430.211 Programming Methodology (프로그래밍 방법론)

Parameters & Overloading

Lab 1 Week 4

Fall 2025

TA 김용호 peterkim98@snu.ac.kr

Machine Intelligence & Data science LAB

Outline

- Class Review

(Chapter 4 Parameters and Overloading)

  - Parameters, arguments
  - Call-by-values vs Call-by-references
  - Function overloading / Default arguments

- Assignment
- Attendance Check

Announcement

- There is a Zoom class at 10/3 (Fri) for Lab1
- There will be no attendance checks
- Don’t forget to do your homework!

Parameters, Arguments

- Parameters, arguments
  - Formal parameter : parameters listed in the function declaration
  - Argument : values passed to the formal parameters

```
#include <iostream>
using namespace std;

double totalInches(int feet, int inches){
    inches = 12 * feet + inches;
    return inches;
}

int main(void)
{
    int inches(2), feet(1), total_inches;

    total_inches = totalInches(feet, inches);

    cout << inches << endl << total_inches;

    return 0;
}
```

Formal parameter

argument

Call-by-value

- Week3 Ex 10. Call-by-value copies to a local variable
  - What if we want to modify the argument?

```cpp
#include <iostream>
using namespace std;

int totalInches(int feet, int inches){
    inches = 12 * feet + inches;
    return inches;
}

int main(void)
{
    int inches(2), feet(1), total_inches;

    total_inches = totalInches(feet, inches);

    cout << inches << endl << total_inches;

    return 0;
}
```

Values of the arguments are copied, new local variables are created

```
/* 코드가 실행되는 중입니다... */
2
14
/* 코드 실행이 완료되었습니다! */
```

Call-by-reference

- Week3 Ex 10. Call-by-reference passes the variable itself

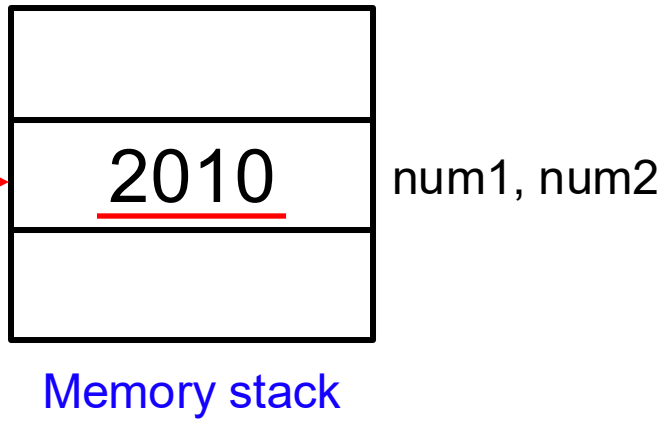
```cpp
#include <iostream>
using namespace std;

int totalInches(int& feet, int& inches){
    inches = 12 * feet + inches;
    return inches;
}

int main(void)
{
    int inches(2), feet(1), total_inches;

    total_inches = totalInches(feet, inches);

    cout << inches << endl << total_inches;

    return 0;
}
```

```
/* 코드가 실행되는 중입니다... */
14
14
/* 코드 실행이 완료되었습니다! */
```

Reference

- Reference variable
  - Same as giving another name for an already existing variable
  - It must be initialized with a variable

```
1   #include <iostream>
2   using namespace std;
3
4   int main(void){
5
6       int num1 = 2010;
7       int &num2 = num1;
8
9       return 0;
10  }
```



https://www.tutorialspoint.com/cplusplus/cpp_references.htm

& Operator

- Week3 Ex 11. & operator has 2 different uses
  - Declare reference if used during variable declaration
  - Returns address if used in front of an already declared variable

```cpp
#include <iostream>
using namespace std;

int main(void){

    int num1=2010;
    int &num2 = num1;

    num2 = 2021;
    cout << "VAL: " << num1 << endl;
    cout << "REF: " << num2 << endl;

    cout << "VAL: " << &num1 << endl;
    cout << "REF: " << &num2 << endl;
    return 0;
}
```

```
/* 코드가 실행되는 중입니다... */
VAL: 2010
REF: 2010
VAL: 0x7ffce30c39fc
REF: 0x7ffce30c39fc

/* 코드 실행이 완료되었습니다! */
```

Declare reference

Returns address

https://www.tutorialspoint.com/cplusplus/cpp_references.htm

& Operator

- Week3 Ex 11. & operator has 2 different uses
  - Declare reference if used during variable declaration

```cpp
#include <iostream>
using namespace std;

int totalInches(int& feet, int& inches){
    inches = 12 * feet + inches;
    return inches;
}

int main(void)
{
    int inches(2), feet(1), total_inches;

    total_inches = totalInches(feet, inches);

    cout << inches << endl << total_inches;

    return 0;
}
```

Declare reference

https://www.tutorialspoint.com/cplusplus/cpp_references.htm

Call-by-value vs Call-by-reference

- Week3 Ex 12. Call-by-value vs call-by-reference

```cpp
#include <iostream>
using namespace std;

void figureMeOut(int& x, int y, int& z);

int main(){

    int a, b, c;

    a=10;
    b=20;
    c=30;
    figureMeOut(a, b, c);
    cout << a << " " << b << " " << c << endl;
    cout << &a << " " << &b << " " << &c << endl;
    return 0;
}

void figureMeOut(int& x, int y, int& z)
{
    cout << x << " " << y << " " << z << endl;
    cout << &x << " " << &y << " " << &z << endl;
    x = 1;
    y = 2;
    z = 3;
    cout << x << " " << y << " " << z << endl;
    cout << &x << " " << &y << " " << &z << endl;
```

terminal

```
10 20 30
0x7ffeefbff568 0x7ffeefbff534 0x7ffeefbff560
1 2 3
0x7ffeefbff568 0x7ffeefbff534 0x7ffeefbff560
1 20 3
0x7ffeefbff568 0x7ffeefbff564 0x7ffeefbff560
```

Call-by-value creates a local variable!
(value of variable b stays unchanged
)

For more information, check out:
https://linuxhint.com/call-by-address-and-call-by-reference-cpp/

Constant Reference Parameter

- Const reference
  - What if we don't want to copy (waste time/memory) but also don't want to accidentally change the argument variable?

```
1   #include <iostream>
2   using namespace std;
3
4   void HappyFunc(int &val);
5
6   int main(void){
7
8       int num = 24;
9       HappyFunc(num);
10      cout << num << endl;
11
12      return 0;
13  }
```

< code # 1>

```
1   #include <iostream>
2   using namespace std;
3
4   void HappyFunc(const int &val);
5
6   int main(void){
7
8       int num = 24;
9       HappyFunc(num);
10      cout << num << endl;
11
12      return 0;
13  }
```

< code # 2>

Return By Reference

- Return by reference
  - Both function receives call-by-reference parameter, what will be the difference of two codes below ?

```
int& RefRetFuncOne(int &ref){
    ref++;
    return ref;
}
```

```
int RefRetFuncOne(int &ref){
    ref++;
    return ref;
}
```

Return By Reference

- Return by reference
  - Both function receives call-by-reference parameter, what will be the difference of two codes below ?

```
int& RefRetFuncOne(int &ref){
    ref++;
    return ref;
}
```

▸ Returns the reference to ref

```
int RefRetFuncOne(int &ref){
    ref++;
    return ref;
}
```

▸ Returns the value of ref

- Let's see an example!

Return By Reference

- Ex 1. Return by reference
- Let's type the code and see the result

- You can use return by reference to pass on the reference obtained from call by reference

```cpp
#include <iostream>
using namespace std;

int returnVal(int& num){
    num++;
    return num;
}
int& returnRef(int& num){
    num++;
    return num;
}

int main(void)
{
    int num_1(1), num_2(1), num_3(1), num_4(1);

    int new_num_1 = returnVal(num_1);
    //int& new_num_2 = returnVal(num_2);
    int new_num_3 = returnRef(num_3);
    int& new_num_4 = returnRef(num_4);

    new_num_1++;
    new_num_3++;
    new_num_4++;

    cout << "num_1: " << num_1 << endl;
    cout << "num_3: " << num_3 << endl;
    cout << "num_4: " << num_4 << endl;

    return 0;
}
```

Return by value

Return by reference

Compile error...Why?

Copies the reference

Return By Reference

- Ex 2. Return by reference pitfall
- Let's type the code and see the result

- Do not use return by reference on local variables

```cpp
#include <iostream>
using namespace std;

int& returnRef(int num){
    int local_num = 10;
    local_num *= num;
    return local_num;
}

int main(void)
{
    int num(2);

    int returned_num = returnRef(num);

    cout << "num: " << returned_num << endl;

    return 0;
}
```

```
/* 코드가 실행되는 중입니다... */
main.cpp: In function 'int& returnRef(int)':
main.cpp:5:9: warning: reference to local variable 'local_num' returned [-Wreturn-local-addr]
     int local_num = 10;
         ^

/elice/runner.sh: line 2:    26 Segmentation fault      (core dumped) ./main

/* 코드 실행이 완료되었습니다! */
```

Parameters and Overloading

- Function overloading

Giving two(or more) function definitions for the same function name

- Each function definition has its own declaration
- Must have different numbers of formal parameters or some formal parameters of different types (different signature)

```
double average(double n1, double n2);

int average(int n1, int n2);

double average(double n1, double n2, double n3);
```

Parameters and Overloading

- Overloading resolution

Compilers choose a function among overloaded ones by two rules

1. Exact match

If the number and types of arguments exactly match a definition

2. Matching using automatic type conversion

If there is no exact match but there is a matching using automatic type conversion

Parameters and Overloading

- Ex 3. Overloading resolution (automatic type conversion)
  - Let's guess what will happen and then see the results

```cpp
#include <iostream>
using namespace std;

int average(int n1, int n2);

int main(){

    int a(2), b(3);
    double c(2.0), d(3.0), e, f;


    e = average(a, b);
    f = average(c, d);

    cout << e << endl;
    cout << f << endl;

    return 0;
}

int average(int n1, int n2)
{
    return ((n1 + n2)/2);
}
```

Parameters and Overloading

- Ex 3. Overloading resolution (automatic type conversion)
  - Let's guess what will happen and then see the results

```
#include <iostream>
using namespace std;

int average(int n1, int n2);

int main(){

    int a(2), b(3);
    double c(2.0), d(3.0), e, f;


    e = average(a, b);
    f = average(c, d);

    cout << e << endl;
    cout << f << endl;

    return 0;
}

int average(int n1, int n2)
{
    return ((n1 + n2)/2);
}
```

- Automatic type conversion
(No compile error)

but wrong result !!

- Let's add overloading function for double type variable

Parameters and Overloading

- Default arguments

  - All the default argument positions must be in the rightmost position

  - Default argument should be given in the function declaration but not in the function definition. Some compilers consider this an error.

```
1   #include <iostream>
2   using namespace std;
3
4   void foo(int a, int b=2);
5
6   int main(){
7
8       foo(1);
9
10      return 0;
11  }
12
13  void foo(int a, int b){
14      cout << a+b << endl;
15  }
16
```

Parameters and Overloading

- Ex 4. Overloading and default arguments
  - Guess the outputs and see the results

```cpp
#include <iostream>
using namespace std;

void foo(int a);

void foo(int a, int b=2);

void foo(double a, double b, double c);

int main(){

    foo(1);

    return 0;
}
```

```cpp
void foo(int a){
    cout << a << endl;
}

void foo(int a, int b=2){
    cout << a+b << endl;
}

void foo(double a, double b, double c){
    cout << a+b+c << endl;
}
```

Parameters and Overloading

- Ex 4. Overloading and default arguments

```cpp
#include <iostream>
using namespace std;

void foo(int a);

void foo(int a, int b=2);

void foo(double a, double b, double c);

int main(){

    foo(1);

    return 0;
}

void foo(int a){
    cout << a << endl;
}

void foo(int a, int b=2){
    cout << a+b << endl;
}

void foo(double a, double b, double c){
    cout << a+b+c << endl;
}
```

Guess the output and see the results

Error

1) Redefinition of default argument
2) Ambiguous call to overloading function

Parameters and Overloading

- Overloading resolution (More complex)

Candidate function

```cpp
#include <iostream>
using namespace std;

void foo(int a);

void foo(int a, int b=2);

void foo(double a, double b, double c);

int main(){

    foo(1);

    return 0;
}

void foo(int a){
    cout << a << endl;
}

void foo(int a, int b=2){
    cout << a+b << endl;
}

void foo(double a, double b, double c){
    cout << a+b+c << endl;
}
```

Viable function

Given set of candidate functions,
the next step of overload resolution is examining arguments and parameters to reduce the set to the set of viable functions

To learn more…
(https://en.cppreference.com/w/cpp/language/overload_resolution)

Assignment

- Relative prime number(서로소) calculator
  - Given two integers, calculate their GCD and reduce them to their relative primes
  - The function convlowterms() should modify the two integers and return the GCD
  - If either integers are not positive, return -1
  - Ex) given 60 and 48, convlowterms() changes them to 5 and 4 and returns 12
  - You can implement other functions if needed
  - DO NOT MODIFY THE MAIN FUNCTION
  - Du

```
/* 코드가 실행되는 중입니다... */
val1: ? 60
val2: ? 48
val1: 5 val2: 4 GCD: 12

/* 코드 실행이 완료되었습니다! */
```

```
/* 코드가 실행되는 중입니다... */
val1: ? -5
val2: ? 10
Input positive integers!

/* 코드 실행이 완료되었습니다! */
```

Assignment

- Relative prime number(서로소) calculator
- Main function

```
// ============= DO NOT CHANGE =================
int main()
{
    int val1(0), val2(0), gcd(0);

    cout << "val1: ? ";
    cin >> val1;
    cout << "val2: ? ";
    cin >> val2;

    gcd = convlowterms(val1, val2);

    if (gcd != -1){
        cout << "val1: " << val1 << " val2: " << val2 << " GCD: " << gcd << endl;
    }
    else{
        cout << "Input positive integers!" << endl;
    }

  return 0;
}
// ============= DO NOT CHANGE =================
```

Attendance Check

- Triangle area calculator

The area of an arbitrary triangle can be computed using the formula

$$Area = \sqrt{s(s-a)(s-b)(s-c)}$$

where a, b, and c are the lengths of the sides, and s is the semiperimeter

$$s = (a+b+c)/2$$

Write a void function getArea() that modifies the area

Note that not all combinations of a, b, and c produce a valid triangle. Your code should output the area only for legal data and round into the 2nd decimal place using cout precision. (Lab1 Week 1)

Attendance Check

- Outputs

```
Enter 3 consecutive lengths of edges : 2 4 6
Invalid triangle!
```

```
Enter 3 consecutive lengths of edges : 3 4 5
Area : 6.00
```

Thank you!