Chapter 4. 매개변수와 오버로딩

박 종 혁 교수
UCS Lab
(http://www.parkjonghyuk.net)

Tel: 970-6702
Email: jhpark1@snut.ac.kr

Learning Objectives

- 매개변수
  - Call-by-value
  - Call-by-reference
  - 혼합 매개변수 리스트
- 오버로딩과 디폴트 인자
  - 예와 규칙
- 테스트와 디버깅 함수
  - assert 매크로(Macro)
  - 스터브(Stubs), 드라이버(Drivers)

매개변수 (Parameters)

- 매개변수로 인자(arguments)를 전달하는 두 가지 방법
- Call-by-value
  - 값이 "복사"되어 전달
- Call-by-reference
  - 실인자(actual arguments)의 "주소"가 전달

Call-by-Value 매개변수

- 실인자(actual arguments)의 복사본이 전달
- 함수 내의 "지역변수"와 같이 취급한다
- 만약 값을 수정하면 전달된 복사본 만이 변경
  - 함수는 실인자에 접근이 불가
- 기본적인 방법
  - 이후의 예에서 확인

디스플레이 4.1 지역변수로 사용된 형식 매개변수(Formal Parameter) (1 of 3)

Display 4.1 Formal Parameter Used as a Local Variable

```
1  //Law office billing program.
2  #include <iostream>
3  using namespace std;

4  const double RATE = 150.00; //Dollars per quarter hour.

5  double fee(int hoursWorked, int minutesWorked);
6  //Returns the charges for hoursWorked hours and
7  //minutesWorked minutes of legal services.

8  int main( )
9  {
10     int hours, minutes;
11     double bill;
```

디스플레이 4.1 지역변수로 사용된 형식 매개변수(Formal Parameter) (2 of 3)

```
12        cout << "Welcome to the law office of\n"
13             << "Dewey, Cheatham, and Howe.\n"
14             << "The law office with a heart.\n"
15             << "Enter the hours and minutes"
16             << " of your consultation:\n";
17        cin >> hours >> minutes;

18        bill = fee(hours, minutes);

19        cout.setf(ios::fixed);
20        cout.setf(ios::showpoint);
21        cout.precision(2);
22        cout << "For " << hours << " hours and " << minutes
23             << " minutes, your bill is $" << bill << endl;

24        return 0;
25    }
```

The value of `minutes` *is not changed by the call to* `fee`.

(continued)

디스플레이 4.1 지역변수로 사용된 형식 매개변수(Formal Parameter) (3 of 3)

Display 4.1 Formal Parameter Used as a Local Variable

```
26  double fee(int hoursWorked, int minutesWorked)
27  {
28      int quarterHours;

29      minutesWorked = hoursWorked*60 + minutesWorked;
30      quarterHours = minutesWorked/15;
31      return (quarterHours*RATE);
32  }
```

`minutesWorked` *is a local variable initialized to the value of* `minutes`.

SAMPLE DIALOGUE

Welcome to the law office of
Dewey, Cheatham, and Howe.
The law office with a heart.
Enter the hours and minutes of your consultation:
5 46
For 5 hours and 46 minutes, your bill is $3450.00

Call-by-Value 함정

- 일반적인 실수:
  - 함수 내에서 매개변수를 중복해서 선언하는 경우:

```
double fee(int hoursWorked, int minutesWorked)
{
  int quarterHours;          // local variable
  int minutesWorked          // NO!
}
```

  - 컴파일 에러의 결과
    - "Redefinition error..."
- Call-by-value 매개변수는 "지역변수"와 같다
  - 하지만 함수는 매개변수를 지역변수로 "자동으로" 사용함(중복선언 NO!)

Call-By-Reference 매개변수

- 호출자(함수를 호출한 주체)가 실인자에 직접 접근할 수 있게 한다
- 호출자의 데이터는 호출된 함수에 의해 수정이 가능해짐!
- 일반적으로 입력 함수에 사용
  - 데이터의 갱신
  - 호출자에게 전달
- 형식 매개변수 리스트의 형 다음에 &(ampersand)를 표시

디스플레이 4.2 Call-by-Reference 매개변수 (1 of 3)

Display 4.2 Call-by-Reference Parameters

```
1  //Program to demonstrate call-by-reference parameters.
2  #include <iostream>
3  using namespace std;

4  void getNumbers(int& input1, int& input2);
5  //Reads two integers from the keyboard.

6  void swapValues(int& variable1, int& variable2);
7  //Interchanges the values of variable1 and variable2.

8  void showResults(int output1, int output2);
9  //Shows the values of variable1 and variable2, in that order.

10 int main( )
11 {
12     int firstNum, secondNum;

13     getNumbers(firstNum, secondNum);
14     swapValues(firstNum, secondNum);
15     showResults(firstNum, secondNum);
16     return 0;
17 }
```

디스플레이 4.2 Call-by-Reference 매개변수 (2 of 3)

```
18  void getNumbers(int& input1, int& input2)
19  {
20      cout << "Enter two integers: ";
21      cin >> input1
22          >> input2;
23  }

24  void swapValues(int& variable1, int& variable2)
25  {
26      int temp;

27      temp = variable1;
28      variable1 = variable2;
29      variable2 = temp;
30  }
31
32  void showResults(int output1, int output2)
33  {
34      cout << "In reverse order the numbers are: "
35           << output1 << " " << output2 << endl;
36  }
```

디스플레이 4.2 Call-by-Reference 매개변수 (3 of 3)

Display 4.2 Call-by-Reference Parameters

SAMPLE DIALOGUE

```
Enter two integers: 5 6
In reverse order the numbers are: 6 5
```

Call-By-Reference 상세

- 실제로 무엇이 전달되어지는가?
- 호출자에게 실인자의 "참조(reference)"를 전달!
  - 실인자의 메모리 위치를 참조
  - 메모리 상의 유일한 참조 가능한 숫자를 "주소(address)"라고 한다

상수 참조 매개변수

- 참조 매개변수는 “위험”하다
  - 호출자의 데이터가 변경될 수 있다
  - 종종 필요하지만, 필요하지 않을 때도 있다
- 데이터를 보호하면서 참조로 전달하고 싶을 경우:
  - const 키워드를 사용
    - void sendConstRef(const int &par1,
      const int &par2);
    - 매개변수에 대한 함수의 권한을 “읽기 전용”으로 한정
    - 함수 바디(function body)내부에서 데이터값의 변경을 허용하지 않음

매개변수(Parameters) 와 인자(Arguments)

- 용어의 혼돈, 종종 구분되지 않고 사용된다
- 실제 의미들:
  - 형식 매개변수(Formal parameters)
    - 함수의 선언 과 정의 내에서 사용
  - 인자 또는 인수(Arguments)
    - 형식 매개변수를 채우는 데 사용
    - 함수의 호출에서 사용됨 (인자 리스트)
  - Call-by-value & Call-by-reference
    - 단순히 프로세스에서 사용되는 매커니즘

혼합 매개변수 리스트

- 전달 매커니즘의 혼용이 가능
- 매개변수 리스트는 값에 의한 전달과 참조에 의한 전달 매개변수 모두 포함 가능
- 리스트 내의 인자의 순서가 중요하다:
  void mixedCall(int & par1, int par2, double & par3);
  - 함수의 호출:
    mixedCall(arg1, arg2, arg3);
    - arg1 은 반드시 정수형(int), 참조에 의해 전달
    - arg2 은 반드시 정수형(int), 값에 의해 전달
    - arg3 은 반드시 실수형(double), 참조에 의해 전달

형식 매개변수 이름의 선택

- 식별자와 같은 명명 규칙:
  - 의미있는 이름!
- 함수는 "독립적인 모듈"이다
  - 프로그램의 나머지 부분과 분리되어 설계
  - 프로그래머의 팀에게 할당된다
  - 모든 사람이 적절한 함수의 사용법을 "이해"해야 한다
  - 형식 매개변수의 이름이 인자의 이름과 같다면 -> OK!
- 함수의 이름도 같은 방법으로 명명

오버로딩(Overloading)

- 함수 이름은 같다
- 매개변수의 리스트가 다르다!
- 두 개의 분리된 함수 정의
- 함수 “시그니처(signature)"
  - 함수 이름 & 매개변수 리스트
  - 각 함수 정의마다 함수 시그니처는 유일해야 한다
- 다른 데이터에 대해 같은 작업을 허용한다

오버로딩 예: Average 함수

- 함수는 2개 수의 평균을 계산:

```
double average(double n1, double n2)
{
        return ((n1 + n2) / 2.0);
}
```

- 지금은 3개 수의 평균을 계산:

```
double average(double n1, double n2, double n3)
{
        return ((n1 + n2 + n3) / 3.0);
}
```

- 같은 이름의 2개의 함수

오버로딩 된 Average 함수

- 어떤 함수가 호출되는가?
- 함수 자신의 호출에 의존적:
  - avg = average(5.2, 6.7);
    - 2개의 매개변수를 가진 average 함수 호출
  - avg = average(6.5, 8.5, 4.2);
    - 3개의 매개변수를 가진 average 함수 호출
- 컴파일러는 함수 시그니처에 근거를 두고 함수를 호출
  - 적당한 함수가 호출과 “매칭”되는지
  - 각각은 분리된 함수로 고려된다

오버로딩 함정

- "같은 일"을 수행하는 함수만이 오버로딩 된다
  - mpg() 함수는 모든 오버로딩에서 같은 일을 수행해야 한다
  - 그렇지 않으면, 예기치 못한 결과가 도출
- C++ 함수 호출 해결:
  - 1st: 정확한 시그니처 검색
  - 2nd: "호환되는(compatible)" 시그니처 검색

오버로딩 해결

- 1st: 정확한 매칭
  - 정확한 시그니처를 찾는다
    - 인자의 변경은 허가되지 않음
- 2nd: 호환되는 매칭
  - 자동 형 변환이 가능한 호환되는 시그니처를 찾는다:
    - 1st 승급(e.g., int→double)
      - 데이터의 손실 없음
    - 2nd 강등 (e.g., double→int)
      - 데이터의 손실 가능성 있음

오버로딩 해결 예

- 다음의 주어진 함수:
  - 1. void f(int n, double m);
    2. void f(double n, int m);
    3. void f(int n, int m);
  - 함수의 호출:
    f(98, 99); → Calls #3
    f(5.3, 4); → Calls #2
    f(4.3, 5.2); → Calls ???
- 이와 같이 혼란스러운 오버로딩은 피해야 한다

자동 형 변환과 오버로딩

- 숫자 형식 매개변수는 대체로 double 형으로 변환
- 어떠한 숫자 형식도 허용
  - 어떠한 하위의 데이터로 자동으로 승급
    - int → double
    - float → double
    - char → double *More on this later!
- 다른 숫자 형에 대한 오버로딩은 피해야 한다!

자동 형 변환과 오버로딩 예

- double mpg(double miles, double gallons)
  {
  return (miles/gallons);
  }
- 함수 호출의 예:
  - mpgComputed = mpg(5, 20);
    - 5 & 20은 double 형으로 변환된 후 전달
  - mpgComputed = mpg(5.8, 20.2);
    - 변환이 필요없다
  - mpgComputed = mpg(5, 2.4);
    - 5는 double 형으로 변환된 후 전달

디폴트 인자

- 매개변수의 생략을 허용
- 함수의 선언/원형에 명기
  - void showVolume(int length,
    int width = 1,
    int height = 1);
    - 마지막 2개의 인자는 디폴트 인자
  - 가능한 호출:
    - showVolume(2, 4, 6); //모든 인자 제공
    - showVolume(3, 5); //height는 디폴트 값 1을 가짐
    - showVolume(7); //width & height는 디폴트 값 1을 가짐

Display 4.8 디폴트 인자 (1 of 2)

Display 4.8 Default Arguments

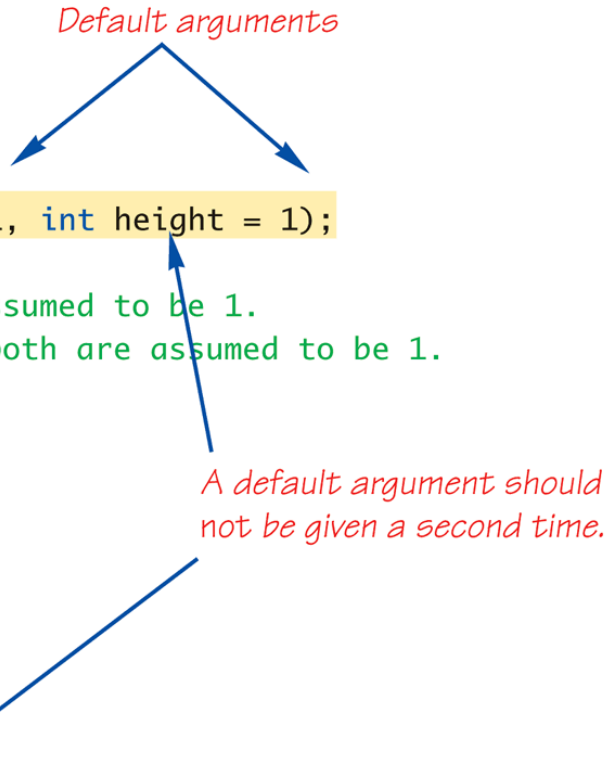
```
1
2   #include <iostream>
3   using namespace std;

4   void showVolume(int length, int width = 1, int height = 1);
5   //Returns the volume of a box.
6   //If no height is given, the height is assumed to be 1.
7   //If neither height nor width is given, both are assumed to be 1.

8   int main( )
9   {
10      showVolume(4, 6, 2);
11      showVolume(4, 6);
12      showVolume(4);

13      return 0;
14  }

15  void showVolume(int length, int width, int height)
```



Display 4.8 디폴트 인자 (2 of 2)

```
16  {
17      cout << "Volume of a box with \n"
18           << "Length = " << length << ", Width = " << width << endl
19           << "and Height = " << height
20           << " is " << length*width*height << endl;
21  }
```

SAMPLE DIALOGUE

```
Volume of a box with
Length = 4, Width = 6
and Height = 2 is 48
Volume of a box with
Length = 4, Width = 6
and Height = 1 is 24
Volume of a box with
Length = 4, Width = 1
and Height = 1 is 4
```

테스트와 디버깅 함수

- 많은 방법:
  - cout 문 이용 – 중간 결과를 출력
    - 호출과 정의 부분
    - 실행의 “추적(trace)”에 사용
  - 컴파일러 디버거
    - 디버거 환경에 의존적
  - assert 매크로(Macro)
    - 필요시 조기 종료
  - 스터브(Stubs) and 드라이버(drivers)
    - 점진적 개발

assert 매크로(Macro)

- 가정(Assertion): 참 또는 거짓 문
- 정확성의 기록 및 체크에 사용
  - 선행조건 & 사후조건
    - 일반적인 assert 사용: 조건의 검증
  - 구문:
    assert(<assert_condition>);
    - 리턴값이 없다 – void 함수처럼 사용
    - assert_condition 평가
    - 거짓이면 종료, 참이면 계속 실행
- 사전정의 라이브러리 <cassert>
  - 매크로는 함수와 비슷하게 사용된다

assert 매크로 예

- 주어진 함수의 선언:
  void computeCoin(int coinValue,
  int& number,
  int& amountLeft);
  //선행조건: 0 < coinValue < 100
  0 <= amountLeft <100
  //사후조건: 최대 동전 개수와 같게 설정
- 선행 조건 체크:
  - assert ((0 < currentCoin) && (currentCoin < 100)
    && (0 <= currentAmountLeft) && (currentAmountLeft < 100));
  - 선행조건이 충족되지 않으면 → 조건은 거짓 → 프로그램 수행이 종료!

assert 매크로 예

- 디버깅에 사용 가능
- 실행이 종료되고 프로그램을 조사

assert On/Off

- 전처리기가 방법을 제공
- #define NDEBUG
  #include <cassert>
- #include 라인 이전의 “#define” 라인 추가
  - 프로그램에서 모든 assert를 off 시킨다
- “#define” 라인을 제거(또는 주석처리)
  - 모든 assert를 on 시킨다

스터브(Stubs)와 드라이버(Drivers)

- 분리 컴파일 단위
  - 각각의 함수는 분리되어 설계, 코딩, 테스트 된다
  - 각 단위의 유효성을 확실하게 한다
  - 분할(Divide)과 정복(Conquer)
    - 하나의 커다란 작업 → 작게 분할, 작업을 관리
- 하지만 어떻게 독립적으로 테스트할 것인가?
  - 드라이버 프로그램

드라이버 프로그램 예: 디스플레이 4.9 드라이버 프로그램 (1 of 3)

Display 4.9 Driver Program

```
1
2   //Driver program for the function unitPrice.
3   #include <iostream>
4   using namespace std;

5   double unitPrice(int diameter, double price);
6   //Returns the price per square inch of a pizza.
7   //Precondition: The diameter parameter is the diameter of the pizza
8   //in inches. The price parameter is the price of the pizza.

9   int main( )
10  {
11      double diameter, price;
12      char ans;

13      do
14      {
15          cout << "Enter diameter and price:\n";
16          cin >> diameter >> price;
```

드라이버 프로그램 예: 디스플레이 4.9 드라이버 프로그램 (2 of 3)

```
17          cout << "unit Price is $"
18               << unitPrice(diameter, price) << endl;

19          cout << "Test again? (y/n)";
20          cin >> ans;
21          cout << endl;
22      } while (ans == 'y' || ans == 'Y');

23      return 0;
24  }
25
26  double unitPrice(int diameter, double price)
27  {
28      const double PI = 3.14159;
29      double radius, area;

30      radius = diameter/static_cast<double>(2);
31      area = PI * radius * radius;
32      return (price/area);
33  }
```

(continued)

드라이버 프로그램 예: 디스플레이 4.9 드라이버 프로그램 (3 of 3)

Display 4.9 **Driver Program**

SAMPLE DIALOGUE

Enter diameter and price:
13 14.75
`Unit price is: $0.111126`
Test again? (y/n): **y**

Enter diameter and price:
2 3.15
`Unit price is: $1.00268`
Test again? (y/n): **n**

스터브(Stubs)

- 점진적인 개발
- 첫째로 함수의 “개요”를 작성
  - 하위 수준의 완벽한 구현은 나중에 수행
  - 함수의 세부적인 기능은 아직 구현중
  - 예:

```
double unitPrice(int diameter, double price)
{
    return (9.99);// not valid, but noticeably
                                // a "temporary" value
}
```

  - 위의 함수는 아직 구현중

기본 테스트 규칙

- 정확한 프로그램 작성
- 에러(버그)를 최소화
- 데이터의 검증을 확실히 하라
  - 모든 함수는 그 프로그램에 있는 모든 다른 함수들이 이미 완전하게 테스트되고 디버그된 환경에서 테스트가 이루어져야 한다
  - 연속된 에러와 충돌을 피해야 한다

요약 1/2

- 형식 매개변수는 공간확보자(placeholder)이고, 함수의 호출시에는 실 인자로 채워진다
- Call-by-value 매개변수는 함수의 바디 내로 "복사"되어 전달
  - 실 인자는 수정될 수 없다
- Call-by-reference 매개변수는 실 인자의 메모리 주소를 전달
  - 실 인자는 수정될 수 있다
  - 인자는 반드시 변수이어야 한다. 상수는 불가

요약 2/2

- 같은 함수의 이름을 다중적으로 정의 : 오버로딩
- 디폴트 인자는 인자 리스트에서의 인자의 생략을 허용하게 한다
  - 인자가 생략되면 → 기본값이 할당된다
- assert 매크로는 조건이 거짓이면 프로그램을 종료한다
- 함수는 반드시 독립적으로 테스트되어야 한다
  - 드라이버 프로그램을 사용하여 분리컴파일 단위로 테스트한다

Q&A