Tutorial: C++ basics for OpenCV

Tutorial: C++ basics

Deep Learning Image Processing. Updated. 2024.2

I. Introduction

The OpenCV Library has **>2500** algorithms, extensive documentation, and sample code for real-time computer vision. You can see basic information about OpenCV at the following sites,

- Homepage: https://opencv.org
- Documentation: https://docs.opencv.org
- Source code: https://github.com/opencv
- Tutorial: https://docs.opencv.org/master
- Books: https://opencv.org/books

In this tutorial, you will learn fundamental concepts of the C++ language to use the OpenCV API. You will learn namespace, class, C++ syntax to use image reading, writing and displaying.

OpenCV Example Code
Image File Read / Write / Display

```
#include <iostream>
#include <opencv.hpp>

using namespace std;
using namespace cv;

int main()
{
  /*  read image  */
  String filename1 = "image.jpg";  // class
  Mat img = imread(filename1);  //Mat class
  Mat img_gray = imread("image.jpg", 0);  // read in grayscale

  /*  write image  */
  String filename2 = "writeTest.jpg";  // C++ class/syntax (String, cout, cin)
  imwrite(filename2, img);

  /*  display image  */
  namedWindow("image", WINDOW_AUTOSIZE);
  imshow("image", img);

  namedWindow("image_gray", WINDOW_AUTOSIZE);
  imshow("image_gray", img_gray);

  waitKey(0);
}
```

C++ for OpenCV

OpenCV is provided in C++, Python, Java. We will learn how to use OpenCV in

1. C++ (general image processing)
2. Python (for Deep learning processing)

For C++, we need to learn

- Basic C++ syntax
- Class
- Overloading, namespace, template
- Reference

C++

C++ is a general-purpose programming language created by Bjarne Stroustrup as an **extension of the C programming language**. C++ is portable and can be used to develop applications that can be adapted to multiple platforms. You can see basic C++ tutorials in following site,

- https://www.w3schools.com/cpp/
- https://www.cplusplus.com/doc/tutorial/variables/

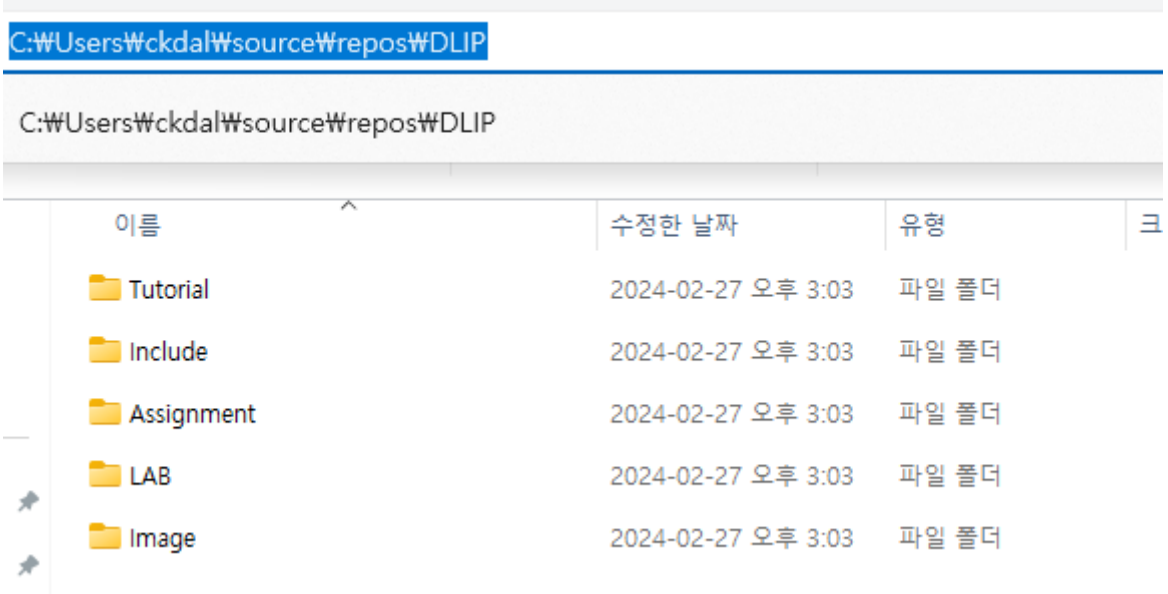
Project Workspace Setting

Create the lecture workspace as **C:\Users\yourID\source\repos**

- e.g. `C:\Users\ykkim\source\repos`

Then, create sub-directories such as :

- `C:\Users\yourID\source\repos\DLIP`
- `C:\Users\yourID\source\repos\DLIP\Tutorial`
- `C:\Users\yourID\source\repos\DLIP\Include`
- `C:\Users\yourID\source\repos\DLIP\Assignment`
- `C:\Users\yourID\source\repos\DLIP\LAB`
- `C:\Users\yourID\source\repos\DLIP\Image`



Define and Declare Functions in Header Files

We will learn how to declare and define functions in the header file

Exercise 1

1. Create a new C++ project in Visual Studio Community
    - Project Name: `DLIP_Tutorial_C++_Ex1`
    - Project Folder: `C:\Users\yourID\source\repos\DLIP\Tutorial\`
2. Create a new C+ source file
    - File Name: `DLIP_Tutorial_C++_Ex1.cpp`
3. Create new header files
    - File Names: `TU_DLIP.h`, `TU_DLIP.cpp`
    - Lib Folder: `C:\Users\yourID\source\repos\DLIP\Include\`
4. Declare the sum function in the header file(`TU_DLIP.h`) as

```
int sum(int val1, int val2);
```

5. Define the sum function in the header source file(`TU_DLIP.cpp`) as

```
int sum(int val1, int val2){...}
```

6. Include the header file in the main source code of `DLIP_Tutorial_C++_Ex1.cpp`.
7. Modify the source main code to print the sum value of any two numbers
8. Compile and run.

{% tabs %}
{% tab title="DLIP_Tutorial_C++_Ex1.cpp" %}

```
//#include "TU_DLIP.h"
#include "../../../Include/TU_DLIP.h"

#include <iostream>

int main()
{
    // ==============================
    // Exercise 1 :: Define Function
    // ==============================
    int val1 = 11;
    int val2 = 22;

    // Add code here


    std::cout << out << std::endl;
}
```

{% endtab %}
{% tab title="TU_DLIP.h" %}

```
#ifndef _TU_DLIP_H      // same as "#if !define _TU_DLIP_H" (or #pragma once)
#define _TU_DLIP_H

#include <iostream>

// ==============================
// Exercise 1 :: Define Function
// ==============================

// Add code here

#endif // !_TU_DLIP_H
```

{% endtab %}

{% tab title="TU_DLIP.cpp" %}

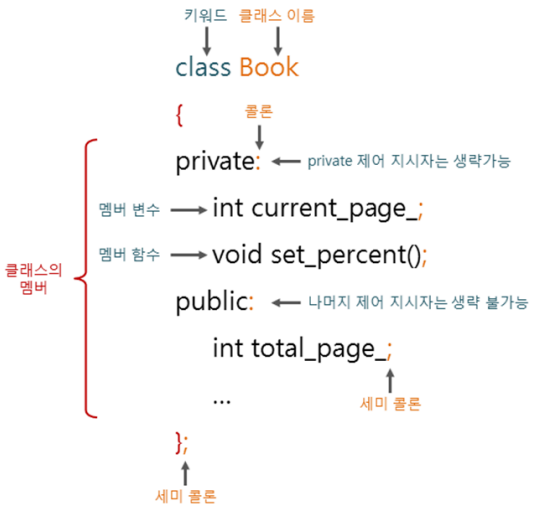
```
#include "TU_DLIP.h"

#include <iostream>

// ==============================
// Exercise 1 :: Define Function
// ==============================

int sum(int val1, int val2)
{
    // Add code here
}
```

{% endtab %}
{% endtabs %}

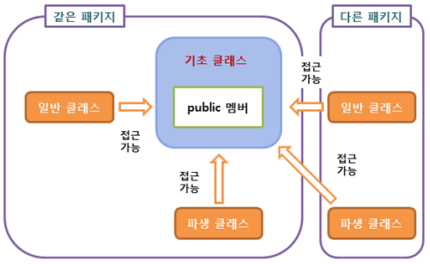
C++ Class

Class is similar to C structure:

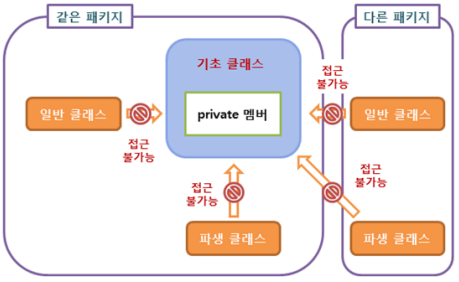
- Structure: Cannot inclue functions. Only variables
- Class: Can include variables, functions definition/declaration, other classes



Public:



Private:



{% tabs %}
{% tab title="Structure (C)" %}

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
  char number[20];
  char password[20];
  char name[20];
  int balance;
}Account;
```

{% endtab %}

{% tab title="Class (C++)" %}

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

/*  Class Definition  */
class Account{
  public:
    char number[20];
    char password[20];
    char name[20];
    int balance;
```

```
    void deposit(int money);  // Can include functions
    void withdraw(int money);  // Can include functions
};

/*  Class Function Definition  */
void Account::deposit(int money){
  balance += money;
}
void Account::withdraw(int money){
  balance -= money;
}
```

{% endtab %}
{% endtabs %}

Class Constructor

Constructor is **special method** automatically called when an object of a class is created.

- Use the **same** name as the class, followed by parentheses **()**
- It is always **public**.
- It does not have any return values.

```
class MyNum{
  public:
    MyNum();  // Constructor 1
    MyNum(int x);  // Constructor 2

    int num;
};

// Class Constructor 1
MyNum::MyNum(){}

// Class Constructor 2
MyNum::MyNum(int x)
{
  num = x;
}

int main(){
  // Creating object by constructor 1
  MyNum mynum;
  mynum.num = 10;

  // Creating object by constructor 2
  MyNum mynum2(10);
}
```

Mat Class in OpenCV

The image data are in forms of 1D, 2D, 3D arrays with values 0~255 or 0~1

OpenCV provides the Mat class for operating multi dimensional images

Example

Printing out informations about the source image using OpenCV

```
#include "opencv.hpp"
#include <iostream>

using namespace cv;
using namespace std;

int main(int argc, char* argv[])
{
  cv::Mat src, gray, dst;
  src = cv::imread("testImage.jpg");

  if (src.empty())
    std::cout << "src is empty!!" << std::endl;

  // Print result
  std::cout << "is empty? \t: " << src.empty() << std::endl;
  std::cout << "channels \t: " << src.channels() << std::endl;
  std::cout << "row of src \t: " << src.rows << std::endl;
  std::cout << "col of src \t: " << src.cols << std::endl;
  std::cout << "type of src \t: " << src.type() << std::endl;

  cv::namedWindow("src");
  cv::imshow("src", src);

  cv::waitKey(0);
}
```

Results

Exercise 2

1. Create a new C++ project in Visual Studio Community

- Project Name: `DLIP_Tutorial_C++_Ex2`
- Project Folder: `C:\Users\yourID\source\repos\DLIP\Tutorial\`

2. Create a new C+ source file

- File Name: `DLIP_Tutorial_C++_Ex2.cpp`

Create a Class 'MyNum'

3. Modify the header file `TU_DLIP.h` and `TU_DLIP.cpp` to declare a class member named as **MyNum**.

- Constructor : MyNum()
- Member variables: val1, val2 // integer type
- Member functions: int sum() // returns the sum of val1 and val2

- Member functions: void print() // prints values of **val1, val2, and sum**

4. Then, compile and run the program.

{% tabs %}
{% tab title="DLIP_Tutorial_C++_Ex2.cpp" %}

```
//#include "TU_DLIP.h"
#include "../../../Include/TU_DLIP.h"

int main()
{
    // ==============================
    // Exercise 1: Define Function
    // ==============================

    int val1 = 11;
    int val2 = 22;

    int out = sum(val1, val2);

    std::cout << out << std::endl;

    // ====================================
    // Exercise 2: Create a Class 'MyNum'
    // ====================================

    MyNum mynum(10, 20);
    mynum.print();

}
```

{% endtab %}

{% tab title="TU_DLIP.h" %}

```
#ifndef _TU_DLIP_H      // same as "#if !define _TU_DLIP_H" (or #pragma once)
#define _TU_DLIP_H

#include <iostream>

// ==============================
// Exercise 1: Define Function
// ==============================

int sum(int val1, int val2)
{
    return val1 + val2;
}


// ====================================
// Exercise 2: Create a Class "MyNum"
// ====================================

// Declare Constructor, function(sum, print), variable(val1, val2)
```

```
class MyNum
{
    // Add code here
};

#endif // !_TU_DLIP_H
```

{% endtab %}

{% tab title="TU_DLIP.cpp" %}

```
#include "TU_DLIP.h"

#include <iostream>

// ==============================
// Exercise 1: Define Function
// ==============================

int sum(int val1, int val2)
{
    return val1 + val2;
}

// ====================================
// Exercise 2: Create a Class "MyNum"
// ====================================

// Constructor: x1 -> val1, x2 -> val2
MyNum::MyNum(int x1, int x2)
{
    // Add code here
}

int MyNum::sum(void)
{
    // Add code here
}

void MyNum::print(void)
{
    // Add code here
}
```

{% endtab %}
{% endtabs %}

Solution

{% tabs %}
{% tab title="TU_DLIP_solution.h" %}

```
#ifndef _TU_DLIP_H      // same as "#if !define _TU_DLIP_H" (or #pragma once)
#define _TU_DLIP_H
```

```
#include <iostream>

// ==============================
// Exercise 1: Define Function
// ==============================

int sum(int val1, int val2);


// ====================================
// Exercise 2: Create a Class "MyNum"
// ====================================

class MyNum
{
    public:
        MyNum(int x1, int x2);
        int val1;
        int val2;
        int sum(void);
        void print(void);
};

#endif // !_TU_DLIP_H
```

{% endtab %}

{% tab title="TU_DLIP_solution.cpp" %}

```
#include "TU_DLIP.h"

#include <iostream>

// ==============================
// Exercise 1: Define Function
// ==============================

int sum(int val1, int val2)
{
    int out = val1 + val2;

    return out;
}

// ====================================
// Exercise 2: Create a Class "MyNum"
// ====================================

// Constructor: x1 -> val1, x2 -> val2
MyNum::MyNum(int x1, int x2)
{
    val1 = x1;
    val2 = x2;
}
```

```
int MyNum::sum(void)
{
    return val1 + val2;
}

void MyNum::print(void)
{
    std::cout << "MyNum.val1: " << val1 << std::endl;
    std::cout << "MyNum.val2: " << val2 << std::endl;
    std::cout << "Sum : " << sum() << std::endl;
}
```

{% endtab %}
{% endtabs %}

Namespace

A namespace provides a scope to the identifiers (the names of types, functions, variables, etc) inside it.

- Uses **::** as scope resolution operator
- Use **namespace** in order to avoid collision using functions with the same name e.g. KimHandong --> Student::KimHandong, TA::KimHandong

```
// Method 1. Calling specific function(recommended)
int main(void){
  project_A::add_value(3, 7);
  project_A::subtract_value(10, 2);
  return 0;
}

// Method 2. Calling all functions in the namespace
using namespace project_A;

int main(void){
  add_value(3, 7);
  subtract_value(10, 2);
  return 0;
}
```

- **std::cout, std::cin, std::endl** are also defined in **iostream**

```
// Method 1
std::cout<<"print this"<<std::endl;

// Method 2
using namespace std
cout<<"print this"<<endl;
```

Exercise 3

Create another Class 'MyNum'

In this exercise, you create the `MyNum` class, previously implemented in **Exercise 2**, with the same class name in different namespaces, `proj\_A`, and `proj\_B`.

1. Create a new C++ project in Visual Studio Community

- Project Name: `DLIP_Tutorial_C++_Ex3`
- Project Folder: `C:\Users\yourID\source\repos\DLIP\Tutorial\`

2. Create a new C+ source file

- File Name: `DLIP_Tutorial_C++_Ex3.cpp`

3. Modify the header file `TU_DLIP.h` and `TU_DLIP.cpp` to declare two class members named as **MyNum** in `proj\_A` and `proj\_B`.

4. Use namespace to identify two classes clearly
    - First **MyNum** class: namespace name **proj_A**
    - Second **MyNum** class: namespace name **proj_B**

5. Also, declare class member variables for each MyNum class: **Constructor / val1 / val2 / val3 / sum / print**
    - Constructor `MyNum(int in1, int in2, int in3)`: A constructor for specifying values `val1`, `val2`, `val3`
    - `val1`, `val2`, `val3`: member variable of integer type
    - `sum(void)`: member function that returns the sum of val1, val2, and val3
    - `print(void): member function that prints val1, val2, val3, and sum

{% tabs %}
{% tab title="DLIP_Tutorial_C++_Ex3.cpp" %}

```
#include "../../../Include/TU_DLIP.h"
#include "TU_DLIP.h"

void main()
{
    proj_A::MyNum mynum1(1, 2, 3);
    proj_B::MyNum mynum2(4, 5, 6);

    mynum1.print();
    mynum2.print();

    system("pause");
}
```

{% endtab %}

{% tab title="TU_DLIP.h" %}

```
#ifndef _TU_DLIP_H      // same as "#if !define _TU_DLIP_H" (or #pragma once)
```

```
#define _TU_DLIP_H

#include <iostream>

// ============================
// Exercise 1: Define Function
// ============================

int sum(int val1, int val2);

// ====================================
// Exercise 2: Create a Class "MyNum"
// ====================================

class MyNum
{
public:
    MyNum(int x1, int x2);
    int val1;
    int val2;
    int sum(void);
    void print(void);
};

// ========================================================
// Exercise 3: Create two Class "MyNum" in proj_A, proj_B
// ========================================================
namespace proj_A
{
    // Add code here
}

namespace proj_B
{
    // Add code here
}
#endif // !_TU_DLIP_H
```

{% endtab %}

{% tab title="TU_DLIP.cpp" %}

```
#include "TU_DLIP.h"

#include <iostream>

// ============================
// Exercise 1: Define Function
// ============================

int sum(int val1, int val2)
{
    return val1 + val2;
}
```

```cpp
// ====================================
// Exercise 2: Create a Class "MyNum"
// ====================================

MyNum::MyNum(int x1, int x2)
{
    val1 = x1;
    val2 = x2;
}

int MyNum::sum(void)
{
    return val1 + val2;
}

void MyNum::print(void)
{
    std::cout << "MyNum.val1 : " << val1 << std::endl;
    std::cout << "MyNum.val2 : " << val2 << std::endl;
    std::cout << "Sum : " << sum() << std::endl;
}

// ============================================================
// Exercise 3: Create two Class "MyNum" in proj_A, proj_B
// ============================================================
proj_A::MyNum::MyNum(int x1, int x2, int x3)
{
    // Add code here
}

int proj_A::MyNum::sum(void)
{
    // Add code here
}

void proj_A::MyNum::print(void)
{
    // Add code here
}

proj_B::MyNum::MyNum(int x1, int x2, int x3)
{
    // Add code here
}

int proj_B::MyNum::sum(void)
{
    // Add code here
}

void proj_B::MyNum::print(void)
{
    // Add code here
}
```

{% endtab %}
{% endtabs %}

{% tabs %}
{% tab title="TU_DLIP_Solution.h" %}

```
#ifndef _TU_DLIP_H      // same as "#if !define _TU_DLIP_H" (or #pragma once)
#define _TU_DLIP_H

#include <iostream>

// ==============================
// Exercise 1: Define Function
// ==============================

int sum(int val1, int val2);

// ====================================
// Exercise 2: Create a Class "MyNum"
// ====================================

class MyNum
{
public:
    MyNum(int x1, int x2);
    int val1;
    int val2;
    int sum(void);
    void print(void);
};

// ========================================================
// Exercise 3: Create two Class "MyNum" in proj_A, proj_B
// ========================================================
namespace proj_A
{
    class MyNum
    {
    public:
        MyNum(int x1, int x2, int x3);
        int val1;
        int val2;
        int val3;
        int sum(void);
        void print(void);
    };
}

namespace proj_B
{
    class MyNum
    {
    public:
        MyNum(int x1, int x2, int x3);
        int val1;
```

```
        int val2;
        int val3;
        int sum(void);
        void print(void);
    };
}
#endif // !_TU_DLIP_H
```

{% endtab %}

{% tab title="TU_DLIP_solution.cpp" %}

```
#include "TU_DLIP.h"

#include <iostream>

// ============================
// Exercise 1 :: Define Function
// ============================

int sum(int val1, int val2)
{
    return val1 + val2;
}

// ====================================
// Exercise 2 :: Create a Class "MyNum"
// ====================================

MyNum::MyNum(int x1, int x2)
{
    val1 = x1;
    val2 = x2;
}

int MyNum::sum(void)
{
    return val1 + val2;
}

void MyNum::print(void)
{
    std::cout << "MyNum.val1 : " << val1 << std::endl;
    std::cout << "MyNum.val2 : " << val2 << std::endl;
    std::cout << "Sum : " << sum() << std::endl;
}

// ========================================================
// Exercise 3: Create two Class "MyNum" in proj_A, proj_B
// ========================================================
proj_A::MyNum::MyNum(int x1, int x2, int x3)
{
    val1 = x1;
    val2 = x2;
    val3 = x3;
```

```
}

int proj_A::MyNum::sum(void)
{
    return val1 + val2 + val3;
}

void proj_A::MyNum::print(void)
{
    std::cout << "MyNum.val1 : " << val1 << std::endl;
    std::cout << "MyNum.val2 : " << val2 << std::endl;
    std::cout << "MyNum.val3 : " << val3 << std::endl;
    std::cout << "Sum : " << sum() << std::endl;
}

proj_B::MyNum::MyNum(int x1, int x2, int x3)
{
    val1 = x1;
    val2 = x2;
    val3 = x3;
}

int proj_B::MyNum::sum(void)
{
    return val1 + val2 + val3;
}

void proj_B::MyNum::print(void)
{
    std::cout << "MyNum.val1 : " << val1 << std::endl;
    std::cout << "MyNum.val2 : " << val2 << std::endl;
    std::cout << "MyNum.val3 : " << val3 << std::endl;
    std::cout << "Sum : " << sum() << std::endl;
}
```

{% endtab %}
{% endtabs %}

Template

A template can make a variable type(int, float, char..) as a variable.
How can you use the same function but with a different number type as the input argument?
: add(float A, float B), add(int A, int B) `&rarr` add(T A, T B) where T=int or T=float

C++, pasted just now:

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

int add_value(int A, int B){
     int result;
     return result=A+B;

}

double add_value(double A, double B){
     double result;
     return result=A+B;
}

int main(void){
   cout<<add_value(13 , 7)<<endl;
   cout<<add_value(12.5 , 7.5)<<endl;
   return 0;
}
```

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

template <typename T>
T add_value(T A, T B){
     return A+B;
}

int main(void) {
     cout<<add_value(13,7)<<endl;
     cout<<add_value(12.5,7.5)<<endl;
     return 0;
}
```

Make function

Without Data type

Output:

```
20
20
```

stdout

```
20
20
```

Function Overloading

Functions with the same name (but with different types or number of parameters) can be defined.

- Different return type (with everything else the same) is not a function overloading.

C, pasted just now:

```c
int add_value(int A, int B){
     int result;
     return result=A+B;
}

double add_value(double A, double B){
     double result;
     return result=A+B;
}

int main(void){
   add_value(13 , 7);
   add_value(12.5 , 7.5);
   return 0;
}
```

SAME NAME

Output: Error

```
Line 8: error: conflicting types for 'add_value'
Line 2: error: previous definition of 'add_value' was here
```

C++, pasted just now:

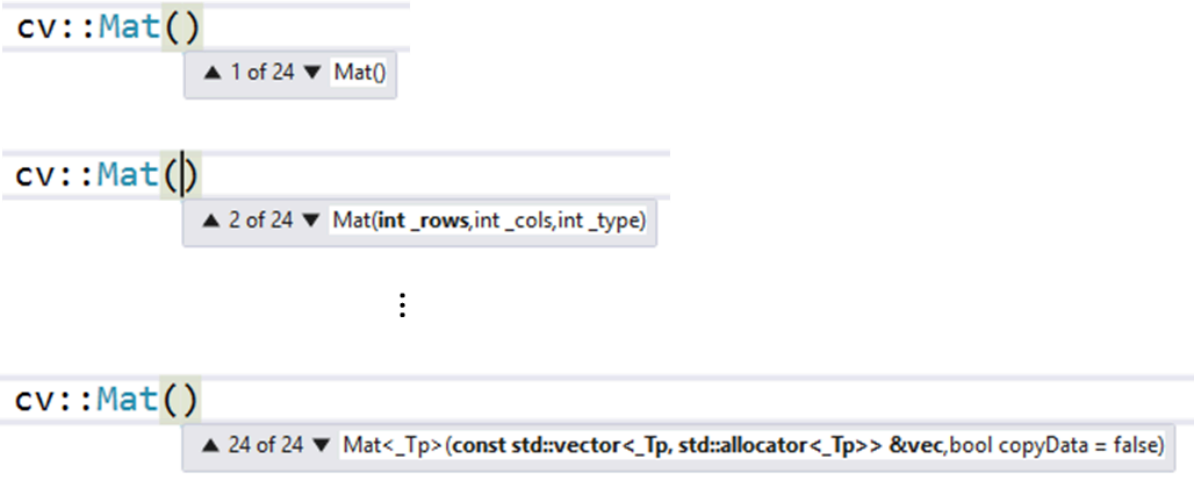
```cpp
#include <iostream>
using namespace std;

int add_value(int A, int B){
     int result;
     return result=A+B;

}

double add_value(double A, double B){
     double result;
     return result=A+B;
}

int main(void){
   cout<<add_value(13 , 7)<<endl;
   cout<<add_value(12.5 , 7.5)<<endl;
   return 0;
}
```

Output:

```
20
20
```

Example

`cv::Mat` can be created in many different ways. Use the up or down the keyboard to see what the options are.



Function Overloading Reference

```
#include <opencv2/opencv.hpp>
#include <iostream>

cv::Mat cvtGray(const cv::Mat color);
void cvtGray(cv::Mat color, cv::Mat & gray);

void main() {
    cv::Mat src, gray, dst;
    src = cv::imread("image.jpg");

    if (src.empty())
        std::cout << "src is empty!!" << std::endl;

    cvtGray(src, gray);

    cv::namedWindow("src");
    cv::imshow("src", src);

    cv::namedWindow("gray");
    cv::imshow("gray", gray);

    cv::waitKey(0);
}

cv::Mat cvtGray(cv::Mat color)
{
    cv::Mat gray;
    color = cv::Mat::zeros(color.size(), CV_8UC3);
    cv::cvtColor(color, gray, CV_RGB2GRAY);
    return gray;
}

void cvtGray(cv::Mat color, cv::Mat gray)
{
    cv::cvtColor(color, gray, CV_RGB2GRAY);
    cv::namedWindow("inside_function");
    cv::imshow("inside_function", gray);
}
```

Default Parameter

- Default value setting

```
#include <iostream>

int add_value(int A, int B=5)
{
    int result=A+B;
    cout<<result<<": result of add_value function"<<endl;
    return result;
}

int main(void) {
    add_value(10);
    add_value(10,7);
    return 0;
}
```

Output:

```
15: result of add_value function
17: result of add_value function
```

Parameter value can be predetermined

```
#include <iostream>

int add_value(int A, int B=5);

int main(void) {
    add_value(10);
    add_value(10,7);
    return 0;
}

int add_value(int A, int B)
{
    int result=A+B;
    cout<<result<<endl;
    return result;
}
```

If using function declaration before its definition, Default value must be defined **in declaration**

- Ambiguity problem

```
#include <iostream>

int add_value(int A=3, int B=3)
{
    int result=A+B;
    cout<<result<<endl;
    return result;
}

int add_value(void)
{
    return 1;
}

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

It is possible to make functions w/ same name (∵function overloading)

BUT!!

Complier doesn't know which function user wants to use

Output:

```
In function 'int main()':
Line 16: error: call of overloaded 'add_value()' is ambiguous
compilation terminated due to -Wfatal-errors.
```

Ambiguity problem must be avoided when setting a default value

Default parameter in OpenCV

```
cv::imread()
```

```
cv::Mat imread(const std::string &filename, int flags = 1)
```

```
src = cv::imread("testImage.jpg");
```

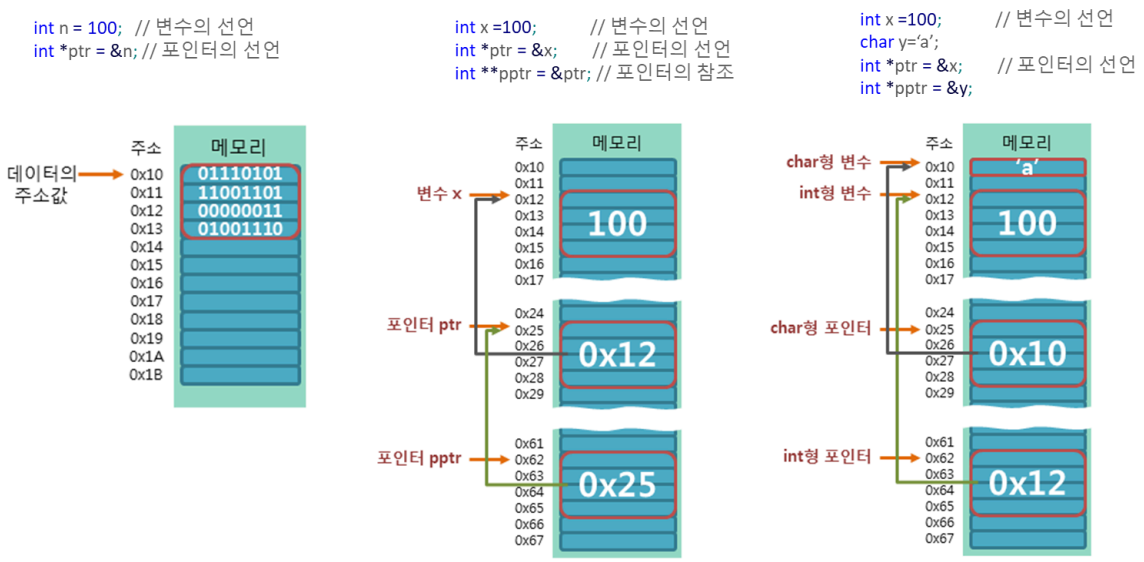
flags == 1

```
src = cv::imread("testImage.jpg",0);
```

flags == 0

Pointer

A **pointer** is a variable whose value is the address of another variable.



What are Pointers?

A **pointer** is a variable whose value is the address of another variable.
i.e. direct address of the memory locations
Pointers are the basis for data structures.

- **Define** a pointer variable `int *ptr;`
- **Assign** the address of a variable to a pointer `ptr = &var`
- **Access** the value at the address available in the pointer variable `int value = *ptr`

Example

```
#include <iostream>

void swap(int *a, int *b);

int main() {
    int val1 = 10;
    int val2 = 20;

    printf("Before SWAP operation \n");
    printf("val1: %d \n", val1);
    printf("val2: %d \n", val2);

    swap(&val1, &val2);

    printf("After SWAP operation \n");
    printf("val1: %d \n", val1);
    printf("val2: %d \n", val2);

    system("pause");
    return 0;
}

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
```

```
}
```

Reference

Reference can replace the use of pointer *****

- The disadvantage of reference is "We do not know if a function is call-by-value or call-by-reference"

```
// C/C++ syntax (Pointer)
#include <iostream>

void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main(void){
    int value1 = 10;
    int value2 = 20;

    std::cout << "value1 :" << value1 << "   ";
    std::cout << "value2 :" << value2 << std::endl;

    swap(&value1, &value2);

    std::cout << "value1 :" << value1 << "   ";
    std::cout << "value2 :" << value2 << std::endl;

    return 0;
}

// C++ only (Reference)
#include <iostream>

void swap(int &a, int &b){
    int temp = a;
    a = b;
    b = temp;
}

int main(void){
    int value1 = 10;
    int value2 = 20;

    std::cout << "value1 :" << value1 << "   ";
    std::cout << "value2 :" << value2 << std::endl;

    swap(value1, value2);

    std::cout << "value1 :" << value1 << "   ";
    std::cout << "value2 :" << value2 << std::endl;

    return 0;
```

```
}
```