

Chapter 3. 함수의 기본

박 종 혁 교수

UCS Lab

(<http://www.parkjonghyuk.net>)

Tel: 970-6702

Email: jhpark1@snut.ac.kr

Learning Objectives

- 사전정의 함수
 - 값을 리턴하는 함수, 리턴하지 않는 함수
- 사용자(프로그래머) 정의 함수
 - 정의, 선언, 호출
 - 재귀 함수
- 영역 규칙
 - 지역 변수
 - 전역 상수와 전역 변수
 - 블록, 내포된 영역

함수의 소개

- 프로그램의 빌딩 블록(Building Blocks)
- 다른 언어에서의 함수의 용어:
 - 프로시저(Procedures), 서브프로그램(subprograms), 메소드(methods)
 - C++에서는 : 함수(functions)
- I-P-O
 - Input – Process – Output
 - 프로그램의 기본 구성 요소
 - 이들 “조각(pieces)”을 위한 함수의 사용

사전 정의 함수 (Predefined functions)

- 우리가 사용하는 라이브러리 내의 함수!
- 두 가지 형태:
 - 값을 리턴하는 함수
 - 값을 리턴하지 않는 함수 (void)
- “#include”를 이용하여 적절한 라이브러리를 포함시켜야 함
 - 예,
 - <cmath>, <cstdlib> (“C언어” 라이브러리)
 - <iostream> (cout, cin을 위한)

사전 정의 함수의 사용

- 풍부한 수학 함수
 - `<cmath.h>` 라이브러리 내에 있다
 - 대부분 값을 리턴한다 (결과)
- 예: `theRoot = sqrt(9.0);`
 - 구성 요소:
 - `sqrt` = 라이브러리 함수명
 - `theRoot` = 결과 값이 할당되는 변수
 - `9.0` = 함수에서 사용되는 인자 또는 초기 입력값
 - In I-P-O:
 - I = 9.0
 - P = “루트 계산”
 - O = 3, 리턴되고 `theRoot`에 할당

함수 호출(Function Call)

- 결과의 할당:

`theRoot = sqrt(9.0);`

- 식 "`sqrt(9.0)`"은 함수를 호출(function call 또는 function invocation)하는 부분
- 함수 호출 내의 인자 (9.0) 부분은 리터럴, 변수, 식 모두 가능
- 호출 부분이 식의 일부가 될 수 있음:
 - `bonus = sqrt(sales)/10;`
 - 함수의 리턴 형이 정확하게 사용된다면 함수 호출이 허용된다

디스플레이 3.1 값을 리턴하는 사전 정의 함수 (1 of 2)

Display 3.1 A Predefined Function That Returns a Value

```
1  //Computes the size of a doghouse that can be purchased
2  //given the user's budget.
3  #include <iostream>
4  #include <cmath>
5  using namespace std;

6  int main( )
7  {
8      const double COST_PER_SQ_FT = 10.50;
9      double budget, area, lengthSide;

10     cout << "Enter the amount budgeted for your doghouse $";
11     cin >> budget;

12     area = budget/COST_PER_SQ_FT;
13     lengthSide = sqrt(area);
```

디스플레이 3.1 값을 리턴하는 사전 정의 함수 (2 of 2)

```
14     cout.setf(ios::fixed);
15     cout.setf(ios::showpoint);
16     cout.precision(2);
17     cout << "For a price of $" << budget << endl
18         << "I can build you a luxurious square doghouse\n"
19         << "that is " << lengthSide
20         << " feet on each side.\n";

21     return 0;
22 }
```

SAMPLE DIALOGUE

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

사전 정의 함수들

- `#include <cstdlib>`
 - 다음과 같은 함수를 포함:
 - `abs()` `// int` 형의 절대값을 리턴
 - `labs()` `// long int` 형의 절대값을 리턴
 - `*fabs()` `// float` 형의 절대값을 리턴
 - `*fabs()`은 실제로 `<cmath>` 라이브러리에 있음!
 - 혼란이 올 수 있다
 - C++이 탄생되고 라이브러리가 추가되는 과정 중에 발생
 - 부록 또는 매뉴얼을 참조할 것

수학 함수들

- `pow(x, y)`
 - x 를 y 승한 값을 리턴
double result, $x = 3.0$, $y = 2.0$;
result = `pow(x, y)`;
cout << result;
 - $3.0^{2.0} = 9.0$ 이므로 9.0을 출력
- 이 함수는 두 개의 인자를 가짐
 - 함수는 복수 개의 인자와 다양한 데이터 형을 가질 수 있다

디스플레이 3.2 사전 정의 함수 몇 가지 (1 of 2)

Display 3.2 Some Predefined Functions

NAME	DESCRIPTION	TYPE OF ARGUMENTS	TYPE OF VALUE RETURNED	EXAMPLE	VALUE	LIBRARY HEADER
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 <code>int</code>	int	int	abs(-7) abs(7)	7 7	cstdlib
labs	Absolute value for <code>long</code>	long	long	labs(-70000) labs(70000)	70000 70000	cstdlib
fabs	Absolute value for <code>double</code>	double	double	fabs(-7.5) fabs(7.5)	7.5 7.5	cmath

디스플레이 3.2 사전 정의 함수 몇 가지 (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 pro- gram	int	void	exit(1);	None	cstdlib
rand	Random number	None	int	rand()	Varies	cstdlib
srand	Set seed for rand	unsigned int	void	srand(42);	None	cstdlib

사전 정의 void 함수

- 값의 리턴이 없음
- “행위(action)”만 수행하고 “답(answer)”을 보내지는 않음
- 호출될 때, 식 자체가 문장이 됨
 - `exit(1);` // 값의 리턴이 없으므로 할당되지 않음
 - 프로그램을 종료
 - void 함수도 인자를 가질 수 있다
- 값을 리턴하는 함수와 동일한 형태
 - 단지 값을 리턴하지 않는다는 점만 다름!

난수 생성기 (Random Number Generator)

- “임의로 선택한” 숫자를 리턴
- 시뮬레이션(simulation) 또는 게임에 사용
 - rand()
 - 인자가 없다
 - 0 ~ RAND_MAX 사이의 값을 리턴
 - 범위 설정(Scaling)
 - 나머지 연산을 사용하여 리턴값의 범위를 축소
 - $\text{rand()} \% 6$
0~5 사이의 값을 리턴
 - 범위 이동(Shifting)
 $\text{rand()} \% 6 + 1$
 - 범위를 1 ~ 6 으로 이동 (예:주사위)

난수의 씨드(Seed)

- 의사 난수(Pseudorandom numbers)
 - rand()를 호출하면 동일한 순서로 난수가 발생됨
- “씨드(seed)”를 사용하여 순서를 변경
srand(seed_value);
 - void 함수
 - 하나의 인자를 씨드 값으로 받음
 - 시스템 시간을 포함하여 어떠한 값이라도 씨드로 사용 가능 :
srand(time(0));
 - time() 은 시스템의 시간을 숫자값으로 리턴
 - time() 함수는 라이브러리 <time> 에 포함

난수 예

- 0.0 ~ 1.0 사이의 double형 난수:
 $(\text{RAND_MAX} - \text{rand}()) / \text{static_cast<double>}(\text{RAND_MAX})$
 - double형 정밀도를 위하여 강제 형 변환 사용
- 1 ~ 6 사이의 int형 난수:
 $\text{rand}() \% 6 + 1$
 - “%”는 나머지 연산자(remainder)
- 10 ~ 20 사이의 int형 난수:
 $\text{rand}() \% 10 + 10$

프로그래머(사용자) 정의 함수

- 자신의 함수를 작성해보자!
- 프로그램의 빌딩 블록
 - 나눔(Divide) & 획득(Conquer)
 - 가독성(Readability)
 - 재사용(Re-use)
- 다음의 두 위치에 정의 가능:
 - main()과 같은 파일
 - 분리된 다른 파일에도 정의 가능

함수 사용의 요소

- 함수 사용의 세 부분 :
 - 함수의 선언(Declaration)/원형(Prototype)
 - 컴파일러를 위한 정보 전달
 - 호출을 적절하게 해석
 - 함수 정의
 - 함수의 기능에 대한 실제 구현 / 코드
 - 함수 호출
 - 함수에게 제어권 이양

함수의 선언

- “함수 원형” 이라고도 함
- 컴파일러를 위한 함수 정보의 선언
- 컴파일러에게 호출을 해석하는 방법에 대해 알려줌
 - 구문:
`<return_type> FnName(<formal-parameter-list>);`
 - 예:
`double totalCost(int numberParameter,
 double priceParameter);`
- 호출 전에 위치
 - main() 함수 공간 안에 선언
 - 또는 main() 함수 위의 전역 공간에 선언

함수 정의

- 함수의 구현
- main() 함수 구현과 동일함

- 예:

```
double totalCost(int numberParameter,  
                 double priceParameter)  
{  
    const double TAXRATE = 0.05;  
    double subTotal;  
    subtotal = priceParameter * numberParameter;  
    return (subtotal + subtotal * TAXRATE);  
}
```

- 들여쓰기 주의

함수 정의 위치

- main() 함수 뒤에 위치
 - main() 함수 안에는 정의 불가!
- 함수는 “평등하다” : 함수가 다른 함수의 부분이 될 수 없다
- 정의 내의 형식 매개변수(formal parameter)
 - 데이터를 내부로 보내기 위한 공간 확보자(Placeholders)
 - “변수의 이름”은 함수 정의 내에서 데이터의 제공에 사용
- 리턴 문
 - 호출자에게 데이터를 전달

함수 호출

- 사전 정의 함수의 호출과 같음
`bill = totalCost(number, price);`
- 회수: `totalCost` 는 `double` 형 값을 리턴
 - 이름이 “`bill`”인 변수에 값이 할당됨
- 인자 : `number, price`
 - 인자는 리터럴, 변수, 식 또는 이들의 조합 모두 가능
 - 함수의 호출에서, 인자를 “실 매개변수(actual arguments)”라고도 함
 - 전달되는 실제 데이터를 담고 있기 때문

디스플레이 3.5 난수 생성기를 사용하는 함수 (1 of 2)

Display 3.5 A Function Using a Random Number Generator

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


3  double totalCost(int numberParameter, double priceParameter);
4  //Computes the total cost, including 5% sales tax,
5  //on numberParameter items at a cost of priceParameter each.

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);
```

*Function declaration;
also called the function
prototype*



Function call



디스플레이 3.5 난수 생성기를 사용하는 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% sales tax
27     double subtotal;
28
29     subtotal = priceParameter * numberParameter;
30     return (subtotal + subtotal*TAXRATE);
31 }
```

Function
head

Function
body

Function
definition

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

선택적 함수의 선언

- 함수의 선언은 컴파일러를 위한 “정보”
- 컴파일러는 다음의 사항만이 필요:
 - 리턴 형
 - 함수 이름
 - 매개변수 리스트
- 형식 매개변수의 이름은 필요하지 않음 :
`double totalCost(int, double);`
 - 형식 매개변수의 이름을 삽입하는 이유
 - 가독성 향상

매개변수(Parameter) vs. 인자(Argument)

- 두 개의 용어는 자주 혼용된다
- 형식(formal) 매개변수(parameter)/인자(argument)
 - 함수의 선언에 사용
 - 함수 정의의 헤더에 사용
- 실(Actual) 매개변수(parameters)/인자(arguments)
 - 함수의 호출에 사용
- 기술적으로 매개변수(parameter)는 형식(formal) 부분이고 인자(argument)는 실(actual) 부분*
 - *항상 같은 방식으로 사용되지는 않는다

함수를 호출하는 함수

- 이미 하고 있다!
 - main() 함수도 함수다!
- 요구사항:
 - 함수의 선언이 먼저 나타나 있어야 한다
- 함수의 정의는 일반적으로 다음 부분에 존재
 - main()함수의 뒤에 정의
 - 또는 별도의 파일에
- 함수에서 다른 함수들을 호출하는 것은 일반적
- 함수가 자기 자신을 호출하는 경우도 가능 →
“재귀(Recursion)”

부울 리턴 형의 함수

- 리턴 형은 유효한 어떠한 형도 가능
 - 함수의 선언/원형에 주어진다:
`bool appropriate(int rate);`
 - 함수의 정의 :
`bool appropriate (int rate)`
`{`
`return (((rate>=10)&&(rate<20)) || (rate==0));`
`}`
 - “true” 또는 “false”를 리턴
 - 다른 함수에서 함수를 호출:
`if (appropriate(entered_rate))`
`cout << "Rate is valid\n";`

void 함수의 선언

- 값을 리턴하는 함수와 비슷함
- 리턴 형을 void로 표시
- 예:
 - Function declaration/prototype:
void showResults(double fDegrees,
double cDegrees);
 - 리턴 형은 “void”
 - 아무것도 리턴하지 않음

void 함수의 선언

- 함수의 정의:

```
void showResults(double fDegrees, double cDegrees)
{
    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(1);
    cout    << fDegrees
            << " degrees fahrenheit equals \n"
            << cDegrees << " degrees celsius.\n";
}
```

- 리턴 문이 없음에 주의
 - 함수를 위해 사용 가능

void 함수의 호출

- 사전 정의된 void 함수의 호출과 동일
- main()와 같이 다른 함수로부터 호출 :
 - `showResults(degreesF, degreesC);`
 - `showResults(32.5, 0.3);`
- 리턴값이 없으므로 할당되는 부분이 없음에 주의
- 실 매개변수 (`degreesF, degreesC`)
 - 함수로 전달
 - 함수는 값을 전달받아 자신이 할 일을 수행

void 함수의 리턴문

- 호출한 함수에게 제어권을 이양
 - 리턴 형이 있는 함수의 경우 반드시 리턴문이 있어야 한다
 - 일반적으로 함수 정의의 맨 마지막에 위치
- void 함수의 리턴문은 선택적
 - }가 함축적으로 void 함수로부터 제어권을 넘김 (리턴문의 역할)

선행조건(Preconditions)과 사후조건(Postconditions)

- “I-P-O” 와 유사함
- 함수 선언부의 주석:
void showInterest(double balance, double rate);
// 선행 조건 (Precondition): balance 는 음수가 아닌 계좌 잔액
// rate 는 %로 표시되는 이자율
//사후조건(Postcondition): 주어진 이자율로 주어진 잔액에 대한
// 이자를 화면에 출력,
- 종종 입력과 출력으로 불리기도 함

main() 함수 : “특별하다”

- main() 또한 함수
- 특별한 점:
 - main() 함수는 프로그램에서 단 하나만 존재
- 누가 main() 함수를 호출하는가?
 - 운영체제(Operating system)
 - 관례적으로 반드시 리턴문이 존재
 - 호출자에게 값을 리턴 → 운영체제(operating system)
 - “int” or “void” 형으로 리턴

영역 규칙(Scope Rules)

- 지역 변수
 - 주어진 함수의 바디(body) 안에 선언
 - 함수 내에서만 사용 가능
- 다른 함수 안에 같은 이름을 가지는 변수가 존재 가능
 - 영역은 지역 : 함수가 변수의 영역이 된다
- 지역 변수 우선
 - 각각의 데이터 제어를 유지
 - 근거지를 알아야 한다
 - 함수 자신의 역할 수행에 필요한 지역 데이터가 필요하면 함수 내에서 변수가 선언되어야 한다

프로시저 추상화 (Procedural Abstraction)

- 함수가 “무엇”이 필요한지는 알아야 하지만 “어떻게” 동작하는지는 몰라도 된다!
- “블랙박스(black box)”와 같이 생각함
 - 사용 방법은 알지만 어떻게 동작하는 지는 모른다
- 블랙박스과 같이 구현
 - 함수의 사용이 필요하면 : 선언
 - 함수의 정의부분은 필요 없다
 - “정보의 은닉” 으로 부른다
 - 함수가 어떻게 동작하는지 자세한 과정은 숨겨짐

전역 상수(Global Constants)와 전역 변수(Global Variables)

- 함수 바디의 외부에 선언
 - 해당 파일 안의 모든 함수에게 전역
- 함수 바디의 내부에 선언
 - 함수의 지역 변수
- 상수의 전역 선언 : 전역 상수(External Constants)
 - `const double TAXRATE = 0.05;`
 - 전역으로 선언되었으므로 모든 함수가 영역을 가진다
- 전역 변수?
 - 가능하지만 잘 사용하지는 않음
 - 위험 : 사용시 제어가 불가능!

블록

- 복합문 내부에 선언된 변수
 - “블록” 이라 부름 : 복합문 영역에서만 사용되는 지역 변수
 - “블록 영역”을 가진다
- 모든 함수의 정의는 블록!
 - 지역 함수 영역을 제공한다
- 루프 블록:

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

 - 변수 `ctr` 은 루프 바디 블록의 영역만을 가짐

중첩된 영역(Nested Scope)

- 다중 블록(중첩된 블록) 안에서 각 블록마다 선언된 같은 이름의 변수들 : 각 변수는 서로 다른 변수!
- 매우 정당한 방법 : 영역은 “블록 영역”
 - 모호하지 않음
 - 각각의 변수의 이름은 영역 내에서 구분됨
 - 각 변수들은 선언된 블록 안에서만 영향을 받음

요약 1

- 두 종류의 함수:
 - 값을 리턴하는 함수와 값의 리턴이 없는(void) 함수
- 함수는 “블랙박스”
 - “어떻게” 동작하는지는 숨겨짐
 - 자신의 지역 변수 선언
- 함수의 선언에는 자신의 문서가 있어야 한다
 - 주석 내에 사전 및 사후 조건이 제공되어야 함
 - 모든 호출자가 필요시 제공되어야 함

요약 2

- 지역 데이터
 - 함수 정의 안에서 선언
- 전역 데이터
 - 함수 정의 이전에 선언
 - 상수는 좋지만 변수는 바람직하지 않다
- 매개 변수(Parameters)/인자(Arguments)
 - 형식(formal): 함수의 선언과 정의에서 사용
 - 입력 데이터의 공간 확보자(placeholder)
 - 실(Actual): 함수의 호출
 - 실 데이터가 함수에 전달됨

Q&A