.0	cmath
abs	Absolute value for int	int	int	abs(-7) abs(7)	7 7	cstdlib
labs	Absolute value for long	long	long	labs(-70000) labs(70000)	70000 70000	cstdlib
fabs	Absolute value for double	double	double	fabs(-7.5) fabs(7.5)	7.5 7.5	cmath

더 많은 수학 함수 (2 of 2)

ceil	Ceiling (round up)	double	double	ceil(3.2) ceil(3.9)	4.0 4.0	cmath
floor	Floor (round down)	double	double	floor(3.2) floor(3.9)	3.0 3.0	cmath
exit	End program	int	void	exit(1);	없음	cstdlib
rand	Random number	없음	int	rand( )	다양함	cstdlib
srand	Set seed for rand	unsigned int	void	srand(42);	없음	cstdlib

사전 정의 void 함수

- 리턴되는 값이 없다
- 작업을 수행하지만, 해답을 보내지 않는다.
- 호출은 식(statement) 그 자체이다.
  - exit(1); // 리턴값이 없으므로 할당하지 않는다.
    - 이 호출은 프로그램을 종료
    - 마찬가지로 void 함수는 인자들을 가질 수 있다.
- 모든 측면에서 "값을 리턴하는" 함수와 똑같다.
  - 그러나, void 함수는 값을 리턴하지 않는다.

난수 생성기

- “무작위로 선택된” 수를 리턴
- 게임과 같은 시뮬레이션에 사용
  - rand()
    - 인자가 없음
    - 0 에서RAND_MAX 사이의 값을 리턴
  - 범위설정
    - rand() % 6은 6보다 작은 난수를 발생
    - 0 에서 5 사이의 난수를 리턴
  - 범위이동

    rand() % 6 + 1
    - 1 에서 6 사이로 범위 이동(예, 주사위)

난수 시드

- 의사 난수
  - rand 함수는 주어진 “시퀀스”에 따라 난수를 생성
- 시퀀스 대신에 “시드” 사용하기: srand(seed_value);
  - void 함수
  - “시드”로서 하나의 인자를 받음
  - 시스템 시간을 포함하여 어떠한 시드값이라도 사용 가능 : srand(time(0));
  - time() 함수는 숫자값으로 시스템 시간을 리턴
  - time() 함수는 <time> 라이브러리에 포함

난수 예제

- 0.0에서 1.0 사이의 double형 난수 :

  (RAND_MAX – rand())/static_cast<double>(RAND_MAX)

  – 부동 소수점 수로 나누기 위해 강제 형변환

- 1에서 6사이의 int형 난수:

  rand() % 6 + 1

  – “%”는 모듈러스 연산자(나머지)

- 10에서 20사이의 int형 난수 :

  rand() % 10 + 10

프로그래머 정의 함수

- 사용자가 정의한 함수!
  - 클래스에 포함시키지 않고 단독으로 정의 가능
- 프로그램의 블록을 만드는 것
  - 분할 & 정복
  - 가독성
  - 재사용
- 사용자가 정의한 것은 아래중의 하나 :
  - main()부분과 같은 파일
  - 분리된 파일

함수 사용의 구성

- 함수 사용의 세 부분:
  - 함수 선언/원형
    - 컴파일러에 대한 정보
    - 적절하게 해석된 함수를 호출하기 위함
  - 함수 정의
    - 함수가 하는 작업에 대한 실제 구현/코드
  - 함수 호출
    - 함수에 제어를 전달

함수 선언

- 함수 원형이라고도 함
- 컴파일러를 위한 “정보를 제공하는” 선언
- 컴파일러에게 어떻게 해석해서 호출할 것인지를 알려주는 것
  - 문법:

    <return_type> FnName(<formal-parameter-list>);
  - 예제:

    double totalCost(int numberParameter,

      double priceParameter);
- 함수 호출 이전에 위치해야 함
  - main() 부분에 선언
  - 또는 main() 위의 전역 공간

총합을 계산하기 위한 함수 (1 of 2)

디스플레이 3.5

```
1   #include <iostream>
2   using namespace std;

3   double totalCost(int numberParameter, double priceParameter);
4   //개당 가격이 priceParameter인 numberParameter개 물품에 대해
5   //5%의 판매 세금을 포함한 총가격을 계산한다.

6   int main( )
7   {
8       double price, bill;
9       int number;

10      cout << "Enter the number of items purchased: ";
11      cin >> number;
12      cout << "Enter the price per item $";
13      cin >> price;

14      bill = totalCost(number, price);
```

함수 선언.
함수 원형이라고도 한다.

함수 호출

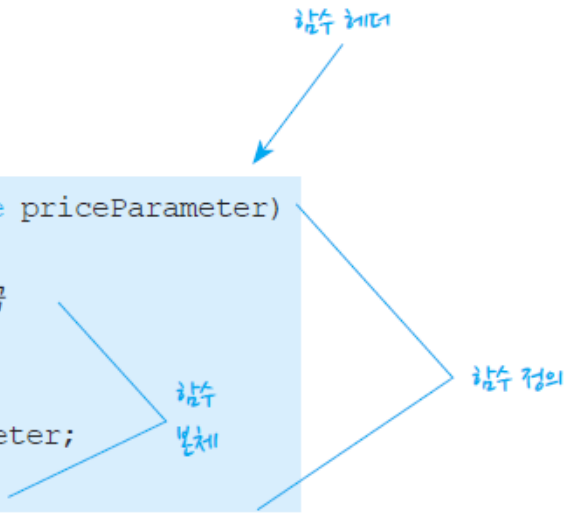
총합을 계산하기 위한 함수 (2 of 2)

```
15        cout.setf(ios::fixed);
16        cout.setf(ios::showpoint);
17        cout.precision(2);
18        cout << number << " items at "
19             << "$" << price << " each.\n"
20             << "Final bill, including tax, is $" << bill
21             << endl;

22        return 0;
23    }

24    double totalCost(int numberParameter, double priceParameter)
25    {
26        const double TAXRATE = 0.05; //5% 판매 세금
27        double subtotal;

28        subtotal = priceParameter * numberParameter;
29        return (subtotal + subtotal*TAXRATE);
30    }
```



Sample Dialogue

```
Enter the number of items purchased: 2
Enter the price per item: $10.10
2 items at $10.10 each.
Final bill, including tax, is $21.21
```

함수 선언 대안

- 다시 말하면 : 함수 선언은 컴파일러에 대한 정보
- 컴파일러는 오직 다음 내용만 필요:
  - 리턴 유형
  - 함수 이름
  - 매개변수 목록
- 형식 매개변수 이름은 필요하지 않음:

  double totalCost(int, double);
  - 형식매개 변수 이름을 여전히 사용
    - 가독성 향상

매개변수 vs. 인자

- 두 용어는 종종 서로 바꿔가며 사용
- 형식 매개변수/인자
  - 함수 선언에서
  - 함수 정의 머리부분에서
- 실 매개변수/인자
  - In function call
- 기술적으로 매개변수는 “형식적인” 부분이고, 인자는 “실질적인”* 부분
  - *항상 이와 같이 사용되진 않는다

함수를 호출하는 함수

- 이미 이러한 일을 하고 있었다!
  - main()은 함수이다!
- 유일한 제약사항:
  - 함수의 선언은 반드시 처음에 나타나야 함
- 전형적으로 함수의 정의는
  - main() 이후에 정의
  - 또는 다른 파일에서
- 함수에서 다른 함수를 호출하는 것이 보통
- 함수는 자기 자신을 호출할 수도 있다 → : “재귀”

선행조건과 사후조건

- "I-P-O" 논의와 비슷
- 함수 선언 주석:

```
void showInterest(double balance, double rate);
//선행조건: balance는 음수가 아닌 계좌 잔액이다.
      //rate는 이자율로서 5%라면 5로 표현된다.
      //사후조건: 주어진 이자율로
     //주어진 잔액에 대한 이자를 화면에 표시한다.
```

  - 종종 입력과 출력으로 얘기함

main(): “특별함”

- 다시 말하면 : main()은 함수이다.
- 특별하다는 의미:
  - 프로그램내에 오직 하나의 main()이 존재한다는 것
- main()은 누가 호출하나?
  - 운영 체제
  - 전통적으로 return 구문을 필요로 함
    - 호출자에게 리턴된 값 → 운영 체제
  - "int" 또는 "void" 를 리턴해야만 함

영역 규칙

- 지역 변수
  - 주어진 함수의 본체 내에 선언된 변수
  - 오직 해당 함수 안에서만 이용가능
- 다른 함수에서 같은 이름을 사용할 수 있다.
  - 영역은 지역적이다: “이 함수가 변수의 영역”
- 지역변수는 우선적
  - 각각의 데이터에 대한 통제를 유지
  - 기본적인 것을 알아야만 함
  - 함수는 작업에 필요한 지역 데이터가 무엇이든지 간에 지역 변수를 선언해야 함

프로시저의 추상화

- 함수가 어떻게 동작하는지 보다 무슨 일을 하는지 아는 것이 필요
- 블랙박스를 생각하자.
  - 장비에 대해 작동하는 방법을 아는 것이 아니라 어떻게 사용하는 지를 아는 것
- 블랙박스처럼 함수를 구현하자.
  - 함수의 사용자에게 필요한 것: 선언
  - 함수 정의는 필요하지 않다.
    - 정보 은닉이라 함
    - 함수가 어떻게 작동하는 지에 대한 자세한 부분을 감추는 것

전역상수와 전역 변수

- 함수 본체 외부에 선언
  - 그 파일의 모든 함수에게 전역적
- 함수 본체 내부에 선언
  - 그 함수에게 지역적
- 상수는 전형적으로 전역 선언:
  - const double TAXRATE = 0.05;
  - 전역적으로 선언하면 모든 함수 영역에서 이용가능
- 전역 변수?
  - 가능하나, 잘 사용하지 않는다!
  - 위험: 사용에 대한 통제가 없다!

블록

- 복합 구문 내에 데이터 선언
  - 블록이라 부름
  - 블록 영역을 가짐
- 주목: 모든 함수 정의는 블록이다!
  - 이것은 함수 영역이 지역적임을 의미
- 반복 블록:

```
for (int ctr=0;ctr<10;ctr++)
{
        sum+=ctr;
}
```

  - 변수 ctr은 오직 반복 블록안의 영역을 가짐

요약

- 함수는 블랙박스처럼 만들라
  - 자세한 부분을 은닉
  - 자신의 지역 데이터를 선언
- 함수 선언은 자체 문서화하라
  - 주석으로 사전조건과 사후조건을 제공
  - 호출자에게 사용에 필요한 모든 것을 제공
- 지역 데이터 v.s. 전역 데이터
  - 함수 정의 바깥에 선언됨Declared above function definitions
  - 전역 상수는 좋으나 변수는 안 좋다
- 매개변수/인자
  - 형식: 함수 선언과 정의내에서 들어오는 데이터에 대한 공간 확보자
  - 실제: 함수 호출에서 함수에 전달된 실제 데이터

실습 문제 - 1

- 다음과 같이 동작하는 프로그램을 작성하세요.
  - rand()를 사용하시오.
  - srand()를 쓴 경우와 아닌 경우의 차이점을 살펴보시오.

```
나는 100이하의 자연수 하나를 생각하고 있습니다.
숫자를 짐작해서 입력하세요: 85
내가 생각하고 있는 숫자는 85보다는 작은 숫자입니다.
숫자를 짐작해서 입력하세요: 78
내가 생각하고 있는 숫자는 78보다는 큰 숫자입니다.
숫자를 짐작해서 입력하세요: 81
맞췄습니다!!! 3번 만에 성공하셨네요!
```

실습 문제 - 2

- 인자 n에 대해서 n!를 return하는 함수 int getFactorial(int n)을 구현하고, 테스트 하는 main()을 작성하시오.
  - for문을 사용하는 경우로 작성하시오.
  - 재귀함수의 형태로 작성하시오.

  - 두 경우 모두에서 50!의 값이 이상하게 나오는 이유를 생각하시오.
  - 틀린 값이 나오지 않게 하는 방법에 대해서 고민해 보시오.

함수 문제 해결

- 재귀 호출을 이용하여 다음과 같이 동작하는 프로그램을 작성
  - 반복문 사용하지 말고 재귀호출만 활용

```
#include <iostream>

int main()
{
    int val ;

    cout >> "정수 입력: " ;
    cin >> val ;
    cout >> "바로 출력  : " ;
    showDigit(val) ;
    cout >> "거꾸로 출력: " ;
    showReverseDigit(val) ;
    return 0 ;
}
```

```
void showDigit(int n)
{
    if  ( ....  )
        showDigit( ......)
    ...
}
```

```
입력 : 7845
바로출력    : 5    4    8    7
거꾸로출력 : 7    8    4    5
```