

Prerequisites

Modern cpp Programming lecture 0

Practice basic cpp features!!

- Contents
 - basic features of cpp
 - standard input / output
 - loop statement
 - condition statement
 - basic function call
 - array
 - recursion

iostream

- **#include**
 - includes library or other files (module)
 - allow you to use functions / data in other module
 - **#include <iostream>**
 - you can use basic functionalities provided by **<iostream>** library!
 - iostream library includes functions for standard input / output.
 - **cout** : prints data in a console (standard output)
 - **endl** : ends line with newline character
 - » **endl** also empties output stream (don't need to know now)
 - **cin** : reads data from a console (standard input)

Standard input / output

```
int number;
```

cf) \n : newline character (Enter key)

```
cin >> number;
```

standard input

```
cout << number << endl;
```

standard output

without endl, we cannot use
next line to print something
cf) you can just print "\n" instead

```
cout << "Hello, ";  
cout << "World!!" << endl;
```

```
cout << "Hello, " << endl;  
cout << "World!!" << endl;
```

>> Hello, World!!

>> Hello,
World!!

Newline character

- \n : newline character

```
cout << "hello, \n world!!" << endl;
```

```
>> hello,  
world!!
```

- In Korean windows, represented as \n...
- \ (or ¶) locates below backspace
- \ (backslash) is used to represent various escape sequences:
 - \n : newline
 - \t : tab
 - \b : backspace
 - \r : move cursor to the starting point of line

namespace

- a set of symbols which organizes objects of various kind
- simple example!!
 - dog (of Alice)
 - dog (of Bob)
 - If we just write “dog”, we cannot differentiate two dogs
 - but if we write “**Alice :: dog**”, “**Bob :: dog**”, now two dogs can be differentiated easily

namespace

- `using namespace std`
 - “I’ll use objects and functions in std (standard) namespace”
 - ex) `cout, cin...`
 - declare every c++ standard library function

```
using namespace std;

int main() {
    cout << "Hello, world!! << endl;
    return 0;
}
```

or

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

int main() {
    std :: cout << "Hello, world!! << std ::endl;
    return 0;
}
```

Programming!!

- Problem 1
- sample code available at
 - [https://github.com/jeonhyun97/cpp_4_undergraduates/blob/master
/cpp/lecture0/code/1_hello_world.cpp](https://github.com/jeonhyun97/cpp_4_undergraduates/blob/master/cpp/lecture0/code/1_hello_world.cpp)
- write program which reads your name with standard input
- and say hello to you!!
- output should be like this:

>> Your name: hyeon

hello, hyeon!!



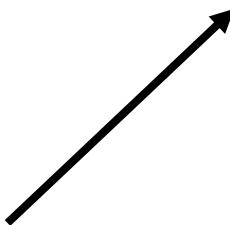
input

Data type

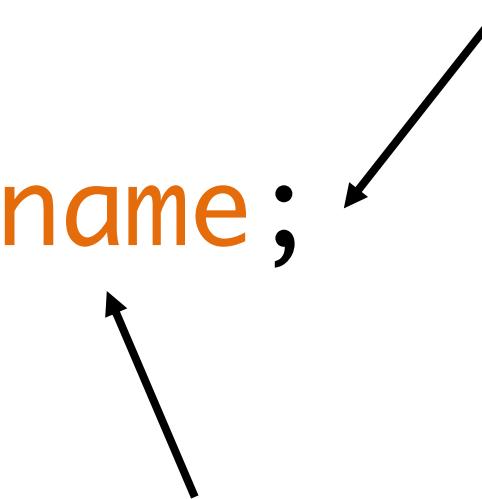
- In the prev example, you might saw such syntax...
- `string name;`
- Let's look careful...

never forget semicolon!!

`string name;`



Data type of the variable
denotes kinds of the variable



variable name (identifier)
represents variable itself &
differentiate with other variables

Primitive data types

- Basic data types of cpp
 - int : represents integer
 - float : represents real number (**floating point**)
 - bool : Boolean, denotes true or false
 - char: denotes single alphabet
 - ex) a, b, 1, 5, &...
 - void: typeless
 - you'll see...

Primitive data types

- Data type is really important!!
- especially for static language
 - determine the type of variables at compile time
 - ex) c, cpp, Java
 - essential to write “data type” for every variable
- cf) dynamic language
 - no need to write data type
 - ex) javascript
 - `let num = 1;`
 - `let name = "cpp";` => every variable can be declared with "let"

cpp string

- String is not a primitive type
- Supports "sequence of characters"
- but you should be familiar to it
- can be used by including <string> header
- declared as `std::string`
 - using namespace std will help you use string syntax in alone.

Data type – warning!!

- Data type can make significant error!!

```
#include <iostream>

using namespace std;

int main() {
    char aChar = 97;
    int aInt = 97;

    cout << aChar << endl;
    cout << aInt << endl;
}
```

as "97" is the number which encodes alphabet "a" in cpp



>> a
>> 97

For more information, refer <https://en.wikipedia.org/wiki/ASCII>

Then, what does “main” mean??

```
int main() {  
    cout << "Hello, word! ! << endl;  
    return 0;  
}
```

we will come back...after we study “function”
for now, just think as an essential part to execute certain program

Cpp Operator

- Assignment operator
 - “=”
 - assigns value to a variable
 - you can assign value at declaration, or after declaration
 - `int num = 1;`
 - `int num;`
`num = 1;`

Cpp Operator

- Arithmetic operators
 - Addition (+)
 - Subtraction(-)
 - Multiplication(*)
 - Division (/)
 - if real number : acts as common division
 - if integer : returns quotient
 - `float num = 3; cout << (num / 2) << endl;` >> 1.500000
 - `int num = 3; cout << (num / 2) << endl;` >> 1
 - Remainder(%)
 - `int num = 19; cout << (num % 8) << endl;` >> 3

Cpp Operators

- Special assignment operators

- $a += b;$
- $\Leftrightarrow a = a + b;$
- $a -= b;$
- $\Leftrightarrow a = a - b;$
- $a *= b;$
- $\Leftrightarrow a = a * b;$
- $a /= b;$
- $\Leftrightarrow a = a / b;$

Cpp Operators

- Increment / Decrement operators
 - `++,--` : increments / decrements 1
 - `a = 2; a++;` => a becomes 3
 - however, `a++` and `++a` is different!!

```
// postfix increment operator
int a = 2;
cout << a++ << endl;    >> 2
cout << a << endl;      >> 3
```

```
// prefix increment operator
int a = 2;
cout << ++a << endl;    >> 3
cout << a << endl;      >> 3
```

- postfix : uses (returns) original value at the line
- prefix : use (returns) modified value at the line

Cpp Operators

- Relational operators
 - <, >, <=, >= : common operators
 - != : true when two operands are different, otherwise false
 - == : true when two operands are same, otherwise false
 - Examples will help you

```
cout << (1 < 3) << endl;      >> 1 (which means True)
cout << (1 > 3) << endl;      >> 0 (which means False)
cout << (1 <= 1) << endl;     >> 1
cout << (3 >= 1) << endl;     >> 1
cout << (1 != 1) << endl;      >> 0
cout << (1 != 2) << endl;      >> 1
cout << (1 == 1) << endl;      >> 1
cout << (1 == 2) << endl;      >> 0
```

Cpp Operators

- Logical operators
 - and : `&&`
 - or : `||`
 - not : `!`
 - evident (middle school mathematics...)
 - again, examples will help you!!

```
cout << (true && true) << endl;           >> 1
cout << (false && true) << endl;          >> 0
cout << (false || true) << endl;           >> 1
cout << (false || false) << endl;          >> 0
cout << (!false) << endl;                  >> 1
cout << (!false || false) << endl;         >> 1
cout << ((!false || false) && true) << endl; >> 1
cout << ((1 < 3) && (4 == 4)) << endl;    >> 1
```

Cpp Operators

- Order of operators
 - we should perform multiplication before addition... if there is no parenthesis.
 - Rule b/w operators is very important!!
 - Able to make significant error!!
 - However, you don't need to memorize the order
 - you can easily search it, or just simple use parenthesis to make certain ordering

Cpp operators

1	() [] -> . ::	Function call, scope, array/member access
2	! ~ - + * & <i>sizeof type cast</i> ++ --	(most) unary operators, <i>sizeof</i> and <i>type casts</i> (right to left)
3	* / % MOD	Multiplication, division, modulo
4	+ -	Addition and subtraction
5	<< >>	Bitwise shift left and right
6	< <= > >=	Comparisons: less-than and greater-than
7	== !=	Comparisons: equal and not equal
8	&	Bitwise AND
9	^	Bitwise exclusive OR (XOR)
10		Bitwise inclusive (normal) OR
11	&&	Logical AND
12		Logical OR
13	? :	Conditional expression (ternary)
14	= += -= *= /= %= &= = ^= <<= >>=	Assignment operators (right to left)
15	,	Comma operator

Loop statement

- you might want to print number 1~100
- how about...

```
cout << 1 << endl;  
cout << 2 << endl;  
cout << 3 << endl;  
cout << 4 << endl;  
...  
cout << 100 << endl;
```

- inefficient!!
- You should order your program to...
- loop through numbers and print them

Loop statement

- for statement

A (initialization) B (condition) D (change)

```
for (int i = 0; i < 10; i++) {  
    cout << i << endl;  
}
```

C (body)

- execution order: A -> B -> C -> D -> B -> C -> D -> B -> C -> D...
- This statement is saying...
 - first, initialize variable i as 0 (A)
 - print i (C) while incrementing it, (D)
 - until i is no longer smaller than 10 (B)

Loop statement

- while statement

A (initialization)

```
int i = 0;      B (condition)
while (i < 10) {
    cout << i << endl; C (body)
    i++          D(body)
}
```

- execution order : same with for-loop

- A -> B -> C -> D -> B -> C -> D...
- initialization (A) should be performed before the loop
- increment should be wrote explicitly in the body

Loop statement

- do-while statement

```
let i = 0;  
do {  
    cout << i << endl; C  
    i++ D  
} while (i < 10) B
```

- execution order
 - A -> C -> D -> B -> C -> D -> B...
 - condition statement (B) performs after body statements (C, D)

Loop statement

- continue
 - stop current body execution

```
for (int i = 0; i < 5; i++) {  
    cout << i << endl;  
    continue; // ends current body execution  
    cout << i << endl; // never executed  
}
```

output:

0
1
2
3
4

Loop statement

- break
 - stops entire loop (escape from the loop)

```
for (int i = 0; i < 5; i++) {  
    cout << i << endl;  
    break; // ends loop  
}
```

output:

0

After single execution of cout, the loop ends due to break statement

Programming!!

- Problem 2 ~ 5
- sample code available at
 - [Problem 2](#)
 - [Problem 3](#)
 - [Problem 4](#)
 - [Problem 5](#) (requires if-statement...see next section)
- Enjoy loop!!

Condition statement

- If statement

```
int i = 7;
if (i % 3 == 1) {
    cout << "Remainder : 1" << endl;
}
else if (i % 3 == 2) {
    cout << "Remainder : 2" << endl;
}
else {
    cout << "Remainder : 0" << endl;
}
```

- Quite Intuitive!!

- If **the condition of if statement** returns true, executes **its body**
- If not, goes to next statement (**else if**) and checks the condition
- If **the condition in else if statement** returns true, executes **its body**
- **else**, executes **the body in else statement**

Condition statement

- If statement
 - else / else if statement is not essential
 - you can only use if statement alone

```
int i = 1;
while(true) {
    cout << i << endl;
    i++;
    if(i > 3)
        break;
}
>> 1
2
3
```

loops forever until it reaches
break statement

If i is bigger than 3, ends loop

Condition statement

- switch statement

```
switch(score) {  
    case 10 :  
        cout << "Your grade is A";      // executes when score == 10  
        break;  
    case 9  :  
        cout << "Your grade is B";      // executes when score == 9  
        break;  
    case 8  :  
        cout << "Your grade is C";      // executes when score == 8  
        break;  
    case 7  :  
        cout << "Your grade is D";      // executes when score == 7  
        break;  
    default :  
        cout << "Your grade is F";      // execute when score != 10, 9, 8, 7  
}
```

- If the value of variable score is 10, prints that the grade is 10

Condition statement

- Switch Statement

```
switch(score) {  
    case 10 :  
        cout << "Your grade is A";  
        break;  
    case 9 :  
        cout << "Your grade is B";  
        break;  
    case 8 :  
        cout << "Your grade is C";  
        break;  
    case 7 :  
        cout << "Your grade is D";  
        break;  
    default :  
        cout << "Your grade is F";  
}
```

```
if (score == 10) {  
    cout << "Your grade is A";  
}  
else if (score == 9) {  
    cout << "Your grade is B";  
}  
else if (score == 8) {  
    cout << "Your grade is C";  
}  
else if (score == 7) {  
    cout << "Your grade is D";  
}  
else {  
    cout << "Your grade is C";  
}
```

- two code block have same functionality!!
- sometimes using switch statement is more intuitive

Condition statement

- Switch Statement

```
switch(score) {  
    case 10 :  
        cout << "Your grade is A";  
        break;  
    case 9 :  
        cout << "Your grade is B";  
        break;  
    default :  
        cout << "Your grade is F";  
}
```

- Okay...why **break**??

- Without **break**

- switch statement just executes every case below selected case
- i.e. slides down switch statement

Condition statement

- Switch Statement

```
score = 9
switch(score) {
    case 10 :
        cout << "Your grade is A" << endl;
    case 9 :
        cout << "Your grade is B" << endl;
    case 8 :
        cout << "Your grade is C" << endl;
    default :
        cout << "Your grade is F" << endl;
}
```

```
output >>
Your grade is B
Your grade is C
Your grade is F
```

```
score = 9
switch(score) {
    case 10 :
        cout << "Your grade is A" << endl;
        break;
    case 9 :
        cout << "Your grade is B" << endl;
        break;
    case 8 :
        cout << "Your grade is C" << endl;
        break;
    default :
        cout << "Your grade is F" << endl;
}
```

```
output >>
Your grade is B
```

- Never forget break!!

Programming!!

- Problem 6~10
- Sample Code Available at
 - [Problem 6](#)
 - [Problem 7](#)
 - [Problem 8](#)
 - [Problem 9](#)
 - [Problem 10](#)
- The famous “star-printing” example!!
 - Be creative!! every problem can be solved with only two for-statement
 - loop / condition statement is really powerful!!

Programming!!

- Problem 11 / Problem 12
- be familiar to loop / condition statement
- cf) switch statement:
 - less powerful than if statement, but sometimes enhances the readability of your code a lot
- Sample code available at
 - [Problem 11](#)
 - [Problem 12](#)

Functions

- Basic execution module of cpp
- not only for cpp, but also for every programming languages!!
- *A block of code which only runs when it called*
- *Able to pass data as parameters*
- *Able to return data (result) after the execution*

Functions

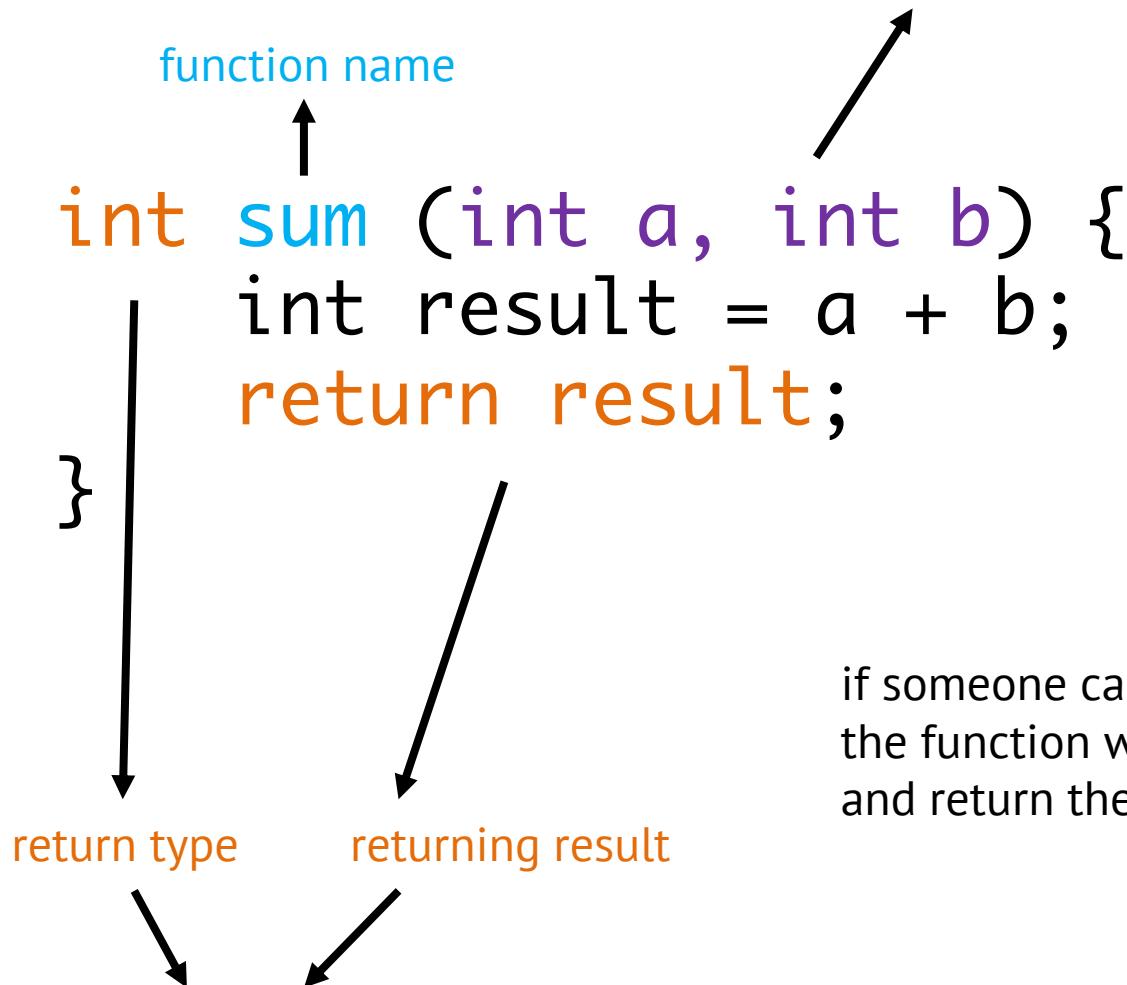
- Contents

```
int sum(int a, int b) {  
    int result = a + b;  
    return result;  
}  
  
int main() {  
    cout << sum(3, 5) << endl;      >> 8  
    return 0;  
}
```

- Easy to use, just declare it, and use it!!
- Why useful??
 - easy to reuse code block
 - whenever you need to add two ints, you only need to call **function sum**

Functions

parameters, passed by the function which is calling this function
Data type / parameter name should be specified



if someone calls `sum(5, 8)`,
the function will calculate $5 + 8$
and return the result (13)

result's data type should match to return type
in this case, variable `result` should be an int

Functions

- Just think as mathematical function!!
- $\sin(\cos \pi)$ can be interpreted as:
 - first, apply parameter π to function \cos .
 - again, apply the value returned from \cos to function \sin .
 - then it will return final value.
- Same procedure can be adopted to the functions in cpp!!

```
float sin (float a) {  
    ...return result;  
}
```

```
float cos (float b) {  
    ...return result;  
}
```

```
int main() {  
    cout << sin(cos(3.14)) << endl;  
    return 0;  
}
```

Functions

- Functions might have zero parameter, or return nothing
 - if no parameter

```
int returnsThree() {  
    return 3;  
}
```

- if returns nothing

```
void printScore(int score) {  
    cout << "Your score is: ";  
    cout << score << endl;  
    return;  
}
```

- use `void` to denote that the function returns nothing
- just use `return` statement in alone (possible to omit)

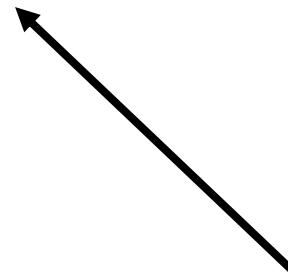
Programming!!

- Problem 13
- Sample code available at
 - https://github.com/jeonhyun97/cpp_4_undergraduates/blob/master/cpp/lecture0/code/13_arithmetic.cpp
- Enjoy function!!
- How about reimplementing problem 1~12 using function??

Recursion

- Powerful tool in computer science!!
- Achieves by implementing functions that calls themselves
- Let's see simple example...

```
int factorial(int n) {  
    if (n == 1) return 1;  
    else return factorial(n - 1) * n;  
}  
  
int main() {  
    cout << "10! = " << endl;  
    cout << factorial(10);  
    return 0;  
}
```



again calling itself

Recursion

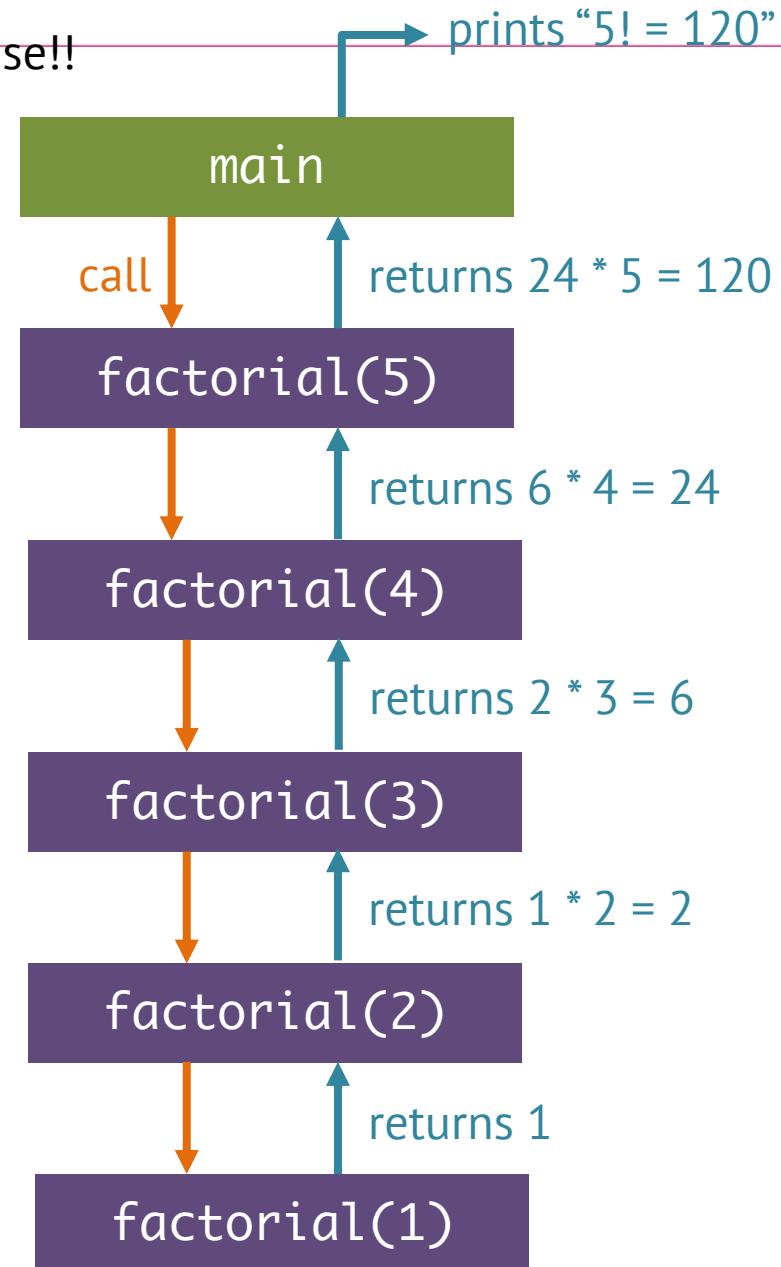
try not to forget base case!!

- Let's analyze!!

```
int factorial(int n) {  
    if (n == 1) return 1;  
    else return factorial(n - 1) * n;  
}
```

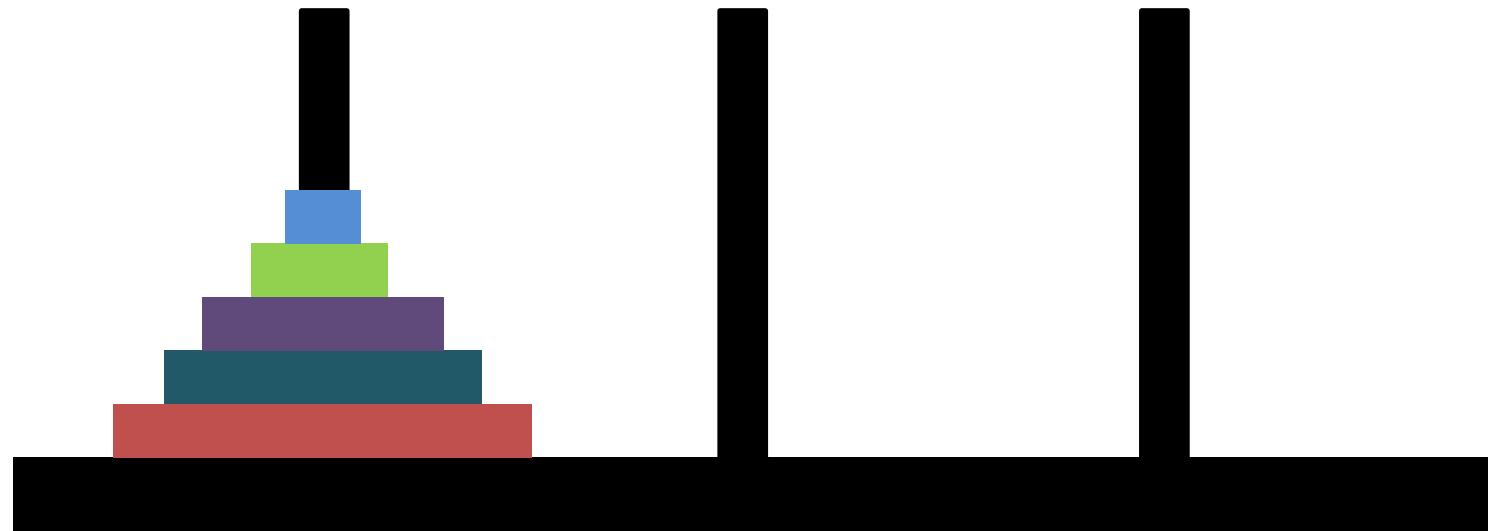
```
int main() {  
    cout << "5! = " << endl;  
    cout << factorial(5);  
    return 0;  
}
```

- Generates Beautiful code!!



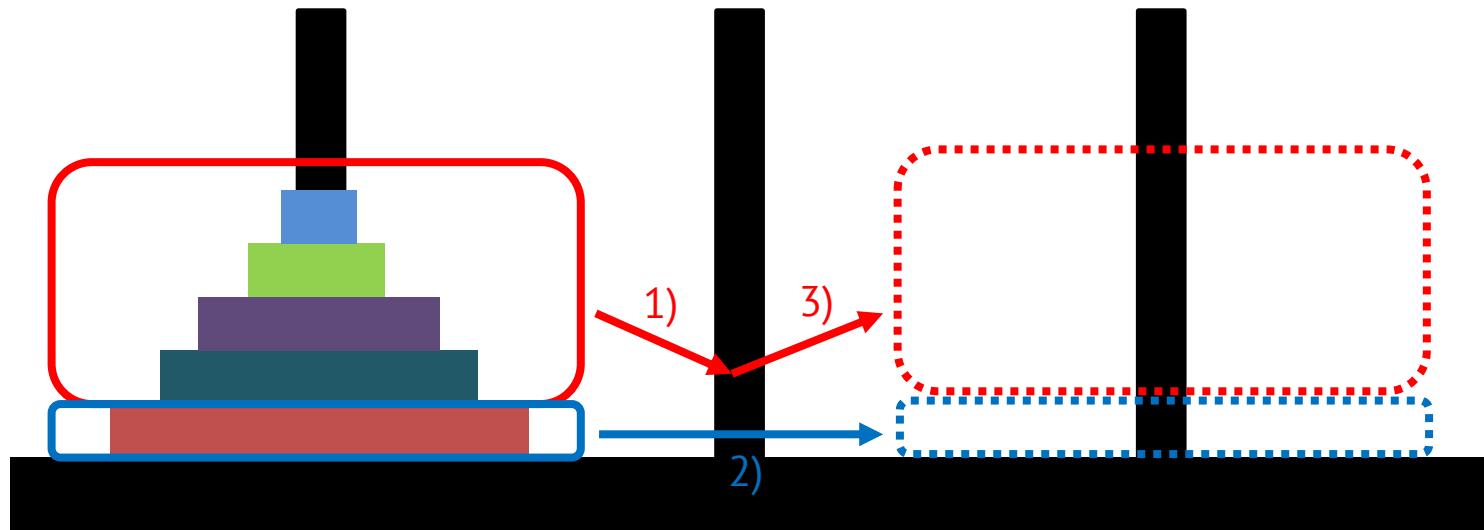
Recursion

- Famous example – Hanoi Tower problem
 - There exists 3 rods and 64 discs with varying size.
 - Monks should move every disc from one rod to other rod
 - However, larger disc cannot placed over smaller disc
 - How many times should the monks move the discs??



Recursion

- Hanoi Tower problem
- let's disassemble the procedure: moving n discs!!
 - Initially, every disc is in first rod
 - 1) move (n-1) discs to second rod
 - 2) move the largest disc to third rod
 - 3) move (n-1) discs to third rod



Recursion

- Hanoi tower problem
 - let's say (number of movements for n disc problem) : hanoi(n)
 - $\text{hanoi}(n) = 2 * \text{hanoi}(n - 1) + 1$
 - Now we can use recursion to solve problem!!

```
int hanoi(int n) {  
    if (n == 1) return 1;  
    else return (hanoi(n - 1) * 2 + 1);  
}
```



base case : hanoi(1)

```
int main() {  
    cout << "Monks need to make ";  
    cout << hanoi(10);  
    cout << " movements to move 10 discs" << endl;  
}
```

>> Monks need to make 1023 movements to move 10 discs

Programming!!

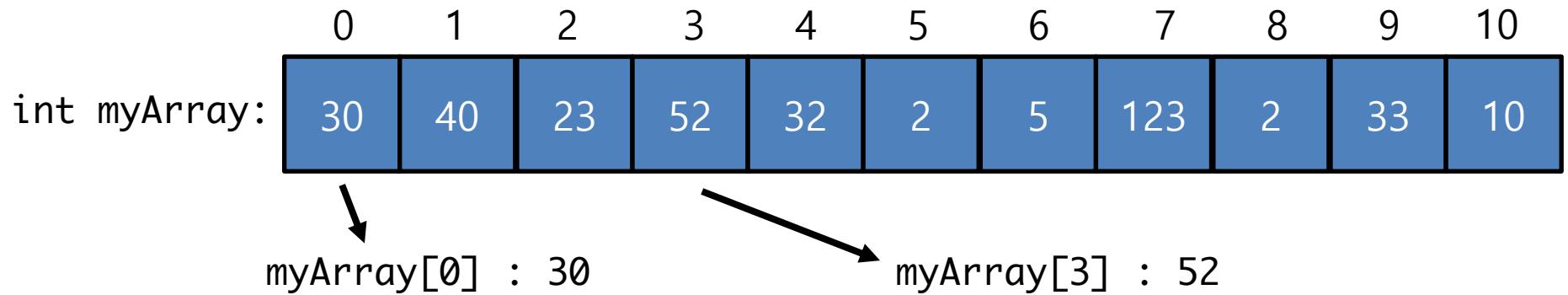
- Problem 14
- Sample code available at
 - https://github.com/jeonhyun97/cpp_4_undergraduates/blob/master/cpp/lecture0/code/14_fibonacci_easy.cpp
- Implement the program that calculates n-th Fibonacci number
- use recursion!!
- $\text{Fibo}(n) = \text{Fibo}(n-1) + \text{Fibo}(n-2)$
- hint : base case
 - $\text{Fibo}(1) = 1, \text{Fibo}(2) = 1$

Array

- Basic container to store data
- easy data assignment / access using index
- Think of storing the score of 30 students
- declaring variable student1, student2...student30...???
- Inefficient!!
- how about just declaring array:
 - `int student[30]`
 - and access the score of i-th student using `student[i]`??

Array

- Any data type can construct array!!
 - int myArray[30];
 - float myArray[30];
 - string myArray[30];
- Index starts with 0 (zero)
 - less intuitive, but one of the most important feature in CS field



Array

- Various ways to initiate value

```
int x[] = {1,2,3}; // x has type int[3] and holds 1,2,3  
int y[5] = {1,2,3}; // y has type int[5] and holds 1,2,3,0,0  
int z[5] = {1,2,3,4,5}; // z has type int[5] and holds 1,2,3,4,5  
int w[3] = {0}; // z has type int[3] and holds 0,0,0
```

- Or just use burdensome method

```
int x[3];  
x[0] = 1;  
x[1] = 2;  
x[2] = 3; // x has type int[3] and holds 1,2,3
```

Array

- Simple example will help you...

```
cout << myArray[0] << endl;      >> c
cout << myArray[1] << endl;      >> cpp
cout << myArray[2] << endl;      >> java
myArray[1] = "c++";
cout << myArray[1] << endl;      >> C++
```

- cf) iterate array:

```
for(int i = 0; i < 3; i++) {
    cout << myArray[i] << endl;
}
```

Output:

c
C++
java

Multi-dimensional array

- sometimes, you want to represent data in 2D
- ex) matrix
 - $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$: better to represent it with 2D array!!
 - `matrix[2][2] = {{1, 2}, {3, 4}}`
 - $\Rightarrow \text{matrix}[0][1] : 2$
 - $\Rightarrow \text{matrix}[1][0] : 3$
- iterating 2D array : use double for-statement!!

```
for(int i = 0; i < 2; i++) {
    for(int j = 0; j < 2; j++) {
        cout << matrix[i][j] << " " << endl;
    }
    cout << endl;
}
```

>> 1 2
 3 4

Programming!!

- Problem 15
- Efficient way to achieve n-th Fibonacci number
- Sample code available at
 - https://github.com/jeonhyun97/cpp_4_undergraduates/blob/master/cpp/lecture0/code/15_fibonacci_normal.cpp
- use array holding Fibonacci series!!
- $\text{Fibo}[0] = 1, \text{Fibo}[1] = 1, \text{Fibo}[2] = 2 \dots \text{Fibo}[n-1] = (\text{n-th Fibo num})$

Programming

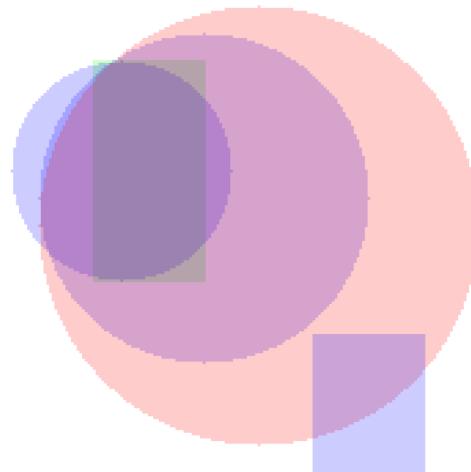
- Problem 16
 - [Sample code](#)
- Much more efficient way for Fibonacci num!!
- hint: use the formula :
- $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_n \end{pmatrix}$ and therefore
- $\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_n \end{pmatrix} = \begin{pmatrix} F_{k+1} & F_k \\ F_k & F_{k-1} \end{pmatrix}^2$ if n is even number ($k = \frac{n}{2}$)
- $\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_n \end{pmatrix} = \begin{pmatrix} F_{k+1} & F_k \\ F_k & F_{k-1} \end{pmatrix} \begin{pmatrix} F_k & F_{k-1} \\ F_{k-1} & F_{k-2} \end{pmatrix}$ if n is odd number ($k = \frac{n+1}{2}$)
- Implement the function which multiplies matrix!!
- Maybe you might be surprised by `int**` syntax
- From now, just use it!!

Programming!!

- Problem 17
- Final problem to learn the prerequisites for this course!!
- Implement bubble sort!!
- Material: <https://www.programiz.com/dsa/bubble-sort>
- Korean: <https://gmlwjd9405.github.io/2018/05/06/algorithm-bubble-sort.html>
- Try not to reference code!!
- Sample code available at
 - https://github.com/jeonhyun97/cpp_4_undergraduates/blob/master/cpp/lecture0/code/17_bubbleSort.cpp

ASSN1

Shape Renderer



Shape renderer

- Functions
 - get circle data with standard input
 - get rect data with standard input
 - process input data => generate image
 - processed data => print `picture.txt` file (already implemented)
 - Render the image by uploading `picture.txt` file (already implemented)
 - [Web Renderer](#)

Shape Renderer

- Main menu

```
=====
Add shapes!!
1. Rect
2. Circle
3. Print
INPUT: █
```

- Select menu with standard input
- 1 => goes to Rect stage
- 2 => goes to Circle stage
- 3 => prints processed image data (implemented)
- other => select menu again

```
=====
Add shapes!!
1. Rect
2. Circle
3. Print
INPUT: 4
Invalid input...
=====
Add shapes!!
1. Rect
2. Circle
3. Print
INPUT: █
```

Shape Renderer

- Rect stage
 - gets rect attributes with standard input
 - x coordinate of left-top vertex
 - y coordinate of right-top vertex
 - width
 - height
 - color

```
Add shapes!!
1. Rect
2. Circle
3. Print
INPUT: 1
-----
x: 40
y: 30
width: 100
height: 40
Pick color (red, green, blue)
color: red
Rect added!!
```

- Calculates pixel data due to attributes
- After calculation, goes back to main menu

Shape Renderer

- Circle stage
 - x coordinate of center
 - y coordinate of center
 - radius
 - color
- Calculates pixel data due to attributes
- After the calculation, goes back to main Menu

```
=====
Add shapes!!
1. Rect
2. Circle
3. Print
INPUT: 2
-----
cx: 40
cy: 80
radius: 40
Pick color (red, green, blue)
color: blue
Circle added!!
```

Shape Renderer

- *TODO*
 - int mixColor()
 - formula:
 - $\text{newColor} = \text{inputColor} * \text{inputAlpha} + \text{originalColor} * (1 - \text{inputAlpha})$
 - Alpha value provided
 - process r,g,b pixel data independently!!
 - string pickColor()
 - possible input : red, green, blue
 - otherwise, get input again
 - void addRect(), addCircle()
 - process data and update r, g, b pixel value

```
Pick color (red, green, blue)
color: grey
Invalid color...
color: yellow
Invalid color...
color: red
Circle added!!
```

Shape Renderer

- Info
 - Picture size
 - 250 * 300
 - r, g, b pixel data initialized as white
 - Default opacity
 - 0.2 (data provided)
 - use provided value!!
 - **Remember: you only need to implement *data processing!!***

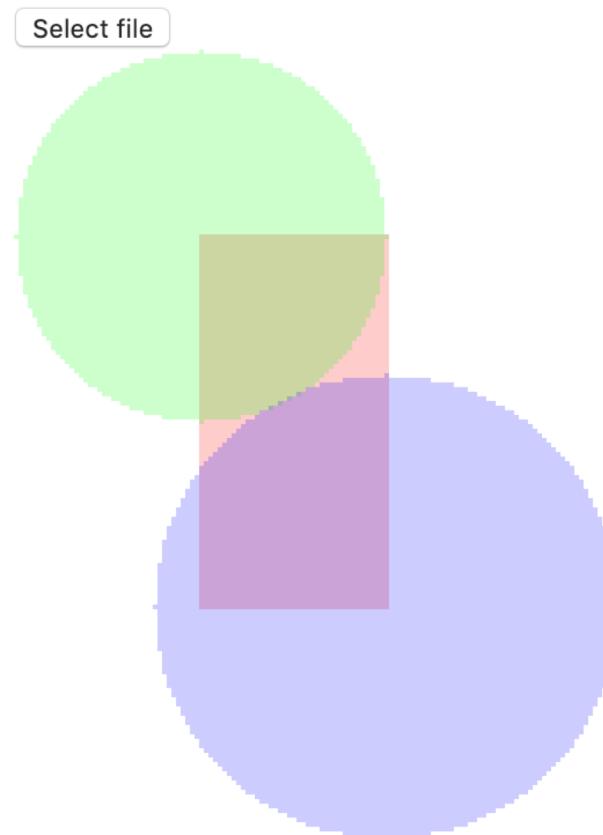
Shape renderer

- You only need to modify `main.cpp`!!
- compile your code with `make all` command
- run program!!
- Upload `picture.txt` file to our [web renderer](#)

Shape Renderer

- Example

```
=====
Add shapes!!
1. Rect
2. Circle
3. Print
INPUT: 1
-----
x: 40
y: 40
width: 40
height: 80
Pick color (red, green, blue)
color: red
Rect added!!
=====
Add shapes!!
1. Rect
2. Circle
3. Print
INPUT: 2
-----
cx: 40
cy: 40
radius: 40
Pick color (red, green, blue)
color: green
Circle added!!
=====
Add shapes!!
1. Rect
2. Circle
3. Print
INPUT: 2
-----
cx: 80
cy: 120
radius: 50
Pick color (red, green, blue)
color: blue
Circle added!!
=====
Add shapes!!
1. Rect
2. Circle
3. Print
INPUT: 3
-----
Printing...
Complete!!
```



Thank you!!

contact: jeonhyun97@postech.ac.kr