

Functions

프로그래밍 입문(2)

Topics

- **Function Basics**
- More about Functions
 - Default Parameter
 - Call by Value, Address, Reference
 - Recursion

What is Function?

- 프로그래밍언어에서 제어 추상화(Control Abstraction)을 위해 서브프로그램(Subprogram)을 도입.
- 함수(Function)는 C/C++에서 이런 서브프로그램을 일컫는 말.
- 다른 언어에서는 프로시저(Procedure) 또는 메소드(Method)라고 부르기도 함.

Control Abstraction

- 복잡한 프로그램의 경우, 이 프로그램이 달성하고자 하는 목표나 해결하고자 하는 문제는 여러개의 세부 목표나 문제로 나눌 수 있습니다.
- e.g.) 상품들의 정보를 읽어 최근 1년간의 평균 가격을 계산한 후, 이 가격이 낮은 순으로 정렬하여 출력하는 프로그램.
 - 상품의 정보를 읽어야 함.
 - 최근 1년간 평균 가격을 계산.
 - 가격이 낮은 순으로 정렬.
 - 정렬된 상품 리스트를 출력.

Control Abstraction

- 연속된 코드에서 모든 문제를 해결하는 것도 가능.
- e.g.) main()에 모든 코드를 집어 넣음.
- 각각의 문제를 해결하는 부분이 순서대로 나열됨.

```
int main() {  
    //상품정보 읽어오기  
    string products[100] = {};  
    int prices[100][12] = {};  
    .....  
    //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;  
    }  
  
    //낮은 가격순 정렬  
    int minIdx; double tmp; str:  
    for(int i=0; i<100; i++) {  
        minIdx = i; .....  
    }  
  
    //정렬된 상품을 출력  
    for(int i=0; i<10; i++) {  
        cout << products[i] << " - .....  
    }  
  
    return 0;  
}
```

Control Abstraction

- 또는 각각의 작은 문제들을 해결하는 서브프로그램을 만들고,
- 이 서브프로그램들을 메인프로그램에서 하나씩 실행하는 방식을 사용할 수 있습니다.
- C++에서는 함수가 서브프로그램들이 됩니다.
- 복잡한 상황에서 메인 코드가 훨씬 단순해집니다.

```
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;  
}
```

Control Abstraction

```
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* products, double* avg_prices, int size);  
void print_products(string* products, double* avg_prices);
```

- 실질적으로는 각각의 함수들이 더 작은 문제들을 해결하기 위한 방법을 구현합니다.
- 방법이 바뀌는 경우 손쉽게 다른 함수를 작성/호출하여 해결.
- 구체적인 구현방식을 몰라도, 무엇을 하는지, 어떻게 사용하는지 알면 함수를 호출하여 사용할 수 있습니다 - 라이브러리 작성.
- 또 각각의 함수들은 재사용이 가능합니다.

함수의 정의

`<return_type>` `<function_name>` `<parameters>`

```
void compute_avg_price(int prices[][12], double* avg_prices, int size) {  
    for(int i=0; i<size; i++) {  
        for(int j=0; j<12; j++)  
            avg_prices[i] += prices[i][j];  
        avg_prices[i] /= 12.0;  
    }  
}
```

`<parameter_type>` `<parameter_name>`

- 함수를 정의하기 위해서 다음과 같은 문법을 사용합니다.
- `<return_type> <function_name>(<parameters>) {
 <statements> (function body)
}`
- `<parameters> = {<parameter_type> <parameter_name>}*`

함수의 정의

```
<return_type>  <function_name>                                <parameters>
void compute_avg_price(int prices[][12], double* avg_prices, int size) {
    for(int i=0; i<size; i++) {
        for(int j=0; j<12; j++)
            avg_prices[i] += prices[i][j];
        avg_prices[i] /= 12.0;
    }
}
```

The diagram illustrates the components of a C++ function definition. Arrows point from labels to the corresponding parts of the code:
- **<return_type>** points to `void`.
- **<function_name>** points to `compute_avg_price`.
- **<parameters>** points to the entire parameter list in parentheses: `(int prices[][12], double* avg_prices, int size)`.
- **<parameter_type>** points to `double*`.
- **<parameter_name>** points to `avg_prices`.

- 반환형(return type)으로는 void를 포함한 데이터형을 사용할 수 있습니다.
- 함수 이름(function name)은 변수 이름과 같은 방식으로 지정.
- 매개변수(parameter) 선언은 변수 선언과 유사한 방식.
- 2차원 배열의 경우, `int prices[][12]` 또는 `int (*prices)[12]`로 매개변수 지정.

함수의 반환형

```
int main() {  
    return 0;  
}
```

```
string log(string message) {  
    return "log message:" + message + "\n";  
}
```

```
double average(int* a, int n) {  
    int sum = 0;  
    for(int i=0; i<n; i++)  
        sum += a[i];  
    double avg = (double) sum / n;  
    return avg;  
}
```

- void라는 특수한 경우를 제외하면, 함수는 반드시 반환형에 정의된 것과 동일한 데이터형의 값을 반환해야 함.
- 반환에는 반환문(return statement)가 쓰임.
 - return <expression>;
- 반환은 반환형과 호환되는 데이터형으로 평가되는 표현식을 사용가능.

함수의 반환형

```
string isNaturalNumber(int n) {  
    if(n > 0) {  
        return "Natural Number!";  
    }else if(n < 0) {  
        return "Not a Natural Number!";  
    }  
}
```

오른쪽과 같은 형태로
변경할 수 있음.

왼쪽 코드의 경우 컴파일러 경고가 출력.

```
: warning:  
    control may reach end of non-void function [-Wreturn-type]  
}  
^
```

```
string isNaturalNumber(int n) {  
    if(n > 0) {  
        return "Natural Number!";  
    }else if(n < 0) {  
        return "Not a Natural Number!";  
    }  
    return "Unknown";  
}
```

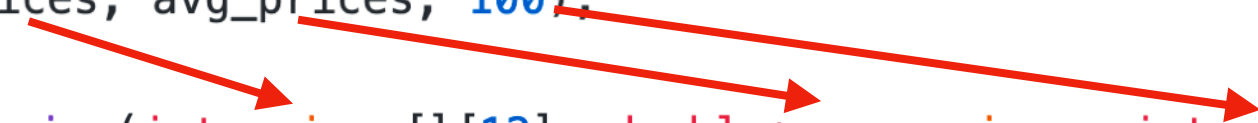
- 함수에 반환문이 쓰였지만, 조건에 따라 실행되지 않을 수 있는 경우를 확인할 필요가 있음.
- C++에서는 컴파일러 에러가 아닌 경고만 출력되므로 특별히 더 주의해야 함.

함수의 사용

//1년 평균가격 계산

```
double avg_prices[100] = {};
```

```
compute_avg_price(prices, avg_prices, 100);
```



```
void compute_avg_price(int prices[][12], double* avg_prices, int size) {  
    for(int i=0; i<size; i++) {  
        for(int j=0; j<12; j++)  
            avg_prices[i] += prices[i][j];  
        avg_prices[i] /= 12.0;  
    }  
}
```

The diagram consists of three red arrows originating from the arguments in the function call `compute_avg_price(prices, avg_prices, 100);` and pointing to the corresponding parameters in the function definition `void compute_avg_price(int prices[][12], double* avg_prices, int size) {`. The first arrow points from `prices` to `prices`, the second from `avg_prices` to `avg_prices`, and the third from `100` to `size`.

- 함수는 함수의 이름과 인자(argument)를 나열하여 호출(사용)할 수 있습니다.
 - <function_name>(<arguments>);
- 사용된 인자는 함수에 전달되어 내부의 코드에서 사용됩니다.
 - 이 때, 실제 참조는 함수의 매개변수(parameter)에 정의된 이름으로 이루어집니다.
 - 이 매개변수들은 오로지 함수 안에서만 유효합니다.


함수의 인자

//1년 평균가격 계산

```
double avg_prices[100] = {};
```

```
compute_avg_price(prices, avg_prices, 100);
```

```
void compute_avg_price(int prices[][12], double* avg_prices, int size) {  
    for(int i=0; i<size; i++) {  
        for(int j=0; j<12; j++)  
            avg_prices[i] += prices[i][j];  
        avg_prices[i] /= 12.0;  
    }  
}
```



- 함수 호출시 인자의 개수와 순서, 데이터형은 반드시 함수에 선언/정의된 매개변수와 일치해야 합니다.
- 함수 호출시 인자(argument)들은 동일한 위치 매개변수의 데이터형과 일치하는 표현식을 사용할 수 있습니다.
 - e.g.) 값, 변수명, 수식, 동일 데이터형을 반환하는 함수 호출 등.

프로토타입과 함수 선언

//1년 평균가격 계산

```
double avg_prices[100] = {};
```

```
compute_avg_price(prices, avg_prices, 100); ← 함수 정의가 되기 전 호출하면 에러
```

```
void compute_avg_price(int prices[][12], double* avg_prices, int size); 프로토타입
```

//1년 평균가격 계산

```
double avg_prices[100] = {};
```

```
compute_avg_price(prices, avg_prices, 100);
```

- 함수를 정의하기 전 함수를 호출하면, 이 호출이 올바른 것인지 알 수 없음.
 - 함수의 정의가 호출되는 코드의 앞부분에 나오는 경우는 OK.
- 따라서 이를 미리 알려주기 위해 프로토타입을 미리 선언해 줌.
- 함수의 프로토타입(prototype)은 함수의 형태(반환형, 함수이름, 매개변수)를 알려줍니다.
- 보통 헤더파일(e.g. stdio.h)에 이런 프로토타입 선언들이 들어 있음.

프로토타입

```
void sort(string* products, double* avg_prices, int size);
```



```
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++) {
```

```
void sort(string* s, double* p, int n);
```

```
void sort(string*, double*, int);
```

- 가장 간단하게 프로토타입을 선언하는 방법은 함수 정의의 첫 줄을 복사하고, 세미콜론(;)을 붙이는 것.
- 매개변수명은 달라도 상관없고, 생략도 가능함.
- 데이터형은 반드시 일치해야 함.
- 특정 위치에 어느 데이터형의 매개변수가 있는지가 중요하기 때문.

Function Overloading

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

프로토타입 선언에서는 OK

함수 정의에서는 **Error**

```
: error:  
  redefinition of 'sort'  
void sort(double* avg_prices, int n) {  
  ^
```

> void sort(double* avg_prices, int size) { ...

> void sort(double* avg_prices, int n) { ...

- 함수 오버로딩(Function Overloading)은 동일한 이름의, 매개변수가 다른 함수 여러 개를 정의하는 것.
- 함수 이름은 같지만 (하는 일이 유사하지만) 매개변수가 다른 (다른 형식의 입력을 받는) 함수를 작성가능.
- 매개변수 숫자가 다르거나 데이터형이 다른 경우만 가능.
 - 매개변수 이름만을 바꾸는 것, 반환형만 다른 것은 허용되지 않음.

Function Overloading

```
void sort(string* products, double* avg_prices, int size);
```

상품의 이름을 평균가격과 같이 정렬, 상품 숫자를 입력으로 받음.

```
void sort(string* products, double* avg_prices);
```

상품의 이름을 평균가격과 같이 정렬, 상품 숫자는 고정.

```
void sort(double* avg_prices, int size);
```

상품의 이름 없이 평균가격만 정렬.

```
void sort(double* avg);
```

상품의 평균가격을 정렬, 상품 숫자는 고정.

```
void sort(double* avg_prices, int size, bool descending);
```

상품의 평균가격을 정렬, 오름차순/내림차순을 지정 가능.

Summary

- 함수와 제어 추상화
- 함수의 정의와 사용, 프로토타입
- 함수 오버로딩