객체지향프로그래밍 - 과제7

2019110634 컴퓨터공학과 이창렬

A. Code explanation & output analysis (Write the source code and results)

A-1. Listing 10.9

```
#include <iostream>
// Draws a bar n segments long
// using iteration.
void segments1(int n) {
  while (n > 0) {
    std::cout << "*";
    n--;
  }
  std::cout << '\n';
}
// Draws a bar n segments long
// using recursion.
void segments2(int n) {
  if (n > 0) {
    std::cout << "*";
    segments2(n - 1);
  }
  else
    std::cout << '\n';
}
int main() {
  segments1(3);
  segments1(10);
  segments1(0);
  segments1(5);
  std::cout << "------------\n";
  segments2(3);
  segments2(10);
  segments2(0);
  segments2(5);
}
```

```
***
**********

*****
-------------
***
**********

*****
```

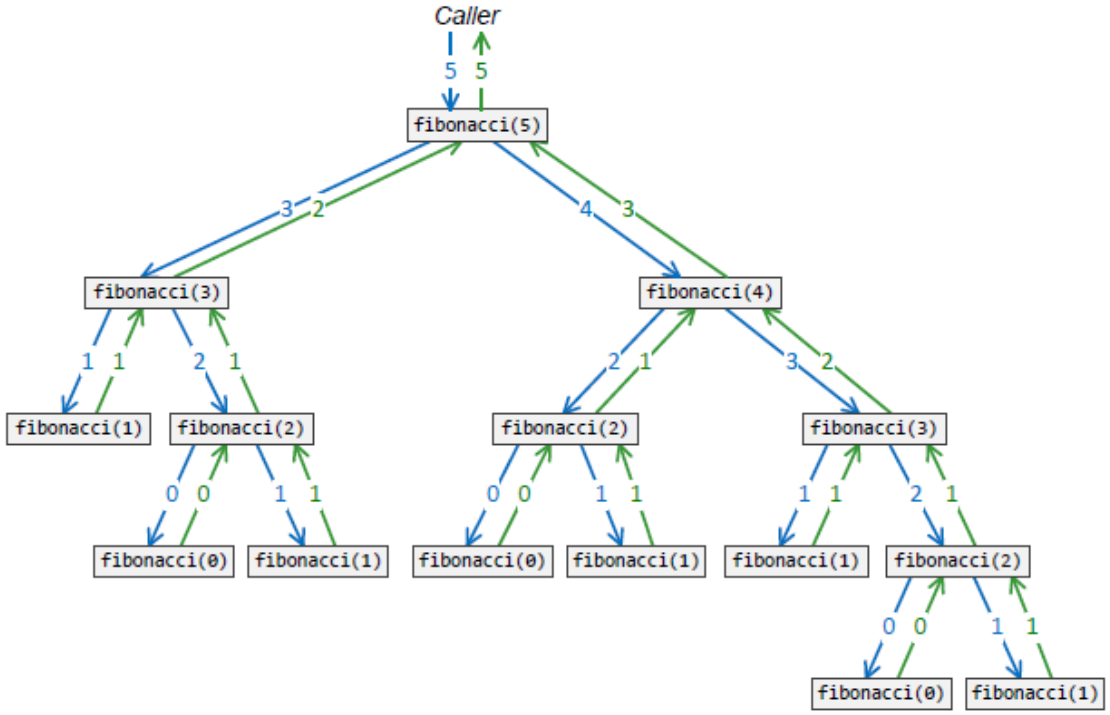
segments1에서는 0보다 큰 숫자를 입력하면 그 입력한 값만큼 *이 찍히고 줄이 바뀝니다. 0이하의 숫자가 입력되면 그냥 다음 줄로 넘어가게됩니다. 이때 n에 -1씩 하며 반복문을 돕니다

segments2에서는 재귀형태로 함수자체를 한번씩 돌아가는 형태로 진행됩니다.

이 둘의 차이점은 segments1은 직접 변수를 하나씩 지우고 for문을 돌리는 반면 segments2는 함수 자체의 변수를 하나씩 줄인 형태로 들어가게됩니다. 즉 segments2는 다시 함수를 불러들이는 것입니다.

A-2. Figure 10.2 (Explain the process of recursion when Fibonacci(5) is called)

Figure 10.2 The recursive computation of `fibonacci(5)`. Each rectangle represents an invocation of the `fibonacci` fuunction. The call at the top of the diagram represents the initial call of `fibonacci(5)`. An arrow pointing down indicates the argument being passed into an invocation of `fibonacci`, and an arrow pointing up represents the value returned by that invocation. An invocation of `fibonacci` with no arrow pointing down away from the invocation represents a base case; observe that any invocation receiving a 0 or 1 is a base case. We see that the recursive process for `fibonacci(5)` invokes the function a total of 15 times.



B. Exercises (Write the questions down on your answer sheet) (pp. 280~287)

(write output analysis for all exercises)

1. Consider the following C++ code:

```
#include <iostream>
int sum1(int n) {
  int s = 0;
  while (n > 0) {
    s++;
    n--;
  }
  return s;
}
int input;
int sum2() {
  int s = 0;
  while (input > 0) {
    s++;
    input--;
  }
  return s;
}
int sum3() {
  int s = 0;
  for (int i = input; i > 0; i--)
    s++;
  return s;
}
int main() {
  // See each question below for details
}
```

(a) What is printed if main is written as follows?

```
int main() {
  input = 5;
  std::cout << sum1(input) << '\n';
  std::cout << sum2() << '\n';
  std::cout << sum3() << '\n';
}
```

5

5

0

먼저 sum1에 5를 대입해 s는 5번 ++를 하고 n은 5번 --하여 5의 값을 가진 s값을 내보냅니다.

그 다음엔 sum2를 실행하여 input인 5만큼 while문을 돌고 이땐 input값이 0으로 바뀌고 s는 5번 ++되어 5라는 값이 나옵니다.

마지막으로 sum3으로 출력하는데 sum2에서 input값이 0으로 바껴서 0이 출력됩니다.

(b) What is printed if main is written as follows?

```
int main() {
  input = 5;
  std::cout << sum1(input) << '\n';
  std::cout << sum3() << '\n';
  std::cout << sum2() << '\n';
}
```

5

5

5

먼저 sum1에 5를 대입해 s는 5번 ++를 하고 n은 5번 --하여 5의 값을 가진 s값을 내보냅니다.

그 다음 sum3을 실행해 input의 5만큼 반복해 s에 5를 더한 값인 5가 출력됩니다.

마지막으로 sum2를 실행해 input값인 5를 대입해 s에 5를 더한 값인 5가 나옵니다.

(c) What is printed if main is written as follows?

```
int main() {
  input = 5;
  std::cout << sum2() << '\n';
  std::cout << sum1(input) << '\n';
  std::cout << sum3() << '\n';
}
```

5

0

0

먼저 sum2를 실행해 5가 나오고 이때 input값이 변했고 뒤에 나오는 sum1과 sum3에 0을 넣기 때문에 모두 0 이 나옵니다.

(d) Which of the functions sum1, sum2, and sum3 produce a side effect? What is the side effect?

sum2는 input의 값을 바꿉니다.

(e) Which function may not use the input variable?

sum1은 input과 별개인 함수입니다.

(f) What is the scope of the variable input? What is its lifetime?

main함수 내에서 모두 적용되며 main함수가 끝나면 저 변수도 종료됩니다.

(g) What is the scope of the variable i? What is its lifetime?

sum3의 i는 for문 내에서만 작동합니다.

(h) Which of the functions sum1, sum2, and sum3 manifest good functional independence? Why?

기능 독립성은 sum1이 가장 좋습니다. 이유는 지역변수의 영향을 받지 않기 때문입니다.

2. Consider the following C++ code:

```
#include <iostream>
int next_int1() {
  static int cnt = 0;
  cnt++;
  return cnt;
}
int next_int2() {
  int cnt = 0;
  cnt++;
  return cnt;
}
int global_count = 0;
int next_int3() {
  global_count++;
  return global_count;
}
int main() {
  for (int i = 0; i < 5; i++)
    std::cout << next_int1() << " "
    << next_int2() << " "
    << next_int3() << '\n';
}
```

(a) What does the program print?

1 1 1
2 1 2
3 1 3
4 1 4
5 1 5

// int1은 static int라는 정적 변수를 써서 변수를 바꿔서 1,2,3,4,5가 나오고 int2는 동적 변수라 1,1,1,1,1이 나오고 int3은 global_count라는 변수를 밖에 썼기 때문에 1,2,3,4,5로 나옵니다.

(b) Which of the functions next_int1, next_int2, and next_int3 is the best function for the intended purpose? Why?

의도된 목적이 어떤지는 잘 모르겠으나 int1은 정적 변수를 써서 가장 의도된 거 같습니다.

(c) What is a better name for the function named next_int2?

always_same_result_next_int2

항상 변수가 0으로 나와서 결과는 항상 동일하기 때문

(d) The next_int3 function works in this context, but why is it not a good implementation of a function that always returns the next largest integer?

함수 밖의 변수를 쓰기 때문에 메모리를 많이 잡아먹습니다.

3. The following C++ program is split up over three source files. The first file, counter.h, consists of

```
int read();
int increment();
int decrement();
```

The second file, counter.cpp, contains

```
static int count;
int read(){
  return count;
}
int increment(){
  if(count < 5) {
```

```
    count++;
}
int decrement(){
  if(count > 0) {
    count--;
}
```

The third file, main.cpp, is incomplete:

```
#include <iostream>
#include "counter.h"

int main(){
  // Add code here
}
```

(a) Add statements to main that enable it to produce the following output:

```
3
2
4
```

The restriction is that the only output statement you are allowed to use (three times) is

```
std::cout << read() << '\n';
```

```
for (int i = 0; i < 3; i++) {
    increment();
  }
  std::cout<<read()<<'\n';
  decrement();
  std::cout << read() << '\n';
  for (int i = 0; i < 2; i++) {
    increment();
  }
  std::cout << read() << '\n';
```

4. Consider the following C++ code:

```
#include <iostream>
int max(int n) {
  return n;
}
int max(int m, int n) {
  return (m >= n) ? m : n;
}
int max(int m, int n, int r) {
  int x = m;
  if (n > x)
    x = n;
  if (r > x)
    x = r;
  return x;
}
int main() {
  std::cout << max(4) << '\n';
  std::cout << max(4, 5) << '\n';
  std::cout << max(5, 4) << '\n';
  std::cout << max(1, 2, 3) << '\n';
  std::cout << max(2, 1, 3) << '\n';
  std::cout << max(2, 1, 2) << '\n';
}
```

(a) Is the program legal since there are three different functions named max?

function overloading으로 각각 parameter가 달라서 괜찮습니다.

(b) What does the program print?

4
5
5
3
3
2

5. Consider the following function:

```
int proc(int n) {
  if (n < 1)
    return 1;
```

```
  else
    return proc(n / 2) + proc(n - 1);
}
```

Evaluate each of the following expressions:

(a) proc(0)

1

// n이 1보다 작아서 return 1

(b) proc(1)

2

// return proc(0.5) + proc(0) ⇒ 2

(c) proc(2)

4

// return proc(1) + proc(1) ⇒ 4

(d) proc(3)

6

// return proc(1.5) + proc(2) = proc(0.75) + proc(0.5) + proc(2) = 6

(e) proc(5)

14

// return proc(2.5) + proc(4) = proc(1.25) + proc(1.5) + proc(2) + proc(3) = 2 + 2 + 4 + 6 = 14

(f) proc(10)

60

// return proc(5) + proc(9) = 14 + proc(9) = 14 + proc(4.5) + proc(8) = 14 + ~~ = 60

(g) proc(-10)

1

// n이 1보다 작아서 return 1

6. Rewrite the gcd function so that it implements Euclid's method but uses iteration instead of recursion.

```
int gcd(int a, int b){
  return b ? gcd(b, a % b) : a;
}
```

7. If x is a variable, how would you determine its address in the computer's memory?

ampersand(&)를 붙입니다.

8. What is printed by the following code fragment?

```
int x = 5, y = 3, * p = &x, * q = &y;
std::cout << "x = " << x << ", y = " << y << '\n';
x = y;
std::cout << "x = " << x << ", y = " << y << '\n';
x = 7;
std::cout << "x = " << x << ", y = " << y << '\n';
*p = 10;
std::cout << "x = " << x << ", y = " << y << '\n';
p = q;
*p = 20;
std::cout << "x = " << x << ", y = " << y << '\n';
```


x = 5, y = 3
x = 3, y = 3
x = 7, y = 3
x = 10, y = 3
x = 10, y = 20

// x=5,y=3은 당연한거고 x=3,y=3은 x=y를 통해 x에 y의 값을 넣었기 때문이고 x=7,y=3은 x=7때문이고 x=10,y=3은 *p=&x=10이라 x의 값이 10으로 바뀌었고, x=10,y=20은 p=q에 *p = 20, *q=&y라서 y를 가리키는 q의 값이 20으로 변경되었기 때문입니다.

9. Given the declarations:

💡 int x, y, *p, *q;

indicate what each of the following code fragments will print.

(a)

```
p = &x;
x = 5;
std::cout << *p << '\n';
```

5

*p로 x에 저장된 값이 나옵니다.

(b)

```
x = 5;
p = &x;
std::cout << *p << '\n';
```

5

*p로 x에 저장된 값이 나옵니다.

(c)

```
p = &x;
*p = 8;
std::cout << *p << '\n';
```

8

*p로 x에 저장된 값이 나옵니다.

(d)

```
p = &x;
q = &y;
x = 100;
```

```
y = 200;
*q = *p;
std::cout << x << ' ' << y << '\n';
std::cout << *p << ' ' << *q << '\n';
```

100 100
100 100

*q = *p를 통해 q가 가리키는 y의 값도 100으로 변했고 *q또한 100으로 변해서 모두 100이란 값이 나옵니다.

(e)

```
p = &x;
q = &y;
x = 100;
y = 200;
q = p;
std::cout << x << ' ' << y << '\n';
std::cout << *p << ' ' << *q << '\n';
```

100 200
100 100

이번엔 q=p를 했기 때문에 y를 건드는 것이 아닌 q의 값만 바뀌게 됐습니다. 만약 *q = &y로 했다면 y도 100으로 변경되었을 것입니다.

(f)

```
x = 5;
y = 10;
p = q = &y;
std::cout << *p << ' ' << *q << '\n';
*p = 100;
*q = 1;
std::cout << x << ' ' << y << '\n';
```

10 10
5 1

p = q = &y를 통해 *p와 *q 모두 y의 값이 나옵니다.

아랫줄에선 x는 따로 연결된 것이 없어서 5가 그대로 나오고 y는 *q를 통해 1로 바뀌어 나옵니다.

(g)

```
x = 5;
y = 10;
p = q = &x;
*p = *q = y;
std::cout << x << ' ' << y << '\n';
```

10 10

p = &x와 *p = *q = y를 통해 y의 값이 x에도 전달되었습니다.

10. The following function does not behave as expected:

```
/*
* (Faulty function)
**
get_range
* Establishes a range of integers. The lower value must
* be greater than or equal to min, and the upper value
* must be less than or equal to max.
* min is the lowest acceptable lower value.
* max is the highest acceptable upper value.
* lower is assigned the lower limit of the range
* upper is assigned the upper limit of the range
*/
void get_range(int min, int max, int lower, int upper) {
  std::cout << "Please enter a data range within the bounds "
    << min << "..." << max << ": ";
  do { // Loop until acceptable values are provided
    std::cin >> lower >> upper;
    if (lower < min)
      std::cout << lower << " is too low, please try again.\n";
    if (upper > max)
      std::cout << upper << " is too high, please try again.\n";
  } while (lower < min || upper > max);
}
```

(a) Modify the function so that it works using pass by reference with pointers.

(b) Modify the function so that it works using pass by reference with reference parameters.

12. Complete the following function that assigns to its mx and my reference parameters the components of the midpoint of the points (x1;y1) and (x2;y2), represented by the parameters x1, y1, x2, and y2.

```
// Computes the midpoint of the points (x1, y1) and (x2, y2).
// The point (mx, my) represents the midoint.
void midpoint(double x1, double y1, double x2, double y2,
  double& mx, double& my) {
  // Add your code . . .

}
```

C. Additional exercises(write the questions down on your answer sheet)

C-1. Code explanation & output analysis (Write the source code and results)

```
#include <iostream>
void Hanoi(int m, char start, char middle, char end) {
  if (m == 1) {
    std::cout << "Move disc " << m << " from " << start << " to " << end
      << std::endl;
  }
  else {
    Hanoi(m - 1, start, end, middle);
    std::cout << "Move disc " << m << " from " << start << " to " << end
      << std::endl;
    Hanoi(m - 1, middle, start, end);
  }
}
int main() {
  int discs = 3;
  Hanoi(discs, 'A', 'B', 'C');
}
```