제4장

매개변수와 오버로딩

학습 목표

- 매개변수
  - Call-by-value 매개변수
  - Call-by-reference 매개변수
  - 혼합된 매개변수 리스트
- 오버로딩과 디폴트 인자
  - 예제, 규칙
- 테스팅과 디버깅 함수
  - assert 매크로
  - 스터브, 드라이버

매개변수

- 매개변수로 인자를 전달하는 2가지 방법
- Call-by-value 매개변수
  - 값의 “복사본”을 전달. 함수 내에서 “지역 변수”로 사용
  - 수정되면, “지역 변수”만 변경
  - 함수는 호출 함수의 “실제 인자”에 접근 불가
  - Call-by-Value 매개변수는 디폴트 방법

- Call-by-reference 매개변수
  - 실제 인자의 “주소”를 전달

Call-By-Reference 매개변수

- 호출한 함수의 실제 인자에 접근을 제공하기 위해 사용된다.
- 호출한 함수의 데이터는 호출된 함수가 수정할 수 있다!
- 전형적으로 입력 함수에 사용된다.
  - 호출한 함수를 위해 데이터를 검색한다.
  - 데이터가 호출한 함수에 주어진다.
- 형식 매개변수 리스트에서 매개변수 형식 뒤에 붙은 &로 구분

Call-by-Reference 매개변수 (1 of 2)

디스플레이 4.2 **call-by-reference 매개변수**

```
1  //call-by-reference 매개변수를 설명하는 프로그램.
2  #include <iostream>
3  using namespace std;

4  void getNumbers(int& input1, int& input2);
5  //키보드에서 정수 2개를 읽는다.

6  void swapValues(int& variable1, int& variable2);
7  //variable1과 variable2의 값을 상호 교환한다.

8  void showResults(int output1, int output2);
9  // variable1과 variable2의 값을을 이 순서로 보여 준다.

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

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

Call-by-Reference 매개변수 (2 of 2)

```
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 매개**

SAMPLE DIALOGUE

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

Call-By-Reference 매개변수 세부 사항

- 무엇이 실제로 전달되나?
- 호출한 함수의 실제 인자에 되돌려지는 “참조”
  - 실제의 메모리 위치를 참조
  - 메모리인자에서 구별되는 위치를 참조하는 유일한 숫자인 호출된 “주소”

상수 참조 매개변수

- 참조 인자는 본질적으로 “위험하다”.
  - 호출된 데이터는 변경될 수 있다.
  - 이 방법이 종종 바람직하지만, 가끔은 그렇지 않다.

- 데이터를 “보호”하고, 그리고 여전히 참조로 전달하기 위해

```
void sendConstRef(const int &par1, const int &par2);
```

  - 함수에 의해 인자를 “읽기 전용”으로 만든다.
  - 함수 몸체 안에서 변경을 허용하지 않는다.
  - 주로, 큰 메모리를 사용하는 class 값을 인자로 전달하는 경우 사용
    - C++에서는 클래스 변수가 동적할당되지 않아도 됨.

매개변수와 인자

- 종종 바꿔서 사용하는 혼동되는 용어
- 정확한 의미:
  - 형식 매개변수
    - 함수 선언과 함수 정의 내에서 사용
  - 인자
    - 형식 매개변수를 “채우기” 위해 사용
    - 함수 호출(인자 리스트)에서 사용
  - Call-by-value 매개변수와 Call-by-reference 매개변수
    - 할당 과정에서 사용되는 “기법”

혼합된 매개변수 리스트

- 전달 기법을 혼합할 수 있다.
- 매개변수 리스트는 pass-by-value 매개변수와 pass-by-reference 매개변수를 포함할 수 있다.
- 리스트에 있는 인자의 순서는 중요하다:

  void mixedCall(int & par1, int par2, double & par3);

  - 함수 호출:

    mixedCall(arg1, arg2, arg3);

    - arg1은 참조로 전달되는 integer형
    - arg2는 값으로 전달되는 integer 형
    - arg3은 참조로 전달되는 double 형

형식 매개변수의 이름 정하기

- 식별자를 명명하는 것과 같은 규칙:
  - 의미 있는 이름!
- "독립되고 내용이 완비된 모듈"로서의 함수
  - 프로그램으로부터 독립 설계
  - 프로그래머 팀들에게 공지
  - 모든 사람들은 적합한 함수 사용을 "이해"해야 한다.
  - 형식 매개변수 이름이 인자 이름과 같으면 OK
- 같은 규칙으로 함수의 이름 정하기

오버로딩

- 같은 함수 이름
- 다른 매개변수 리스트
- 2개의 독립된 함수 정의
- 함수 “시그니처"
  - 함수 이름과 매개변수 리스트
  - 각 함수 정의에 “유일”해야 한다.
- 같은 작업을 다른 데이터에 대해 수행하는 것을 허용한다.

- 컴파일러가 함수 호출의 시그니처에 기반하여 호출 해결
  - 호출을 적당한 함수와 “일치”시킨다.
  - 각각을 독립된 함수로 고려한다.

오버로딩 예제: Average

- 함수가 2개 숫자의 평균을 계산한다:

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

- 이제 3개 숫자의 평균을 계산한다:

```
double average(double n1, double n2, double n3)
{
        return ((n1 + n2) / 2.0);
}
```

- 2개의 함수의 이름은 같다.

오버로딩 함정

- “같은 작업”을 하는 함수만을 오버로딩한다.
  - Mpg() 함수는 모든 오버로딩에 대해서 항상 같은 작업을 수행해야 한다.
  - 그렇지 않으면, 예상하지 못한 결과를 가져온다.
- C++ 함수 호출 해결:
  - 1st: 일치하는 시그니처를 찾는다.
  - 2nd: “호환이 되는" 시그니처를 찾는다.

디폴트 인자

- 몇몇 인자가 누락되는 것을 허용한다.
- 함수 선언/프로토타입에서 명시된다.
  - void showVolume(int length, int width = 1, int height = 1);
    - 마지막 2개의 인자가 디폴트이다.
  - 가능한 호출:
    - showVolume(2, 4, 6); //모든 인자가 주어졌다.
    - showVolume(3, 5); //높이가 디폴트로 1이다.
    - showVolume(7); //넓이와 높이가 디폴트로 1이다.

디폴트 인자

디스플레이 4.8 디폴트 인자

```
1
2   #include <iostream>
3   using namespace std;
4   void showVolume(int length, int width = 1, int height = 1);
5    //상자의 부피를 반환한다.
6    //높이가 주어지지 않으면 높이를 1로 가정한다.
7    //높이도 폭도 주어지지 않으면 둘 다 1로 가정한다.
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)
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  }
```

디폴트 인자

디폴트 인자는 두 번 주어지지 않는다.

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
```

테스팅과 디버깅 함수

- 많은 방법들:
  - 많은 cout 문
    - 호출과 정의에서 사용한다.
    - 실행을 “추적”하기 위해서 사용한다.
  - 컴파일러 디버거
    - 환경에 의존적이다.
  - assert 매크로
    - 필요하면 빠르게 종료한다.
  - 스터브와 드라이버
    - 점진적 개발

assert 매크로

- 가정: 참 혹은 거짓인 문장
- 정확성을 기록하고 체크하기 위해 사용한다.
  - 선행조건과 사후조건
    - 전형적인 assert 사용: 유효성을 확인한다.
  - 구문:

    assert(<assert_condition>);
    - 반환값이 없다.
    - assert_condition을 평가한다.
    - 거짓이면 종료하고 참이면 계속한다.
- 라이브러리 <cassert>에 사전 정의되어 있다.
  - 매크로는 함수와 비슷하게 사용된다.

assert 매크로 예제

- 주어진 함수 정의:

```
void computeCoin(int coinValue, int& number, int& amountLeft);
//선행조건: 0 < coinValue < 100  0 <= amountLeft <100
//사후조건: number set to max. number of coins
{
    assert ((0 < currentCoin) && (currentCoin < 100)
  && (0 <= currentAmountLeft) && (currentAmountLeft < 100));
}
```

- 선행조건 검사:
  - 선행조건을 만족하지 않으면 → 조건은 거짓이다 → 프로그램 실행이 종료된다!
- 디버깅에 유용하다.
- 문제가 생기면 실행을 중지한다.

assert On/Off

- 전처리기가 수단을 제공한다.

```
#define NDEBUG
#include <cassert>
```

- #include 전에 #define을 추가한다.
  - 프로그램 전체에 있는 모든 assertion을 멈춘다.
- #define을 제거한다. (또는 주석 처리한다.)
  - assertion이 다시 동작한다.

```
#define NDEBUG
#include <cassert>
float getSquareRoot (float t)
{
      assert (t >= 0);
      return sqrt(t);
 }
```

스터브와 드라이버

- 독립된 컴파일 단위
  - 각 함수를 독립적으로 설계하고, 코딩하고, 테스트한다.
  - 각 단위의 유효성을 보장한다.
  - 분할정복법(Divide & Conquer)
    - 하나의 큰 작업을 → 처리할 수 있는 더 작은 작업들로 변환한다.
- 그러나 어떻게 독립적으로 테스트하나?
  - 드라이버 프로그램

드라이버 프로그램 예제 (1 of 2)

디스플레이 4.9 드라이버 프로그램

```
1
2   //함수 unitPrice를 위한 드라이버 프로그램.
3   #include <iostream>
4   using namespace std;

5   double unitPrice(int diameter, double price);
6   //피자의 제곱 인치당 가격을 반환한다.
7   //선행조건: 매개변수 diameter는 인치단위의 피자 직격이다.
8   //매개변수 price는 피자의 가격이다.

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 예제 (2 of 2)

```
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  }
```

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**

(continued)

스터브

- 점진적으로 개발
- “전체적인 상황(big-picture)”을 알 수 있는 함수를 먼저 작성한다.
  - 하위 단계 함수를 나중에 작성한다.
  - 구현할 때까지 함수를 “꺼 놓는다(stub-out)”.
  - 예제:

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

  - 함수에 대한 호출은 여전히 “동작”한다.

기본적인 테스팅 규칙

- “정확한” 프로그램을 작성한다.
- “버그” 오류를 최소화한다.
- 데이터의 유효성을 보장한다.
  - 모든 함수는 그 프로그램에 있는 모든 다른 함수가 이미 완전히 테스트되고 디버그된 프로그램에서 테스트한다.
  - “연속적인 오류”와 모순되는 결과를 피하라.

요약

- 형색 매개변수는 함수 호출에서 실제 인자로 채워지는 용기.
- Call-by-value 매개변수는 호출된 함수 몸체에서 “지역 복사본”.
  - 실제 인자는 수정될 수 없다.
- Call-by-reference 매개변수는 실제 인자의 메모리 주소를 전달
  - 실제 인자는 수정될 수 있다.
  - 인자는 상수가 아닌 변수이어야 한다.
- 같은 이름을 가진 함수의 다중 정의가 가능: 오버로딩
- 디폴트 인자는 함수 호출에서 리스트의 뒤쪽 인자 누락 허용
- assert 매크로는 가정이 거짓이면 프로그램을 종료시킨다.
- 함수는 독립적으로 테스트되어야 한다.
  - 드라이버를 가지고 독립적인 컴파일 단위로