

第5回C++輪講

第5回の内容

豆知識のような話

- [型](#)
- [for文](#)
- [文字コード](#)
- [画素へのアクセススピード比較](#)
- [三項演算子](#)
- [デフォルト引数](#)
- [using](#)
- [class](#)
- [typedef](#)
- [構造体](#)
- [template](#)
- [コンテナ](#)
- [enum型](#)
- [enum class](#)
- [constexpr](#)
- [マクロ](#)
- [main引数](#)
- [ファイル出力 cv::FileStorage](#)
- [algorithmヘッダ](#)
- [pairヘッダとtupleヘッダ](#)
- [filesystemヘッダ](#)
- [並列処理](#)
- [外部ライブラリ](#)
- [Outro](#)
- [第5回課題](#)
- [自由課題](#)

型

[基本型](#) - [cppreference.com](#)

[cstdint](#) - [cpprefjp C++日本語リファレンス](#)

for文

今までに説明してきたfor(int i = ...)は潜在的にミスをする可能性が高いため、イテレータやmapで表現したほうがよい

[範囲for文](#) - [cppreference.com](#)

```
std::vector<int> v;

for (const auto& e : v) {
    std::cout << e << std::endl;
}
```

文字コード

本資料ではわかりやすさを重視するため日本語のコメントをしてきたが、日本語のコメントには気を付けなければならない

気を付けなければ、文字コードがSJIS、コメントが日本語で書かれたファイルが出来上がる

SJISは文字化けを引き起こし、トラブルの原因となる

トラブルを避けるためにも、

- 文字コードをUTF-8(or UTF-16)にした上で日本語で必要なコメントをする
- 文字コードをUTF-8にした上で英語で最低限のコメントをする
- 最低限のコメントで理解できるようなコードを書く

などを意識しよう

画素へのアクセススピード比較

[\[OpenCV\]\[Mat\]画素へのアクセススピード比較 - Qiita](#)

本資料ではcv::Pointクラスを用いる方法で画像処理を行ってきた

より高速な画素アクセス方法もあるため参考にすること

三項演算子

三項演算子はCの演算子の一つである

```
z = x > y ? x : y;
```

と

```
if(x > y) z = x;
else z = y;
```

は等しい

無理に三項演算子を使用すると可読性が下がるため要注意

デフォルト引数

関数呼び出し時、引数を渡さなくても事前設定値が自動で用いられる機能

関数の**プロトタイプ宣言**時にデフォルト引数として、事前に値を与える

デフォルト引数の**右側**に、デフォルト引数なしの変数を定義できない

```

#include <iostream>
int times(int x,      int y = 100); // ok
int minus(int x = 10, int y = 100); // ok
// int plus(int x = 10, int y);      // error
int main()
{
    std::cout << minus(100, 10) << std::endl;
    std::cout << minus(100)      << std::endl;
    std::cout << minus()          << std::endl;
    std::cout << times(100)       << std::endl;
    return 0;
}
int times(int x, int y)
{
    return x * y;
}
int minus(int x, int y)
{
    return x - y;
}

```

using

using宣言にはいくつかの使い方があり、うまく活用すれば冗長なタイプを減らすことができる

しかし、これらは用法を誤れば問題の原因となるため気を付けなければならない

全ての用法において、**スコープを用いて使用範囲を明示的に限定するべき**である

特に**usingディレクティブ**は名前空間の存在意義を無に帰しているため、**通常使うことはないはず**である

また、使うことがあったとしても**ヘッダに記述してはならない**

ヘッダに記述した場合、**そのヘッダをインクルードした全てのファイルが汚染される**

```

#include <iostream>
template<typename T>
using Vec = std::vector<T>;
int main()
{
    // using宣言
    {
        using std::cout;
        cout << "using宣言" << std::endl;
    }

    // usingディレクティブ
    {
        using namespace std;
        cout << "usingディレクティブ" << endl;
    }

    // エイリアス宣言
    {
        using vec2 = std::vector<std::vector<int>>;
    }
}

```

```

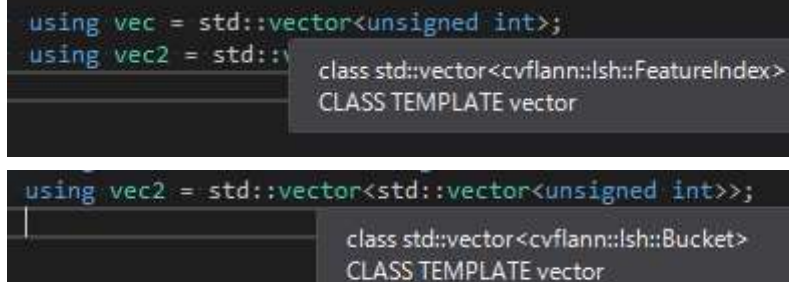
        vec2 temp(5);
        std::cout << temp.size() << std::endl;
    }

    // エイリアステンプレート
    {
        // 1,2行目の記述を利用
        Vec<int> temp(5);
        std::cout << temp.size() << std::endl;
    }
    return 0;
}

```

エイリアステンプレート - cpprefjp C++日本語リファレンス

悪い例



```

using vec = std::vector<unsigned int>;
using vec2 = std::vector<unsigned int>;

```

class std::vector<cvflann::lsh::FeatureIndex>
CLASS TEMPLATE vector

```

using vec2 = std::vector<std::vector<unsigned int>>;

```

class std::vector<cvflann::lsh::Bucket>
CLASS TEMPLATE vector

class

classはC++の機能である

既に扱ってきたstd::stringやcv::Matのようなものである

classを作るときはオブジェクト生成時に呼ばれるコンストラクタ, 演算子のオーバーロードの定義が推奨される

キーワード: コンストラクタ, デストラクタ, 初期化リスト, メンバ変数, メンバ関数, インスタンス, 継承

クラスの使用例

```

#include <iostream>
#include <string>
class MyClass
{
public:
    MyClass() = default;
    MyClass(int num1, int num2) :_num1(num1), _num2(num2) {}
    MyClass(std::string str) :_str(str) {}
    ~MyClass() = default;
    int sum();
    static int sum(int num1, int num2);
    std::string hi = "hi";
    std::string hello();

private:

```

```

        std::string _str;
        int _num1, _num2, _sum;
    };
    int MyClass::sum()
    {
        this->_sum = _num1 + _num2;
        return this->_sum;
    }
    int MyClass::sum(int num1, int num2)
    {
        return num1 + num2;
    }
    std::string MyClass::hello()
    {
        return _str;
    }

    int main(int argc, char *argv[])
    {
        std::cout << MyClass::sum(1, 3) << std::endl;
        MyClass classA("hello");
        std::cout << classA.hi << std::endl;
        std::cout << classA.hello() << std::endl;
        MyClass classB(3, 5);
        std::cout << classB.sum() << std::endl;
        std::cout << classB.sum(1, 3) << std::endl;
        return 0;
    }

```

typedef

Cでデータ型に名前をつけるときに用いる

C++ではエイリアス宣言を使おう

```

typedef unsigned char uchar; // C
using uchar = unsigned char; // C++11以上ではこちらを推奨

```

構造体

構造体はCの機能である

C++ではclassと同様の機能となっている

```

// 構造体の宣言
struct bgrimage {
    unsigned char B;
    unsigned char G;
    unsigned char R;
};
struct bgrimage image; // struct bgrimage型の変数 image を作成

```

```
typedef struct bgrimage {
    unsigned char B;
    unsigned char G;
    unsigned char R;
} BGRIMAGE;
BGRIMAGE image; // BGRIMAGE型の変数 image を作成
```

template

templateはC++の新しい機能である

型をパラメータとして与えたい場合に用いる

Cで複数の型に対応させたい場合にはオーバーロードを用いるしかないが、C++ではtemplateで対応できる
可読性が悪いことが問題点として挙げられるため用法に注意する必要がある

キーワード「テンプレートメタプログラミング」

```
#include <iostream>
// use overload
int max(int x, int y);
double max(double x, double y);
// use template
template<typename T>
T tmax(T x, T y)
{
    return x > y ? x : y;
}
int main()
{
    std::cout << max(5, 3) << std::endl;
    std::cout << tmax<int>(5, 3) << std::endl;
    return 0;
}
int max(int x, int y)
{
    return x > y ? x : y;
}
double max(double x, double y)
{
    return x > y ? x : y;
}
```

コンテナ

C++にはvector以外にも、list、queue、deque、map、set、bitset、stackなどがある。

[C++] STLの型の使い分け

enum型

enum (列挙) 型はC/C++にある機能である

```
#include <iostream>
int main()
{
    enum color
    {
        blue, green, red,
    };
    color channel;
    std::cout << color::blue << std::endl;
    std::cout << color::green << std::endl;
    std::cout << color::red << std::endl;

    channel = red;
    std::cout << channel << std::endl;

    for (int i = 0; i < 3; i++) {
        switch (i) {
            case color::blue:
                std::cout << "blue" << std::endl;
                break;
            case color::green:
                std::cout << "green" << std::endl;
                break;
            case color::red:
                std::cout << "red" << std::endl;
                break;
        }
    }
    return 0;
}
```

enum class

enum classはC++11で追加された機能である。 enumとenum classがあるが、似て異なるものなので注意すること

```
#include <iostream>
int main()
{
    enum class color
    {
        blue, green, red,
    };
    for (int i = 0; i < 3; i++) {
        switch (static_cast<Color>(i)) {
            case Color::blue:
                std::cout << "blue" << std::endl;
                break;
            case Color::green:
```

```

        std::cout << "green" << std::endl;
        break;
    case Color::red:
        std::cout << "red" << std::endl;
        break;
    }
}
return 0;
}

```

constexpr

constexprはC++の機能である

コンパイル時に定数として扱われ、あらかじめ計算される

```

constexpr double pi = 3.14;
constexpr double hoge = pi*2; //コンパイル時に計算

```

マクロ

#defineなど#から始まるものはマクロと呼ばれる

Cでは定数などはマクロで定義されていることが多いがC++では推奨されていない

```

#define PI 3.14 //c
constexpr double PI = 3.14 //cpp

```

Visual Studioでは_MSC_VERでコンパイラのバージョンを取得できる

異なる環境でも実行可能にするためには環境ごとにコーディングする必要がある

main引数

```

#include <iostream>
int main(int argc, char* argv[])
{
    std::cout << argc << std::endl;
    for (int i = 0; i < argc; ++i)
    {
        std::cout << argv[i] << std::endl;
    }
    return 0;
}

```

コンパイルし, project.exeを出力, 実行


```
C:\Users\~~\x64\Release>project.exe
1
project.exe
```

project.exeに引数を与えて実行

```
C:\Users\~~\x64\Release>project.exe aaa bbb 1234
4
project.exe
aaa
bbb
1234
```

このようにmain引数ができる

よくあるコマンドオプションのようにするには外部ライブラリのparcerを利用

OpenCVにもcv::CommandLineParserがある

ファイル出力

- csv
Comma-Separated Valuesの頭文字
数値や文字列の間にコンマを挟み, .csvという拡張子で保存するとエクセルなどで表示ができる
output.txtを作成し, 下記内容を保存後output.csvにリネームしてみよう

```
0,1,2,3,aaa
bbb,ccc
1234,
```

- xml
Extensible Markup Language
- yaml/yml
YAML Ain't a Markup Language
- json
JavaScript Object Notation

OpenCVにcv::FileStorageというクラスが用意されており, json xml yaml形式で出力できる
一長一短なので各自調べる

[File Input and Output using XML and YAML files — OpenCV 3.0.0-dev documentation](#)

GitHubで公開されているjsonパーサ等を利用してもよい

json>yaml>>>xmlという印象

VS Codeなどのアプリケーションでは設定ファイルがjsonとなっておりコメントできるが, 仕様上はコメント構文がない(RFC4627)

そのため, 設定ファイルにコメントを残したいのであればyamlが妥当か

yamlのコード例

```

int num(0);
std::vector<int> ary {0, 1, 2};
cv::Size size(20, 30);
cv::Mat1b img(cv::Mat1b::zeros(size));
cv::FileStorage fs("parameter.yaml", cv::FileStorage::WRITE);
fs << "num" << num;
fs << "ary" << ary;
fs << "size" << size;
fs << "img" << img;
fs.release();

```

```

int num;
std::vector<int> ary;
cv::Size size;
cv::Mat1b img;
cv::FileStorage fs("parameter.yaml", cv::FileStorage::READ);
fs["num"] >> num;
fs["ary"] >> ary;
fs["size"] >> size;
fs["img"] >> img;
fs.release();

```

```

std::string jsonPath("test.json");
cv::FileStorage fs_w(jsonPath, cv::FileStorage::WRITE);
fs_w << "data";
fs_w << "[";
for (int i = 0; i < 10; ++i) {
    fs_w << "{"
        << "id" << i
        << "id2" << i*11
        << "}";
}
fs_w << "]";
fs_w.release(); // デストラクタ呼び出し. なくても動くときは動く

cv::FileStorage fs_r(jsonPath, cv::FileStorage::READ);
if (!fs_r.isOpened()) {
    std::cerr << "failed to open " << std::endl;
}
std::cout << fs_r["data"].size() << std::endl;
std::cout << "-----" << std::endl;
for (int i = 0; i < fs_r["data"].size(); i++) {
    std::cout << i << std::endl;
    std::cout << static_cast<int>(fs_r["data"][i]["id"]) << std::endl;
    std::cout << static_cast<int>(fs_r["data"][i]["id2"]) << std::endl;
    std::cout << "-----" << std::endl;
}
fs_r.release();

```

algorithm

sortや第2回で作成したswap関数などは<algorithm>ヘッダで提供されている

pairヘッダとtupleヘッダ

vectorは一つの型をまとめて扱えるが複数の型を扱えない

複数の型をまとめて扱いたいときは<pair>ヘッダ, <tuple>ヘッダ, もしくはclassを使用する

filesystemヘッダ

C++17よりサポートされる

<filesystem>

2018年現在は名前空間がstd::experimental::filesystemとなっている

正式サポートされたときはstd::filesystemになりそう

ヘッダ使用時は**cppファイルに (hppじゃないよ)**

```
using fs = std::experimental::filesystem;
// using fs = std::filesystem;
```

としておくと良さそう

並列処理

- OpenMP
#pragma omp parallel for
Pragma演算子
- std::thread
std::thread
- std::for_each
std::for_each

外部ライブラリ

外部ライブラリとは, OpenCVのような汎用性の高い機能が開発・提供されているものと言う
以下に有名な外部ライブラリを羅列する

- Boost
Boost provides free peer-reviewed portable C++ source libraries.
C++ β版
- Eigen
Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms.
- TensorFlow

Outro

C++, OpenCVについて重要かつ取り組みやすい項目について説明してきた

Class, ポインタといった重要であるが複雑なものや, コンパイラやCPUなどのハードウェアについてはあまり触れていないので少しずつ頑張っていきましょう

第5回課題

the5thCppLecture.hpp, the5thdCppLecture.cppを作成し以下の関数を名前空間t5cl内に作成せよ
関数run()で全ての関数が実行できるようにすること

- ラベリング(8近傍) ラベリングの可視化には下記関数を使用せよ
デバッグにImageWatchを用いてもよい
- 3チャンネルのBGR画像を入力として与え, 3枚のフィルタを設計適用して1チャンネルの画像を3枚得たのち,
さらにそれぞれの出力に3種類のフィルタを新たに設計適用し, 可視化せよ
フィルタ適用前は値域を0~255から-1~1へ正規化すること.
関数名: convolution

```
// src: ラベル番号(0(背景), 1, 2, ...)が入ったcv::Mat1i型
// dst: 可視化したラベリング結果が入るcv::Mat3b型
void name::t5cl::coloring(const cv::Mat1i & src, cv::Mat3b & dst) {
    CV_Assert(!src.empty());
    dst = cv::Mat3b::zeros(src.rows, src.cols);

    int numLabel = 0;
    for (int y = 0; y < src.rows; ++y) {
        for (int x = 0; x < src.cols; ++x) {
            if (numLabel < src(y, x)) {
                numLabel = src(y, x);
            }
        }
    }

    for (int y = 0; y < src.rows; ++y) {
        for (int x = 0; x < src.cols; ++x) {
            if (src(y, x) != 0) {
                int hue = 180 * (src(y, x) - 1) / numLabel;
                dst(y, x) = cv::Vec3b(hue, 255, 255);
            }
        }
    }
    cv::cvtColor(dst, dst, cv::COLOR_HSV2BGR);
}
```

```
// void run(const int num = 0, const std::string & imagePath =
"sample_images/lena.bmp", const std::string & hirakawaPath =
"sample_images/hirakawa.bmp"); //ヘッダではこのように宣言すること
```

```

void name::t5cl::run(const int num, const std::string & imagePath, const
std::string & hirakawaPath)
{
    const cv::Mat3b lena(cv::imread(imagePath, cv::IMREAD_COLOR));
    CV_Assert(!lena.empty());
    const cv::Mat1b hirakawa(cv::imread(hirakawaPath, cv::IMREAD_GRAYSCALE));
    CV_Assert(!hirakawa.empty());

    switch (num) {
    default:
    case 1:
        std::cout << "run labeling ";
        {
            cv::Mat1i labeled;
            t5cl::labeling(hirakawa, labeled);

            cv::Mat3b colored;
            t5cl::coloring(labeled, colored);
            t3cl::imshow("labeling", colored);
        }
        std::cout << "\r" << "complete labeling" << std::endl;
        if (num != 0) break;
    case 2:
        std::cout << "run shallowConvolution ";
        {
            cv::Mat3b conv;
            // 設計したフィルタ可視化用の出力
            std::vector<cv::Mat1f> outputCreatedFilters_1st,
outputCreatedFilters_2nd;
            t5cl::shallowConvolution(lena, conv, outputCreatedFilters_1st,
outputCreatedFilters_2nd);
            t3cl::imshow("conv", conv);
        }
        std::cout << "\r" << "complete shallowConvolution" << std::endl;
        if (num != 0) break;
    }
}

```

自由課題

- ハフ変換
関数名 : houghTransform
- k平均法
関数名 : k_means