代码黑魔法

郭晓旭

2014 年 3 月 8 日

C++/C++11

Scope

```
int result = 0;
{
    int result = 1;
    std::cout << result << std::endl;
    // => 1
}
std::cout << result << std::endl;
// => 0
```

A better scope

```
do {
    if (failed) {
        break;
    }
    puts("Yes");
} while (false);
```

Placement new

```
// slow
new Node()

// fast
Node nodes[MAX_NODE];

Node* newNode = nodes;
new (newNode++) Node()
```

Stream design

```
struct Info {
    int value;

    Info* update(int delta) {
        value += delta;
        return this;
    }
};

(new Info())->update(1)->update(2)->update(3);
```

std::lower_bound

```
#define ALL(v) (v).begin(), (v).end()

std::vector <int> list;
std::vector <int> values(list);
std::sort(ALL(values));
std::unique(ALL(values), values.end());
for (int &item : list) {
    item = std::lower_bound(ALL(values), item) - values.beg
}
```

```
std::set <int> set;

// bad
std::lower_bound(set.begin(), set.end(), value)
// good
set.lower_bound(value)
```

auto

```
std::vector <int> list;
auto iterator =
        std::lower_bound(list.begin(), list.end(), item);
```

```
std::vector <int> list;
#define FOR(i, v) \
    for (__typeof((v).begin()) i = (v).begin(); \
    i != (v).end(); ++ i)
FOR (item, list) {
    std::cout << *item << std::endl;
}
for (auto &item : list) {
    std::cout << item << std::endl;
}
```

std::unordered_map

告别 std::__gnu_cxx::hash_map

```
std::unordered_map <std::string, int> map;
map["acm"] = 0;
map["icpc"] = 1;
for (auto &item : list) {
    std::cout << item.first << " " <<
                 item.second << std::endl;
}
```

std::function

```
template <typename T>
std::vector <T> map(std::vector <T> list,
                    std::function <T(T)> func) {
    std::vector <T> result;
    for (auto &item : list) {
        result.push_back(func(item));
    }
    return result;
}
```

Lambda

```
std::function <int(int, int)> add =
        [](int a, int b) -> int {
            return a + b;
        };
```

An even better scope

```
double result;
{
    double tmp = x + 1;
    result = tmp * tmp;
}
```

An even better scope

```
double result;
{
    double tmp = x + 1;
    result = tmp * tmp;
}

double result = [&](){
    double tmp = x + 1;
    return tmp * tmp;
}();
```

右值引用

```
void dfs(int u, int &count) {
    position[u] = count ++;
    for (auto v : children[u]) {
        dfs(v, count);
    }
}

int main() {
    int count = 0;
    dfs(0, count);
}
```

```
void dfs(int u, int &&count) {
    position[u] = count ++;
    for (auto v : children[u]) {
        dfs(v, std::move(count));
    }
}

int main() {
    dfs(0, 0);
}
```

Style

Spacing

```
int main(int argc, char* argv[]) {
    int a, b;
    scanf("%d%d", &a, &b);
    if (a < b) {
        printf("%d\n", a);
    } else {
        printf("%d\n", b);
    }
    return 0;
}
```

“我要更多的花括号”

```c
int main(int argc, char* argv[]) {
    int a, b;
    scanf("%d%d", &a, &b);
    printf("%d\n", (a < b ? a : b));
    return 0;
}
```

CamelCase vs snake_case

常量、宏	MAX_COUNT, MAGIC_NUMBER, FOREACH
类型	Pair, AvlTree
变量	pair, avlTree, makeLifeBetter

常量	`MAX_COUNT`, `MAGIC_NUMBER`, `FOREACH`
类型	`Pair`, `AvlTree`
变量、函数	`pair`, `avl_tree`, `make_life_better`

选择并坚持

Don’t Repeat Yourself (DRY)

避免幻数

```
(a *= 2) %= 1000000007;
(a += 1) %= 1000000007;
```

避免幻数

```
(a *= 2) %= 1000000007;
(a += 1) %= 1000000007;

const int MOD = 1000000007;

(a *= 2) %= MOD;
(a += 1) %= MOD;
```

不推荐使用 `#define` 定义常量

逻辑

```
const int N = 100000;

int array[N], operations[N];
```

逻辑

```
const int N = 100000;

int array[N], operations[N];
```

```
const int N = 100000;
const int Q = 100000;

int array[N];
int operations[Q];
```

```
const int V = 100;
const int E = V * (V - 1) / 2;

int vertices[V], edges[E];
```

DRY

```
maximum[x + 1][y] =
    std::max(maximum[x + 1][y],
             maximum[x][y]);
maximum[x + 1][y + 1] =
    std::max(maximum[x + 1][y + 1],
             maximum[x][y]);
```

DRY

```
maximum[x + 1][y] =
    std::max(maximum[x + 1][y],
             maximum[x][y]);
maximum[x + 1][y + 1] =
    std::max(maximum[x + 1][y + 1],
             maximum[x][y]);

void update(int &x, int a) {
    x = std::max(x, a);
}

{
    int &tmp = maximum[x][y];
    update(maximum[x + 1][y], tmp);
    update(maximum[x + 1][y + 1], tmp);
}
```

vs Copy & Paste

```
prefix = partial(array);
suffix = reverse(partial(reverse(array)));

for (int i = 1; i <= n; ++ i) {
    prefix[i] = prefix[i - 1] + array[i];
}
for (int i = n; i >= 1; -- i) {
    suffix[i] = suffix[i + 1] + array[i];
}
```

鲁棒性

```c
while (scanf("%d", &n), n) {
}

while (scanf("%d", &n) != EOF && n) {
}

while (scanf("%d", &n) == 1 && n) {
}
```

Assertion

检验正确性

Polya 定理：群作用下等价类数量是不动点数量的平均值

```
int sum = 0;
for (int i = 0; i < n; ++ i) {
    sum += count_fix_points(i);
}
assert(sum % n == 0);
printf("%d\n", sum / n);
```

对称性

统计 $(0,0)-(a,0)-(0,b)$ 内的格点数量

```
int solve(int a, int b) {
    int result = 0;
    for (int i = 1; i < a; ++ i) {
        result += b * i / a;
    }
    return result;
}

assert(solve(a, b) == solve(b, a));
```

获得对应返回

```
void runtime_error() {
    printf("%d\n", 1 / 0);
}

void time_limit_exceed() {
    int result = 1;
    while (true) {
        result <<= 1;
    }
    printf("%d\n", result);
}
```