競技プログラミングのための C++入門

@natrium11321

C++ って何？

- C言語の進化系 (の一つ)

C言語
- 配列
- 関数
- ポインタ
- 構造体

C++
- オブジェクト指向
- 参照型
- 例外機構
- 演算子オーバーロード
- 実行時型情報
- テンプレート
- STL
- スレッド (C++11より)
- 型推論
- ラムダ式

C++ って何？

- C言語の進化系 (の一つ)

使えると
競技でめっちゃ
便利！！！

- ポインタ
- 構造体

C++

- オブジェクト指向
- 参照型
- 例外機構
- 演算子オーバーロード
- 実行時型情報
- テンプレート
- **STL**
- スレッド (C++11より)
- 型推論
- ラムダ式

C++ことはじめ

C++を入門します

C++ことはじめ

- C言語の Hello, world!

hello.c

```c
#include <stdio.h>

int main(void)
{
    printf(“Hello, world!\n”);
    return 0;
}
```

C++ことはじめ

- C++の Hello, world!

hello.cpp

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

int main()
{
    printf(“Hello, world!\n”);
    return 0;
}
```

本当はC言語版のコードもそのまま動くが…

CとC++の違い

hello.c

```c
#include <stdio.h>

int main(void)
{
    printf("Hello, world!\n");
    return 0;
}
```

！？

hello.cpp

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

int main()
{
    printf("Hello, world!\n");
    return 0;
}
```

CとC++の違い　①拡張子

- C言語では
  - test.c
  - hoge.c
- C++では
  - test.cpp
  - hoge.cpp

CとC++の違い

hello.c

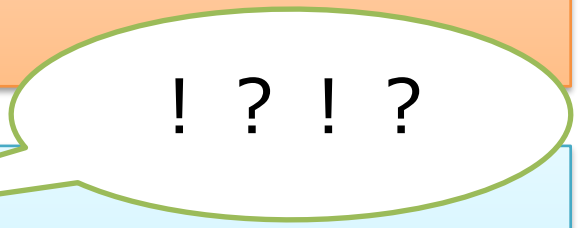
```c
#include <stdio.h>

int main(void)
{
    printf("Hello, world!\n");
    return 0;
}
```

hello.cpp

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

int main()
{
    printf("Hello, world!\n");
    return 0;
}
```



CとC++の違い　②ヘッダ名

- C言語では
  - stdio.h
  - stdlib.h
  - math.h
  - string.h
  - ctype.h
- C++では
  - cstdio
  - cstdlib
  - cmath
  - cstring
  - cctype

cstdio：「C言語の時からあるstdioという名前のヘッダ」
↓
C++で新たに追加されたヘッダには 'c' は付かない

CとC++の違い

hello.c

```c
#include <stdio.h>

int main(void)
{
    printf("Hello, world!\n");
    return 0;
}
```

hello.cpp

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

int main()
{
    printf("Hello, world!\n");
    return 0;
}
```

☆おまじない☆

CとC++の違い

hello.c

```c
#include <stdio.h>

int main(void)
{
    printf("Hello, world!\n");
    return 0;
}
```

hello.cpp

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

int main()
{
    printf("Hello, world!\n");
    return 0;
}
```

！？！？？！！？

CとC++の違い　③引数のvoid

void.c

```c
#include <stdio.h>

void func(void);
int main(void)
{
    func();
    return 0;
}
void func(void)
{
    printf(“fugafuga\n”);
}
```

void.cpp

```cpp
#include <cstdio>
using namespace std;
void func();
int main()
{
    func();
    return 0;
}
void func()
{
    printf(“fugafuga\n”);
}
```

- C++では引数のvoidを省略可
  - 引数のみ

もうひとつ！

One more!

CとC++の違い　④構造体

- C言語の構造体

structure.c

```
struct Human
{
    int age;
    double weight;
    char name[256];
};

int main(void)
{
    struct Human taro;
    taro.age = 21;
    …
}
```

CとC++の違い　④構造体

- C言語の構造体（typedef）

structure.c

```
typedef struct
{
    int age;
    double weight;
    char name[256];
} Human;

int main(void)
{
    Human taro;
    taro.age = 21;
    …
}
```

CとC++の違い　④構造体

- C++の構造体

structure.cpp

```
struct Human
{
    int age;
    double weight;
    char name[256];
};

int main()
{
    Human taro;
    taro.age = 21;
    …
}
```

C++新機能紹介

boolと変数の宣言位置

C++新機能　①bool

eratos.c

```
/*
 * n以下のxについて isprime[x] == 1 だったらxは素数
 * という配列isprimeを作る
 */
void make_sieve(int n, int isprime[])
{
    int i, j;
    isprime[0] = isprime[1] = 0;
    for(i=2; i<=n; ++i)
        isprime[i] = 1;

    for(i=2; i<=n; ++i){
        if(isprime[i] == 1){
            for(j=i*2; j<=n; j+=i)
                isprime[j] = 0;
        }
    }
}
```

C++新機能　①bool

(^o^)

2,147,483,647まで格納できるintに
0か1しか格納しないのは勿体無いな…

(˘⊖˘) 。○ ○

待てよ、書いてたコード
C++だっけ…

(◠‿◠)☛

そこに気づいてしまったか…
貴様にはbool型を使ってもらおう

(’ω’)

うわあああああああ

C++新機能　①bool

- bool型変数(1bit整数)

```
bool flag;
```

- 値は true(真) か false(偽) を代入

```
flag = true;    // 数値としては1
flag = false;   // 数値としては0
```

- if で使う

```
if(flag){
    puts(“flag is true”);
}
else{
    puts(“flag is false”);
}
```

C++新機能　①bool

- 比較の結果を格納

```
int a = 3, b = 5;
flag = a < b;   // true
flag = a != b;  // true
flag = a >= b;  // false
```

- 否定演算

```
bool flag1 = true;
bool flag2 = !flag1; // false
bool flag3 = !flag2; // true
if(!flag){
    puts(“flag is false”);
}
else{
    puts(“flag is true”);
}
```

C++新機能　①bool

eratos.cpp

```
// n以下のxについて isprime[x] がtrueだったらxは素数
// という配列isprimeを作る
void make_sieve(int n, bool isprime[])
{
    int i, j;
    isprime[0] = isprime[1] = false;
    for(i=2; i<=n; ++i)
        isprime[i] = true;

    for(i=2; i<=n; ++i){
        if(isprime[i]){
            for(j=i*2; j<=n; j+=i)
                isprime[j] = false;
        }
    }
}
```

C++新機能　①bool

- bool を使うメリット
  - `int flag;` ➡ `bool flag;`
    - ✓「これはフラグを表す変数です」というのが分かりやすい
  - `flag = 1;` ➡ `flag = true;`
    - ✓ 0や1といった怪しい数字で場合分けしなくて良い
  - `int` : 4バイト ➡ `bool` : 1バイト
    - ✓ メモリを節約できる

C++新機能　②変数の宣言位置

gucha.c

```
/* プログラム速いけど微妙にバグる */
int main(void)
{
    int i, j, k, n, a[1024];
    int map[1024][1024];
    int from, to, end, *p, *q;
    char c, s, t, str[128];
    …
}
```

＼(^o^)／

どの変数をどこで使ってるかが分からん！

C++新機能　②変数の宣言位置

smart.cpp

```
int main()  {
    int n;
    scanf("%d", &n);

    int a[1024];
    for(int i=0; i<n; ++i)
        scanf("%d", &a[i]);

    int map[1024][1024];
    for(int i=0; i<n; ++i){
        for(int j=0; j<n; ++j){
            map[i][j] = 0;
            for(int k=0; k<n; ++k)
                map[i][j] += a[i] * a[(i+k)%n];
        }
    }
    …
}
```

ここだけ
じゃなく

ここでも

ここでも

ここでも

この辺でも

C++新機能　②変数の宣言位置

- どこでも変数宣言が出来る
  - 特にforの中でのループ変数の宣言は便利

- どこでも変数宣言が出来るメリット
  - ✓ 変数を使う直前に宣言できる
    - ➡ **どこで変数を使ってるかが分かりやすい！**
    - ➡ **宣言部分がごちゃごちゃしない！**
    - ➡ **初期化を忘れない！**

C++新機能　②変数の宣言位置

eratos_smart.cpp

```
// n以下のxについて isprime[x] がtrueだったらxは素数
// という配列isprimeを作る
void make_sieve(int n, bool isprime[])
{
    isprime[0] = isprime[1] = false;
    for(int i=2; i<=n; ++i)
        isprime[i] = true;

    for(int i=2; i<=n; ++i){
        if(isprime[i]){
            for(int j=i*2; j<=n; j+=i)
                isprime[j] = false;
        }
    }
}
```

STL (Standard Template Library)

algorithm, stack, queue, string

STLって何？

- C++の標準ライブラリ（の一部）
- とても便利な機能が沢山詰まってます
  - 特に競技プログラミングでは重宝
- 本日紹介するのは
  - `<algorithm>`
  - `<stack>`
  - `<queue>`
  - `<string>`

algorithm

- 便利な関数の詰め合わせセット

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

- 基本的にSTLの関数は引数の型について不問
  - 「テンプレート」で実現
  - 今日は詳しくやりません

最大値・最小値

- min(a, b)
  - aとbの小さい方の値を返す
- max(a, b)
  - aとbの大きい方の値を返す

```
int a = 10, b = 4;

int small = min(a, b);          // 4
int large = max(a, b);          // 10

double c = 1.2, d = 5.8;
double e = max(c*c, d*d);       // int 以外もOK

double f = min(a, c);           // 違う型を渡すとエラー
double g = min((double)a, c); // キャストすればOK
```

交換

- swap(a, b)
  - aとb の値を交換する

```
int a = 10, b = 4;
printf(“a is %d, b is %d\n”, a, b); // a is 10, b is 4

swap(a, b);
printf(“a is %d, b is %d\n”, a, b); // a is 4, b is 10

int* p = &a;
int* q = &b;
swap(p, q);
*q += 3;
printf(“a is %d, b is %d\n”, a, b); // a is 7, b is 10
```

ソーティング

- sort(f, t)
  - f から t-1 までを昇順にソートする
  - f と t は配列のポインタ
  - 時間計算量 O(nlogn)

```
int a[] = {5, 3, 7, 8, 2};
sort(a, a+5);
for(int i=0; i<5; ++i)
    printf("%d ", a[i]);

/*
 * 出力結果
 * 2 3 5 7 8
 */
```

ソーティング

- reverse(f, t)
  - f から t-1 までを反転させる
  - 降順にソートしたければこれを使う
  - 時間計算量 O(n)

```
int a[] = {5, 3, 7, 8, 2};
reverse(a, a+5); // a = {2, 8, 7, 3, 5}

sort(a, a+5);    // a = {2, 3, 5, 7, 8}
reverse(a, a+5); // a = {8, 7, 5, 3, 2}
```

数え上げ

- count(f, t, x)
  - f から t-1 までに含まれるxの数を返す

```
int a[] = {1, 4, 8, 2, 8, 6, 8, 1};
int cnt8 = count(a, a+8, 8); // 3
int cnt4 = count(a, a+8, 4); // 1
```

  - この関数のせいで“count”という名前の変数を作るとエラーが起こる
    - “cnt”とかで代用

初期化

- fill(f, t, x)
  - f から t-1 までを全て x で初期化する

```
int a[] = {1, 4, 8, 2, 8, 6, 8, 1};
fill(a, a+8, 1);   // a = {1, 1, 1, 1, 1, 1, 1, 1}
fill(a+2, a+5, 8); // a = {1, 1, 8, 8, 8, 1, 1, 1}

bool flags[1000];
fill(flags, flags+1000, true);
```

algorithm

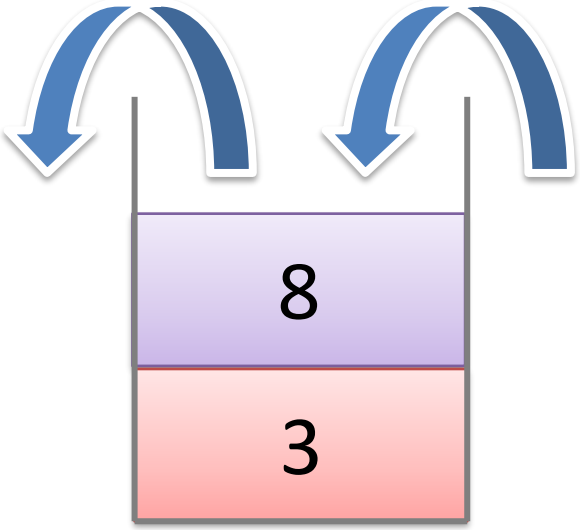
- ほかにも便利な関数いろいろ
  - `equal, find, search, copy, remove, replace,`

    `random_shuffle, unique, lower_bound, upper_bound, binary_search, max_element, min_element, next_permutation, etc…`

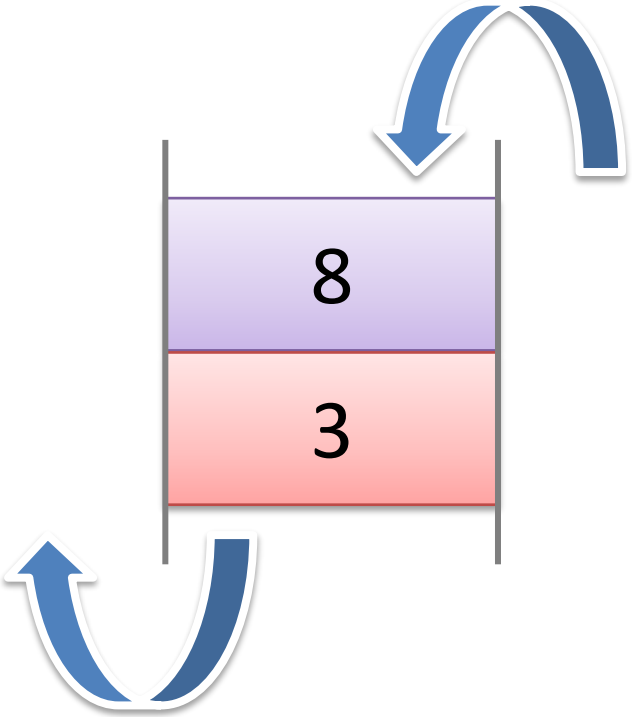
これらを知らないと猛烈に不利ということはない！
（ループ回せば普通にできる、JOI予選程度では使わない、等）

stack/queue

- スタックとキュー



stack
(先入れ後出し)



queue
(先入れ先出し)

stack/queue

- スタックとキュー

```
#include <stack>
#include <queue>
using namespace std;
```

そのまえに

Before it

クラスって何？

- C言語の構造体に毛が生えたもの
- 変数だけではなく、関数も中に入れられる
  - C++では構造体の中にも関数作れるけどね
- クラスの中の変数：**メンバ変数**
- クラスの中の関数：**メンバ関数**
- 使うだけなら簡単です

stack

- スタックの宣言

```
stack<int> S;
```

stack<int> というクラス(型)の、
Sという名前の変数(インスタンス)を作成

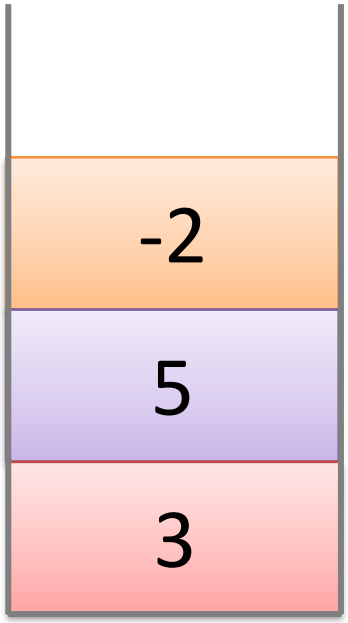
int はスタックに格納する
変数の型を表す

stack

- スタックへの追加

```
stack<int> S;
S.push(3);
S.push(5);
S.push(-2);
```

S.push(x)
Sにxを追加



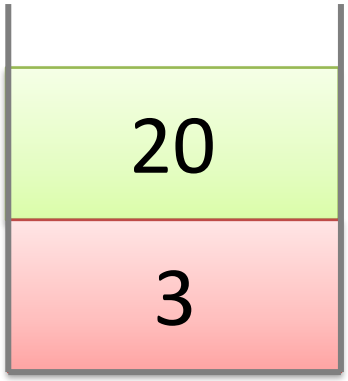
stack

- スタックの先頭要素の取り出し

```
stack<int> S;
S.push(3);
S.push(5);
S.push(-2);
printf(“%d\n”, S.top());
S.pop();
printf(“%d\n”, S.top());
S.pop();
S.push(20);
```

S.top()
現在のSの先頭要素を返す
（削除はしない）

S.pop()
現在のSの先頭要素を削除



stack

- スタックのサイズ確認

```
stack<int> S;
S.push(3);
S.push(5);
S.push(-2);

int cnt = S.size();
bool isempty = S.empty();


// スタックの全ての要素を取り出して表示
while(!S.empty()){
    int data = S.top();
    S.pop();
    printf("%d\n", data);
}
```

S.size()
Sに含まれる要素数を返す

S.empty()
Sが空かどうかを返す

queue

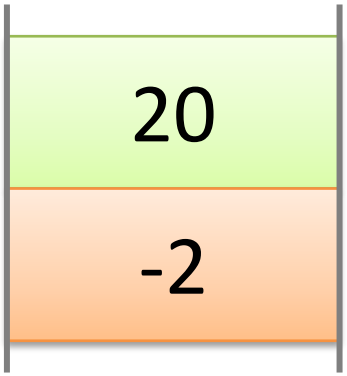
- キューの宣言・追加・先頭要素の取り出し

```
queue<int> Q;
Q.push(3);
Q.push(5);
Q.push(-2);
printf(“%d\n”, Q.front());
Q.pop();
printf(“%d\n”, Q.front());
Q.pop();
Q.push(20);
```

Q.push(x)
Qにxを追加

Q.front()
現在のQの先頭要素を返す
（削除はしない）

Q.pop()
現在のQの先頭要素を削除



queue

- キューのサイズ確認

```
queue<int> Q;
Q.push(3);
Q.push(5);
Q.push(-2);

int cnt = Q.size();
bool isempty = Q.empty();


// キューの全ての要素を取り出して表示
while(!Q.empty()){
    int data = Q.front();
    Q.pop();
    printf("%d\n", data);
}
```

Q.size()
Qに含まれる要素数を返す

Q.empty()
Qが空かどうかを返す

stack/queue 応用編

- キューの要素に構造体

```
struct Human {
    int age;
    double height;
    char name[256];
};
int main()
{
    queue<Human> Q;
    Human taro = {21, 168.5};
    strcpy(taro.name, “Taro Tanaka”);
    Q.push(taro);
    printf(“%d\n”, Q.front().age);
    return 0;
}
```

stack/queue 応用編：幅優先探索

```
struct Data {
    int y, x, c;
};
int main() {
    int h, w, field[1000][1000], sy, sx, gy, gx, ans = -1;
    // 入力部省略
    bool visited[1000][1000] = {{false}};
    queue<Data> Q;
    Data start = {sy, sx, 0};
    Q.push(start);
    while(!Q.empty()){
        Data data = Q.front();
        Q.pop();
        if(data.y == gy && data.x == gx){
            ans = data.c;
            break;
        }
        if(visited[data.y][data.x]) continue;
        visited[data.y][data.x] = true;
        for(int i=0; i<4; ++i){
            int dy[] = {-1, 0, 0, 1}, dx[] = {0, 1, -1, 0};
            int py = data.y + dy[i], px = data.x + dx[i];
                  if(py<0 || h<=py || px<0 || w<=px || field[py][px]==1)
                continue;
            Data next = {py, px, data.c+1};
            Q.push(next);
        }
    }
    printf("%d\n", ans);
    return 0;
}
```

string

- 文字列クラス

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

  - インクルードするファイル名は “string”
  - “string.h” はC言語のヘッダ
  - “cstring” はC++での string.h

- C言語では　char型の配列で表現
- C++では　char型の配列やstring型で表現

文字列操作　①入出力・基本操作

- C言語では

```
char str[256];
int length;

/* 入力(空白・改行区切り) */
scanf(“%s”, str);

/* 長さ取得 */
length = strlen(str);

/* 先頭文字の変更 */
str[0] = ‘a’;

/* 出力 */
printf(“%s”, str);
```

文字列操作　①入出力・基本操作

- C++では

```
string str;

// 入力(空白・改行区切り)
char tmp[256];
scanf(“%s”, tmp);  // 直接stringにscanf出来ない
str = tmp;

// 長さ取得
int length = str.size();

// 先頭文字の変更
str[0] = ‘a’;

// 出力
printf(“%s”, str.c_str());
```

文字列操作　②コピー・連結

- C言語では

```
char str1[256], str2[256], str3[256];

/* str2を ”abcde” で初期化 */
strcpy(str2, “abcde”);

/* str1にstr2をコピー */
strcpy(str1, str2);

/* str1の末尾にstr2を連結 */
strcat(str1, str2);

/* str1の末尾にstr2を連結したものをstr3に代入 */
strcpy(str3, str1);
strcat(str3, str2);
```

文字列操作　②コピー・連結

- C++では

```
string str1, str2, str3;

/* str2を ”abcde” で初期化 */
str2 = “abcde”;

/* str1にstr2をコピー */
str1 = str2;

/* str1の末尾にstr2を連結 */
str1 += str2;

/* str1の末尾にstr2を連結したものをstr3に代入 */
str3 = str1 + str2;
```

文字列操作　③辞書順比較

- C言語では

```c
char str1[256], str2[256];
int res;
/* 入力省略 */
/* str1とstr2について、以下のように場合分けする
 *   ・A: str1がstr2よりも辞書順で早い場合
 *   ・B: str1とstr2が等しい場合
 *   ・C: str1がstr2よりも辞書順で遅い場合
 */
res = strcmp(str1, str2);
if(res < 0){
    /* (A) */
}else if(res == 0){
    /* (B) */
}else{
    /* (C) */
}
```

文字列操作　③辞書順比較

- C++では

```
string str1, str2;

// 入力省略
/*
 * str1とstr2について、以下のように場合分けする
 *   ・A: str1がstr2よりも辞書順で早い場合
 *   ・B: str1とstr2が等しい場合
 *   ・C: str1がstr2よりも辞書順で遅い場合
 */
if(str1 < str2){
    // (A)
}else if(str1 == str2){
    // (B)
}else{
    // (C)
}
```

文字列操作　④部分文字列の抽出

- C言語では

```
char str1[256], str2[256], str3[256];
strcpy(str1, “abcde”);

/* str2 にstr1の3文字目から最後までの部分文字列を代入 */
strcpy(str2, str1+2); /* str2 = “cde” */

/* str3 にstr1の2文字目から3文字分の部分文字列を代入 */
strncpy(str3, str1+1, 3);
str3[3] = ‘\0’;        /* str3 = “bcd” */
```

文字列操作　④部分文字列の抽出

- C++では

```
string str1, str2, str3;
str1 = “abcde”;

// str2 にstr1の3文字目から最後までの部分文字列を代入
str2 = str1.substr(2);     // str2 = “cde”

// str3 にstr1の2文字目から3文字分の部分文字列を代入
str3 = str1.substr(1, 3); // str3 = “bcd”
```

文字列操作　⑤応用編

- 文字列の配列をソート

```
string str[100];
for(int i=0; i<n; ++i){
    char tmp[256];
    scanf(“%s”, tmp);
    str[i] = tmp;
}
sort(str, str+n);

/*
 * Sample Input
 *  abcde eifj zzz abc def eifa
 *
 * Sample Output
 *  abc abcde def eifa eifj zzz
 */
```

More Effective STL

- STLにはもっと色々な強力なデータ構造がある
  - vector
  - list
  - deque
  - set / multiset
  - map / multimap
  - priority_queue
  - bitset

今日のまとめ

- CとC++の違い
  - stdio.h → cstdio
  - using namespace std; を忘れない
- C++新機能
  - フラグに使えるbool型
  - どこでも変数宣言可能
- STL
  - algorithm
  - stack / queue
  - string

練習問題

- stack
  - AOJ 0013
    - Switching Railroad Cars
    - PCK2003 本選16問目
- queue
  - AOJ 0558
    - Cheese
    - JOI10-11 予選5問目
  - AOJ 0193
    - Deven-Eleven
    - PCK2008 本選11問目
- queueとstring (とmap)
  - AOJ 0179
    - Mysterious Worm
    - PCK2008 予選7問目