プログラミングゼミ
第２回：C++

日時：2023/04/26

担当：長山

今回のゴール

ビッグデータの数値シミュレーションに必要なC++の基礎を習得する

1. Hello world　　：導入として
2. データの入出力ができる　：コピペでも使えればOK！
3. データの統計量を計算できる：重要．ここに脳みそを使ってほしい

お届け

- テーマ　：Dracula
- フォント：Source Han Code JP
  - インストール方法：下のリンクから.ttcファイルをインストールして実行してください
  - https://github.com/adobe-fonts/source-han-code-jp/releases

C++の位置づけ

C++ （1983~）

- コンパイル言語
- 標準ライブラリ
  STL=Standard Template Library
- 強み： 適当に書いて速い
- 弱み： ファイル操作や図の作成には不向き

Python （1991~）

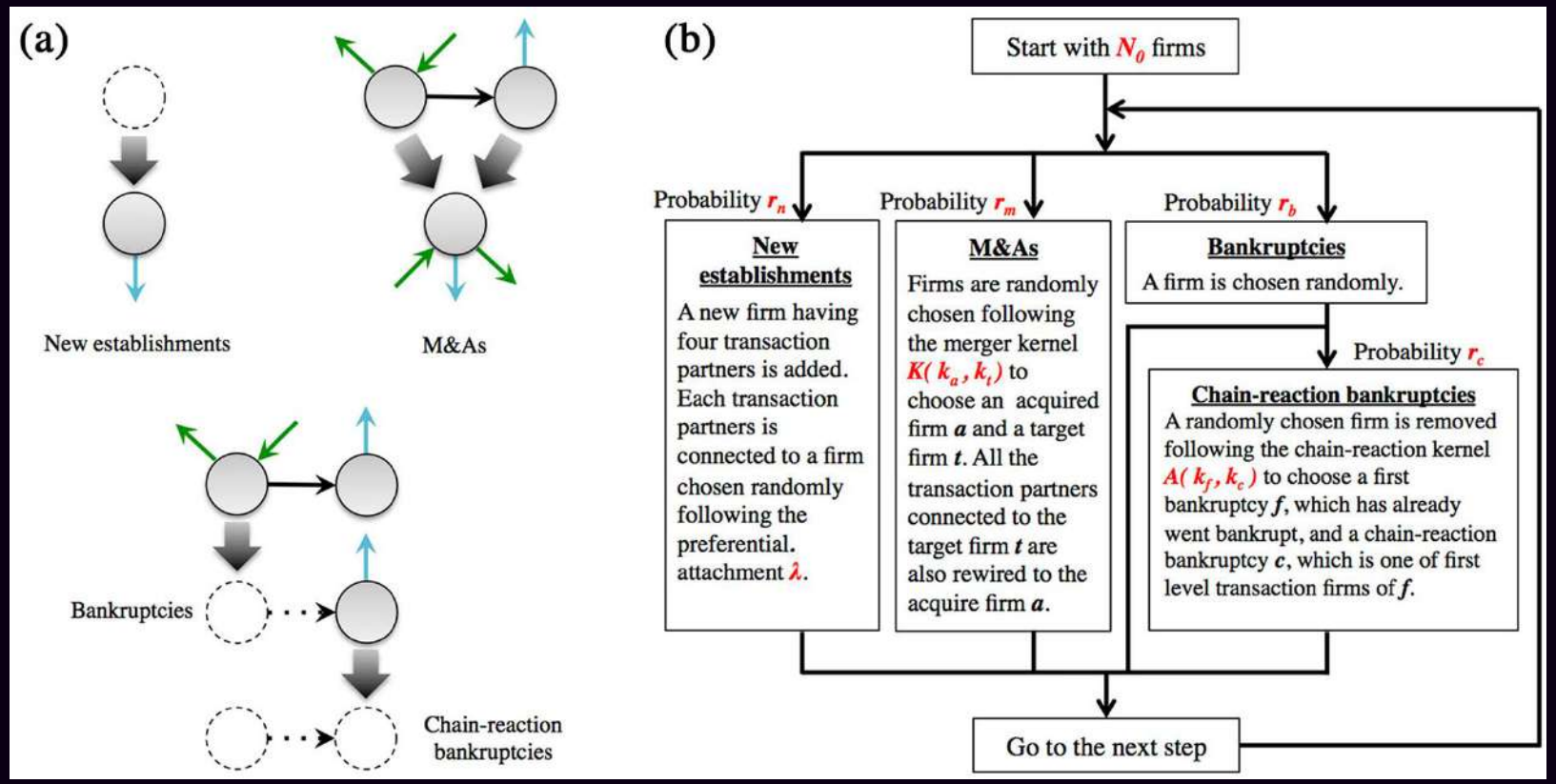
- インタープリタ言語
- ライブラリが豊富
  外部から簡単に追加できる(pip install *)
- 強み： 簡単で多機能，型宣言不要
- 弱み： 遅い

高安研での用途

- Awkなど： データ抽出（C++/Pythonには最低限のデータを読み込ませる）
- C++　　： シミュレーション等の重たい計算
- Python： 基本的な解析と図表作成（多くの場合はこれで済む）

シミュレーションの例

企業の成長過程（新規参入，合併，倒産，連鎖倒産）



$N_0 \sim 10^6$
$T = 12500$

Goto H, Takayasu H, Takayasu M (2017) Estimating risk propagation between interacting firms on inter-firm complex network. PLoS ONE 12(10): e0185712. https://doi.org/10.1371/journal.pone.0185712

C++で覚えること

1. 基本

- C++の使い方: コンパイルと実行方法
- 基本型　　: bool,char,int,double,void
- 関数　　: 定義，呼び出し
- 制御文　　: if,for,~~while,switch~~
- コマンドライン引数: argc,argv

2. STLの使い方

- 文字列　　: 文字列型と数値型の相互変換，split関数
- 入出力　　: 標準入出力，ファイル入出力
- コンテナ　　: vector,map,~~set,queue,stack,pair,tuple~~
- 数値計算　　: cmath,~~random~~
- アルゴリズム: ~~sort,unique,reverse~~

Part 1: C++の基本

Hello world!

プログラム（hello.cpp）

- 拡張子　　：cpp
- main関数　：実行時に呼ばれる
- return 0;：正常終了

```cpp
#include <iostream>

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

ターミナルでコンパイルして実行

- g++　：コンパイラ
- -std ：C++のバージョンを指定
- a.out：実行ファイル

```
knaga@lab fish:3.6.1 ~/prg_seminar/02_CPP/sample
$ g++ -std=c++17 hello.cpp

knaga@lab fish:3.6.1 ~/prg_seminar/02_CPP/sample
$ ./a.out
Hello world!
```

g++の他のオプション

- -o *name*：実行ファイルに名前を付ける
- -O2,-O3：最適化オプション
- -Wall　：警告を出す

脱線）実行ファイルの中身

hexdumpコマンド：
機械語（バイナリ）を読める

Hello world!の周辺を見てみる

grepのオプション

- -B *num*： マッチした行の前も表示
- -A *num*： マッチした行の後も表示

```
knaga@lab fish:3.6.1 ~/prg_seminar/02_CPP/sample
$ hexdump -C a.out | head -n5
00000000  7f 45 4c 46 02 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  03 00 3e 00 01 00 00 00  c0 10 00 00 00 00 00 00  |..>.............|
00000020  40 00 00 00 00 00 00 00  b8 3b 00 00 00 00 00 00  |@........;......|
00000030  00 00 00 00 40 00 38 00  0d 00 40 00 1f 00 1e 00  |....@.8...@.....|
00000040  06 00 00 00 04 00 00 00  40 00 00 00 00 00 00 00  |........@.......|

knaga@lab fish:3.6.1 ~/prg_seminar/02_CPP/sample
$ hexdump -C a.out | grep -B5 -A5 Hello
000012b0  41 5e 41 5f c3 66 66 2e  0f 1f 84 00 00 00 00 00  |A^A_.ff.........|
000012c0  f3 0f 1e fa c3 00 00 00  f3 0f 1e fa 48 83 ec 08  |............H...|
000012d0  48 83 c4 08 c3 00 00 00  00 00 00 00 00 00 00 00  |H...............|
000012e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00002000  01 00 02 00 48 65 6c 6c  6f 20 77 6f 72 6c 64 21  |....Hello world!|
00002010  00 00 00 00 01 1b 03 3b  50 00 00 00 09 00 00 00  |.......;P.......|
00002020  0c f0 ff ff 84 00 00 00  5c f0 ff ff ac 00 00 00  |........\.......|
00002030  6c f0 ff ff c4 00 00 00  ac f0 ff ff 6c 00 00 00  |l...........l...|
00002040  95 f1 ff ff dc 00 00 00  cc f1 ff ff fc 00 00 00  |................|
00002050  19 f2 ff ff 1c 01 00 00  3c f2 ff ff 3c 01 00 00  |........<...<...|
```

基本型

数値型

- int　　：整数を表す．4バイト
- double：倍精度浮動小数点．8バイト

その他の型

- bool　：true/falseの2値をとる
- char　：1バイトのASCII文字を表す
  シングルクオーテーションで囲む
- void　：値を持たない
  関数の戻り値などに使用

```
// int型の使用例
int age = 20;
int new_age = age + 10;

// double型の使用例
double height = 170.5;
double new_height = height + 5.0;

// bool型の使用例
bool is_active = true;
bool has_error = false;

// char型の使用例
char letter = '\n'
```

関数・制御文

関数

- 宣言: 戻り値型 関数名(型 引数名)
- デフォルト値を指定できる

制御文

- if(){} else if(){} else{}
- for(){}
- while(){}
- continue/break

```cpp
// べき乗を計算（ただし，底，指数ともに非負整数）
// 注意：オーバーフロー（桁あふれ）
// * power(10, 9)  = 1000000000
// * power(10, 10) = 1410065408
int power(int base, int exponent)
{
    if (exponent <= 0)
        return 1;
    return base * power(base, exponent - 1);
}

// "="の仕切りを標準出力
// 例: ==========
void print_partition(int length = 30)
{
    for (int i = 0; i < length; i++)
    {
        std::cout << '=';
    }
    std::cout << std::endl;
}
```

コマンドライン引数

main関数に引数を追加する

- argc: CMD引数の数が入る
- argv: CMD引数が格納される

（CMD＝コマンドライン）

コピペして使ってください

```cpp
int main(int argc, char *argv[])
{
    print_partition(20)
    for (int i = 0; i < argc; i++)
    {
        std::string arg = argv[i];
        std::cout << i << ": " << arg << std::endl;
    }
    print_partition(20);
    return 0;
}
```

```
knaga@lab fish:3.6.1 ~/prg_seminar/02_CPP/sample
$ ./a.out awk c++ python
====================
0: ./a.out
1: awk
2: c++
3: python
====================
```

0番目は実行ファイル名

1番目以降に追加の引数

課題

1) ChatGPTに Hello world を書いてもらう
2) 1から20までの整数の階乗を計算して標準出力する
3) 20の階乗の桁数はいくつ？

補足

- ChatGPTを使っていない方は今すぐログイン
- 数値型も標準出力できます

Part 2: STLの使い方

STL(Standard Template Library)

代表例

- 文字列　　：string(文字列型)，数値型の相互変換，split関数
- 入出力　　：iostream(標準入出力)，fstream(ファイル入出力)
- コンテナ　：vector,map(連想配列),~~set,queue,stack,pair,tuple~~
- 数値計算　：cmath,~~random~~
- アルゴリズム：~~sort,unique,reverse~~

使い方

1. includeする
2. std::関数名のように指定

```cpp
#include <iostream>

int main()
{
    // 文字列を標準出力
    std::cout << "Hello!" << std::endl;
    return 0;
}
```

文字列(string)

- ダブルクオーテーションで囲む（charはシングル）
- 連結 +
- 辞書順の比較演算子 <,>,<=,>=
- int型への変換 std::stoi(*)
- double型への変換 std::stod(*)

```
#include <string>

int main(int argc, char *argv[])
{
    std::string msg = "Hello World!";
    int i = std::stoi(argv[1]);
    double d = std::stod(argv[2]);

    return 0;
}
```

- 文字列への変換 std::to_string(*)
- 行の読み込み std::getline(stream, buffer, delim='\n')

標準入出力(iostream)

- 標準入力 std::cin
- 標準出力 std::cout
- 改行してflush std::endl

標準出力には printf も使える
（Cから引き継がれた）

```cpp
#include <iostream>

int main()
{
    double PI = 3.1415;
    std::cout << "PI = " << (int)PI << std::endl;
    std::cout << "PI = " << PI << std::endl;
    printf("PI = %1.0f\n", PI);
    printf("PI = %1.2f\n", PI);

    return 0;
}
```

ファイル入出力(fstream)

- ファイル入力 std::ifstream
- ファイル出力 std::ofstream

- 行の読み込み std::getline()

- コピペして使ってください

```cpp
#include <string>
#include <iostream>
#include <fstream>

int main()
{
    // ファイル入力の例
    std::string input_path = "input.txt";
    std::ifstream ifs(input_path); // stream
    std::string line;              // buffer
    while (std::getline(ifs, line, '\n'))
        std::cout << line << std::endl;

    // ファイル出力の例
    std::string output_path = "output.txt";
    std::ofstream ofs(output_path);
    ofs << "Hello World!" << std::endl;

    return 0;
}
```

課題

UNIXのheadコマンドを実装する

- 仕様1: ファイルの先頭 n 行を標準出力する
- 仕様2: データが n 行ない場合はすべての行を出力する
- 仕様3: コマンドライン引数でファイルのパスと n を指定
- 仕様4: n が渡されなかった場合は n=10 とする
- 実行例: ./head filename.txt 5

補足

❖ 余力のある方は，argc!=3 の場合の処理を加えてみる

コンテナ｜vector

概要

- 可変長配列 std::vector<type>
- 何でも配列にできる（vectorのvector等も定義できる）

メソッド

- 末尾への追加: push_back(*)
- 要素数取得　: size()
- 要素アクセス: operator[n]

```
#include <vector>

std::vector<int> generate_serial_numbers(int n)
{
    std::vector<int> v;
    for (int i = 0; i < n; i++)
    {
        v.push_back(i);
    }
    return v;
}
```

コンテナ｜vector

概要

- 可変長配列 std::vector<type>
- 何でも配列にできる（vectorのvector等も定義できる）

メソッド

- 末尾への追加: push_back(*)
- 要素数取得　: size()
- 要素アクセス: operator[n]

```
#include <iostream>
#include <vector>

void output_vector(std::vector<int> &v)
{
    for (int i = 0; i < v.size(); i++)
    {
        std::cout << v[i] << std::endl;
    }
}
```

コンテナ｜map

概要

- 連想配列 std::map<key_type,value_type>
- Keyは全順序であり，自動でsortされる（全順序でないとkeyに使えない）

メソッド

- 要素作成/アクセス: operator[key]
- 要素アクセス2: イテレータ

```cpp
#include <string>
#include <map>

// As of Apr-12 06:35 UTC
std::map<std::string, double> store_EXR()
{
    std::map<std::string, double> rate;
    rate["JPY/USD"] = 133.77;
    rate["EUR/USD"] = 0.92;
    rate["CNH/USD"] = 6.89;
    return rate;
}
```

コンテナ｜map

イテレータ

- コンテナ内の要素の位置を指す．ポインタのようなもの
- 先頭を指すイテレータ： begin()
- 末尾を指すイテレータ： end()

```cpp
#include <string>
#include <iostream>
#include <map>

void output_map(std::map<std::string, double> &mp)
{
    for (auto p = mp.begin(); p != mp.end(); p++)
    {
        std::cout << p->first << ": " << p->second << std::endl;
    }
}
```

コンテナ｜範囲for文

- 概要　：型推論 `auto` を利用し，コンテナからの要素取り出しを簡略化
- 使い方：`for (auto element : container){ /* do something */ }`
- 嬉しさ：index/iteratorを指定しなくて良い！
- 対応　：vectorはc++11，mapはc++17から範囲for文が追加

```
void count_element(std::vector<std::string> &v)
{
    // 要素数をカウント
    std::map<std::string, int> cnt;
    for (auto x : v) //since c++11
        cnt[x]++;

    // 各要素とその出現回数を標準出力
    for (auto [key, value] : cnt) // since c++17
        std::cout << key << ":\t" << value << std::endl;
}
```

コンテナ｜参照渡し

値の渡し方

- 値渡し　：　コピーされた値を渡す
- 参照渡し：　値が格納されたアドレス(8B)を渡す
  変数の前に　&　をつける
  使い方は値渡しした場合と同じ
- Pythonでのshallow/deep-copyに対応

関数に渡す引数の扱い

- 基本は値渡し
- 大容量のコンテナは参照渡しすべき
- 例：　TDBは企業数100万，取引関係数500万

```cpp
#include <iostream>

void update(int a, int &b)
{
    a += 1; // 値渡し　された変数を更新
    b += 1; // 参照渡しされた変数を更新
}

int main()
{
    int a, b;
    printf("a=%d, b=%d\n", a, b);
    update(a, b);
    printf("a=%d, b=%d\n", a, b);

    return 0;
}
```

文字列/コンテナ｜split関数

- 文字列を指定した区切り文字で分割し，vectorに格納する関数
- std::stringstreamを使う

コピペして使ってください

```cpp
#include <string>
#include <sstream>
#include <vector>

// 1つの文字列を区切り文字で分割し，vectorに格納する
// 例: {"2023", "4", "26"} = split("2023,4,26", ',')
// 例: {"2023", "4", "26"} = split("2023/4/26", '/')
std::vector<std::string> split(std::string str, char delim = ',')
{
    std::vector<std::string> items;
    std::stringstream ss(str);
    std::string item;
    while (std::getline(ss, item, delim))
        if (!item.empty())
            items.push_back(item);
    return items;
}
```

コンテナ｜CSVファイル読み込み

- ifstreamで1行ずつ読み込み，splitしてvectorに追加する
- コピペして使ってください（split関数と併せて）

```cpp
#include <iostream> // 標準入出力
#include <fstream>  // ファイル入出力
#include <sstream>  // 文字列の分解に使う
#include <string>
#include <vector>

// パスで指定したcsvファイルを読み込む
std::vector<std::vector<std::string>> read_csv(std::string path)
{
    std::ifstream ifs(path);
    std::string line;
    std::vector<std::vector<std::string>> data;
    while (std::getline(ifs, line))
        data.push_back(split(line, ','));
    return data;
}
```

数値計算｜cmath

- たいていの関数は入っている
  - abs,sqrt
  - pow,exp,log
  - round,floor,ceil
  - etc.

```
#include <iostream>
#include <cmath>

void print_zeta2()
{
    double zeta2 = std::riemann_zeta(2); // c++17
    printf(“zeta(2) = %.5f\n”, zeta2);   // 1.64493
}
```

課題

データ： 金沢市の3年分の気温
（文字コード変換，時刻0:00を前日24:00に変換済み）

~~1）何年から何年のデータがある？~~
~~2）年ごとにディレクトリを作成~~
~~3）年ごとにファイルを分割し，上で作成したディレクトリに保存する~~
4）年ごとの，気温が20度を超えた時刻の割合
5）年ごとの，気温の平均値/最小値/最大値/標準偏差
6）全期間を対象に，気温の前時刻差分の時系列を計算
7）好きなデータにおいて月ごとの平均値を求める
8）[発展]日ごとに，気温の平均値/最小値/最大値/標準偏差を算出し，その時系列を作る

発展｜テンプレート

- 同じ機能をもつ関数を異なる型ごとに定義するのは面倒
- テンプレート： 型に依存せず処理を共通化．型はコンパイル時に自動で定まる

関数定義のやり方

```
template <typename T>
void output_vector(std::vector<T> &v)
{
    for (auto x : v)
        std::cout << x << std::endl;
}

template <typename T0, typename T1>
void output_map(std::map<T0, T1> &mp)
{
    for (auto [key, value] : mp)
        std::cout << key << ": " << value << std::endl;
}
```

発展｜テンプレート

- 同じ機能をもつ関数を異なる型ごとに定義するのは面倒
- テンプレート： 型に依存せず処理を共通化．型はコンパイル時に自動で定まる

定義した関数の使い方

```
int main()
{
    std::vector<int>    v0 = {0, 1, 2};
    std::vector<double> v1 = {3.14, 3.141, 3.1415};
    std::map<int, int>    mp0 = {{0, 0}, {1, 1}, {2, 2}};
    std::map<int, double> mp1 = {{0, 3.14}, {1, 3.141}, {2, 3.1415}};

    output_vector(v0);
    output_vector(v1);
    output_map(mp0);
    output_map(mp1);

    return 0;
}
```

発展｜クラス（/構造体）

- データの集まり ＋ そのデータについての処理の集まり
- オブジェクト指向プログラミングで用いる
  - 手続きを最初から最後まで羅列するのではなく，オブジェクトに処理を付随させる
  - オブジェクトが直感的な動作をするので，わかりやすい
  - カプセル化： 複雑な処理を隠蔽し，インターフェイスのみ残す
  - チームでプログラミングするときに有用

クラスの定義

```
class Counter
{
private:
    int count = 0;

public:
    void tick() { count++; }
    void reset() { count = 0; }
    int get() { return count; }
    void output() { std::cout << "Count is " << count << std::endl; }
};
```

最後にセミコロンが必要

発展｜クラス（/構造体）

クラスの使い方

```
int main()
{
    Counter counter;
    counter.tick();
    counter.output();
    std::cout << counter.get() << std::endl;
    return 0;
}
```

```
knaga@lab fish:3.6.1 ~/prg_seminar/02_CPP
$ g++ -std=c++17 sample/class.cpp -o bin/class

knaga@lab fish:3.6.1 ~/prg_seminar/02_CPP
$ bin/class
Count is 1
1
```

補足

- 構造体(struct)という型もある
- 基本的には class と同じ
- デフォルトのアクセス指定子が public（クラスは private）

発展｜高速化のためのコツ

そもそも，高速化する必要ある？

- ゴールは研究成果を出すこと
  短縮したいのはアウトプットまでの合計時間
- プログラムの実行が1時間程度で終わるなら待てばよい
  その間に論文調査や資料作成，他の解析などの進捗を生める

プログラムの高速化が必要なら…

- コンテナの受け渡しには参照渡しを使う
- なるべくvectorで書く
  - mapはvectorで書けるかも（N個のkeyに0からN-1の番号を付ける）
  - vector<pair<int,int>>はvector<int>に展開できる
- コンパイラに最適化オプションを渡す