C++ 실습 5

실습 내용

- 이번 주 실습은 함수에 관련된 다양한 코드를 확인해 보는 실습입니다.

- 총 6개의 cpp파일이 있고 하나씩 실행하며 확인하면 됩니다.

- Git Pull 명령을 이용해 새로운 파일들을 내려받습니다.

- 이번주도 지난주와 마찬가지로 C++표준을 사용하는 코드들이 있습니다.

첫번째 실습

- pr1_func_basics1.cpp 파일을 실행합니다.
- 함수가 없이 순서대로 코드를 실행하는 예제입니다.
- 주석으로 각각의 작은 문제들을 해결하는 코드를 표시했습니다.
- 실제로 실행하여 낮은 가격 순으로 정렬된 상품 10개가 출력되는지 확인해보세요.

```
//상품정보 읽어오기
string products[100] = {};
int prices[100][12] = {};

//실제로는 난수(Random number)를 생성하여 집어넣음.
random_device rd;
mt19937 gen(rd());    //초기화.
uniform_int_distribution<> dist(1000, 10000); //1000~10000원 가격.
for(int i=0; i<100; i++) {
    products[i] = "product-" + to_string(i);
    for(int j=0; j<12; j++)
        prices[i][j] = dist(gen);
}

//1년 평균가격 계산
double avg_prices[100] = {};
for(int i=0; i<100; i++) {
    for(int j=0; j<12; j++)
        avg_prices[i] += prices[i][j];
    avg_prices[i] /= 12.0;
}
```

첫번째 실습

- 네모친 곳에서 첫 두 문장은 유사난수가 아닌 제대로 된 난수를 생성하기 위해 초기화 하는 부분입니다.

- uniform_int_distribution의 경우 뒤에 주어진 범위 안에서 난수가 균등하게(uniform) 분포하도록 생성합니다.

  - e.g.) (1, 10)으로 범위를 주었다면 1~10의 숫자가 대략 10%확률로 등장.

- dist(gen) 부분이 실제 난수를 생성시키는 부분입니다.

- dist, gen은 여러분이 이름을 지정할 수 있습니다.

```
//상품정보 읽어오기
string products[100] = {};
int prices[100][12] = {};

//실제로는 난수(Random number)를 생성하여 집어넣음.
random_device rd;
mt19937 gen(rd());    //초기화.
uniform_int_distribution<> dist(1000, 10000); //1000~10000원 가격.
for(int i=0; i<100; i++) {
    products[i] = "product-" + to_string(i);
    for(int j=0; j<12; j++)
        prices[i][j] = dist(gen);
}

//1년 평균가격 계산
double avg_prices[100] = {};
for(int i=0; i<100; i++) {
    for(int j=0; j<12; j++)
        avg_prices[i] += prices[i][j];
    avg_prices[i] /= 12.0;
}
```

두번째 실습

- pr1_func_basics2.cpp 파일은 첫번째 실습의 코드를 함수를 사용하도록 수정한 버전입니다.

- 프로토타입 부분과 main() 함수에서 함수를 호출하는 부분을 확인해 보세요.

- main()함수는 간결해졌고, 복잡한 코드는 모두 새로 정의된 함수 내부로 이동한 것을 확인할 수 있습니다.

```
void read_data(string* products, int prices[][12], int size);
void compute_avg_price(int prices[][12], double* avg_prices, int size);
void sort(string*, double*, int);
void print_products(string* products, double* avg_prices);

int main() {

    //상품정보 읽어오기
    string products[100] = {};
    int prices[100][12] = {};
    read_data(products, prices, 100);

    //1년 평균가격 계산
    double avg_prices[100] = {};
    compute_avg_price(prices, avg_prices, 100);

    //낮은 가격순 정렬
    sort(products, avg_prices, 100);

    //정렬된 상품을 출력
    print_products(products, avg_prices);

    return 0;
}
```

두번째 실습

- 예제를 실행하면 첫번째 실습과 동일하게 평균가격이 낮은 순으로 정렬된 상품 정보가 10개 출력됩니다.

- 난수 발생에 따라 실제 상품의 가격이 달라졌으므로 출력되는 내용은 실행시마다 달라집니다.

- 함수 내부의 코드는 복잡하지만, 무엇을 하는지 알고 있으면 실제 구현과 상관없이 사용하는 것이 가능합니다.

sort()는 다른 배열을 정렬할 때도 사용할 수 있음.

```
void sort(string* products, double* avg_prices, int size) {
    int minIdx; double tmp; string tmpStr;//for selection sort.
    for(int i=0; i<size; i++) {
        minIdx = i;
        for(int j=i+1; j<size; j++) {
            if(avg_prices[j] < avg_prices[minIdx])
                minIdx = j;
        }
        if(minIdx != i) {
            tmp = avg_prices[i];
            avg_prices[i] = avg_prices[minIdx];
            avg_prices[minIdx] = tmp;
            tmpStr = products[i];
            products[i] = products[minIdx];
            products[minIdx] = tmpStr;
        }
    }
}
```

세번째 실습

- 세번째 실습은 함수 오버로딩과 관련 된 실습입니다.
- 프로토타입 작성을 줄이기 위해 오버로딩하지 않는 세 개의 함수가 호출되기 전 파일 윗부분에 정의되어 있습니다.
- 프로토타입에 선언된 함수 중 3가지만 실제로 같은 파일에 정의가 되어 있습니다. 어떤 것인지 확인해보세요.
- sort()함수가 두 번 호출되는데, 인자에 따라 어떤 함수가 호출되는지 확인해 보세요.

```
void sort(string* products, double* avg_prices, int size);
void sort(string* products, double* avg_prices);
void sort(double* avg_prices, int size);
void sort(double* avg, int n);
void sort(double* avg);
void sort(double* avg_prices, int size, bool descending);
```

```
int main() {

    //상품정보 읽어오기
    string products[100] = {};
    int prices[100][12] = {};
    read_data(products, prices, 100);
    //1년 평균가격 계산
    double avg_prices[100] = {};
    compute_avg_price(prices, avg_prices, 100);
    //낮은 가격순 정렬 - 상품 가격만
    sort(avg_prices, 100);
    //정렬된 상품을 출력
    print_products(products, avg_prices);

    //평균 다시 계산.
    compute_avg_price(prices, avg_prices, 100);
    //낮은 가격순 정렬
    sort(products, avg_prices, 100);
    //정렬된 상품을 출력
    print_products(products, avg_prices);

    return 0;
}
```

네번째 실습

- 네번째 실습은 Default Parameter에 관한 실습입니다.

- 간단하게 x, y, z값을 출력하는 함수가 구현되어 있습니다.

- 함수 호출시 인자를 달리하면 어떻게 출력이 바뀌는지 확인해 보세요.

```
void print(int x, int y = 5, int z = 10);

void print(int x, int y, int z) {
    cout << "x:" << x << endl;
    cout << "y:" << y << endl;
    cout << "z:" << z << endl;
}

int main() {
    //y, z의 값을 명시적으로 주지 않아도 정상동작.
    cout << "Without z" << endl;
    print(1, 2);    //윗부분/아랫부분의 주석을 해제하면 에러발생.
    cout << "Without y" << endl;
    print(1);
    return 0;
}
```

네번째 실습

- 네번째 실습 파일에는 주석처리된 코드 부분이 2군데 있습니다.

- 윗부분과 아랫부분의 주석을 각각 하나씩 해제해 보면서 print(1, 2)부분에서 컴파일 에러가 발생하는지 아닌지 확인해보세요.

- 주석제거를 위해 //나 ‘/*’와 ‘*/’를 지워주면 코드가 활성화됩니다.

```
//이 부분 주석을 해제하면 print(1, 2)에서 에러발생
// void print(int x, int y);
```

```
print(1, 2);    //윗부분/아랫부분의 주석을 해제하면 에러발생.
```

```
/*
이 부분 주석을 해제하면 print(1, 2)에서 에러발생
void print(int x, int y) {
    cout << "x:" << x << endl;
    cout << "y:" << y << endl;
}
*/
```

다섯번째 실습

- 다섯번째 실습은 Call by Value, Address, Reference의 차이를 확인해 보는 것입니다.

- 오른쪽에는 순서대로 Call by Value, Address, Reference를 위해 정의된 세 개의 함수가 있습니다.

- 모든 함수는 첫번째 매개변수에 두번째 매개변수의 값을 더하도록 되어있습니다.

```
int increase(int x, int y) {
    x += y;
    return x;
}
```

```
void increase(int *a, int y) {
    a[0] += y;
}
```

```
void ref_increase(int &x, int y) {
    x += y;
    y = 10;
}
```

다섯번째 실습

- main()함수에서는 각각의 increase()함수를 다양하게 호출합니다.

- 이에 따라 변수 x나 y, 배열a 등의 값이 어떻게 바뀌는지 출력합니다.

- 코드를 보고 출력값이 어떻게 될지 먼저 유추해 보고, 실제 출력값과 비교해 보세요.

```
//Call by Value
int x = 10;
increase(x, 5); //x를 5만큼 증가.
cout << "1. x = " << x << endl;
x = increase(x, 5);
cout << "2. x = " << x << endl;
```

```
//Call by Address
int a[1] = { x }; //a를 15로 초기화.
increase(a, 5);
cout << "3-1. a[0] = " << a[0] << endl;
cout << "3-2. x = " << x << endl;

//Call by Reference
int y = 5;
ref_increase(x, y);
cout << "4-1. x = " << x << endl;
cout << "4-2. y = " << y << endl;
```

여섯번째 실습

```c
int fact(int n) {
    if(n == 1 || n == 2)
        return n;
    else
        return n * fact(n-1);
}

int iter_fact(int n) {
    int fact = 1;
    for(int i=1; i<=n; i++)
        fact *= i;
    return fact;
}
```

```c
int fibonacci(int n) {
    if(n == 0 || n == 1)
        return n;
    else
        return fibonacci(n-1) + fibonacci(n-2);
}

int iter_fib(int n) {
    int prevPrevNum, prevNum = 0, currNum = 1;
    for(int i=1; i<n; i++) {
        prevPrevNum = prevNum;
        prevNum = currNum;
        currNum = prevPrevNum + prevNum;
    }
    return currNum;
}
```

- 여섯번째 실습은 재귀함수와 관련된 실습입니다.

- 재귀와 반복형태의 factorial과 fibonacci number를 구하는 함수가 구현되어 있습니다.

여섯번째 실습

- main()함수에서는 각각의 함수들을 실행하여 결과를 보여주는 코드가 들어있습니다.

- factorial을 구하는 함수들의 경우 둘 모두 동일한 값을 출력하는지 확인해보세요.
  - 5외의 다른 값으로 계산해보아도 됩니다.

- 아랫부분의 코드는 두 형태의 fibonacci 함수를 실행하여 성능을 비교합니다.
  - 주의) fibonacci()의 경우 n값을 20에서 더 큰 값으로 늘리면 수행시간이 크게 증가할 수 있습니다.

```
cout << "Factorial of 5 " << fact(5) << endl;
cout << "Factorial of 5 " << iter_fact(5) << endl;

clock_t s_time = clock();
int repeat = 10000; //1만번 반복.
for(int n=0; n<repeat; n++) {
    fibonacci(20);
}
clock_t e_time = clock();
cout << "Recursive Fibonacci Exec. Time: " << (doub

s_time = clock();
for(int n=0; n<repeat; n++) {
    iter_fib(20);
}
e_time = clock();
cout << "Iterative Fibonacci Exec. Time: " << (doub
return 0;
```

실습 제출물

```
Factorial of 5 120
Factorial of 5 120
Recursive Fibonacci Exec. Time: 0.410224s
Iterative Fibonacci Exec. Time: 0.000426s
```

- 마지막 여섯번째 실습을 수행하고 재귀 형태와 반복 형태의 수행시간 차이를 확인해 본 뒤, 출력화면을 캡쳐하여 제출하면 됩니다.

실습 정리

- 이번주는 총 6개의 파일로 실습을 진행하였습니다.
- 함수를 사용할 때와 사용하지 않을 때의 코드를 비교해 보았습니다.
- 함수 오버로딩, 디폴트 매개변수 등의 예제를 확인했습니다.
- Call by Value, Address, Reference의 경우 코드가 어떻게 달라지는지 확인해 보았습니다.
- Recursion 예제와 함께 성능 차이에 대해서 확인하였습니다.