HCPC　勉強会
～ C++における実装テクニック ～

D2　鈴木　浩史

概要

- C++によるコーディングで便利なテクニックをいくつか紹介
- コーディングの速さ・正確さに主眼を置きます

- 完全に初心者向け
- 中級者以降は知っている内容だと思っています
- 一部批判する場面がありますのでご容赦ください

- LibreOfficeで作ったので見づらいかも

もくじ

1.よく使う構文を簡潔に

マクロを使おう

- C++のマクロ
  - #define [定義] [意味]

- 定義に従って書くと
- 意味の通りに書いたことと等価になる

使用例

```
#define     for_(i,a,b)     for(int i=(a);i<(b);++i)
#define     rfor_(i,a,b)    for(int i=(a);i>=(b);--i)
#define     allof(a)        (a).begin(),(a).end()
#define     minit(a,b)      memset(a,b,sizeof(a))
```

for文　for_(i,0,n) { … }
イテレータへのアクセス　sort(allof(ary));
配列初期化　minit(dp,0);
など, よく使う構文を簡潔に間違いにくくする

ほどほどに使って欲しい

- マクロマンにありがちな記述
  - #define pb push_back
  - #define F first
  - #define S second

- 可読性を損なうので、やり過ぎだと思う（炎上）

2.型名を簡潔に

エイリアスを使おう

- C++のエイリアス
  - typedef [型名] [エイリアス];
  - using [エイリアス] = [型名];
  - template< typename T > [エイリアス] = [コンテナ名]< T >;

- 形名が長くて嫌な時に

使用例

```
typedef long long lint;
using   pii = pair< int, int >;
template< typename T > using Vec = vector< T >;
template< typename KEY, typename VAL > using UMap = unordered_map< KEY, VAL >;
```

- long long とか pair< int, int > とかはいかにも面倒
- template を使うことでSTLコンテナにも対応
  - 上の例だと、UMap< char, int > は unordered_map< char, int > と等価になる

ほどほどに使って欲しい

- エイリアスマンにありがちな記述
  - typedef double D;
  - typedef pair< int, int > P;
  - typedef vector< int > vi;
  - typedef vector< vector< int > > vii;

- ヤバい以外の感情が無い（炎上）
- 特に、変数名とコンフリクトしそうで怖い
- typedef pair< int, int > pii;も十分破壊力が高いが、まだ読めるし需要があるため目をつぶる

3.作っておくと便利な定数や関数

定数･関数とその意義

- 定数の例
  - const int UB = 10001;
- 関数の例
  - int sqr(int x) { return x * x; }

- 何度も使う場面がある値や処理を、使い回しできるように定義しておく

使用例

```
const int INF = 1L << 30;
const double EPS = 1e-9;

const int dx[4] = {0,1,0,-1};
const int dy[4] = {-1,0,1,0};

bool in_range(int x, int lb, int ub) { return lb <= x && x < ub; }
bool in_range(int x, int y, int W, int H) { return in_range(x,0,W) && in_range(y,0,H); }

template< typename T > void maxUpdate(T& a, T b) { a = max(a, b); }
void modAdd(int& a, int b, int mod) { a = (a + b) % mod; }
```

- 無限大を十分大きな定数INFとしておく
- 許容する浮動小数点誤差をEPSとする
- グリッド上の4方向の移動
- 値が区間内にあるか否かのチェック
- 頻繁に書きうる更新式

4.実装する必要の無い処理を見抜く

例題

- 長さ N の整数列 s = (s_1, …, s_N) が与えられます。Q 回の各クエリで次の処理をしてください。
  - クエリで入力された整数 x について、s が x を含むか否か判定し、含むならば x の個数を、含まないならば -1 を出力してください。

- O(QN)でTLEするくらいな制約

解法

- 基本的な二分探索の問題
  - まず整数列をソート
  - 各クエリに対して二分探索で x の存在範囲を調べる

着眼点

- 実装する内容
  - ソート
  - 二分探索
- これらは、自分で実装する必要があるのか？
  - ない

解答コード例

```
void sortANDserach() {
    int N;
    cin >> N;

    Vec< int > ary(N);
    for_(i,0,N) cin >> ary[i];

    sort(allof(ary));

    int Q;
    cin >> Q;
    for_(q,0,Q) {
        int x;
        cin >> x;

        if (binary_search(allof(ary), x)) {
            int s = lower_bound(allof(ary), x) - ary.begin();
            int t = upper_bound(allof(ary), x) - ary.begin();
            cout << t - s << endl;
        } else {
            puts("-1");
        }
    }
}
```

- STLという便利なものがあるので活用しよう
- ソートや簡単な二分探索は存在している

説明のためにこんな使い方してるけど、ぶっちゃけ無駄

5.複雑な情報を楽に管理

例題

- V個の頂点とE本の有向辺からなる、グラフが与えられます。各辺 i は始点 u_i から終点 v_i の向きにたどることができ、たどるためにはコスト c_i がかかります。多重辺はありません。
- 辺をたどるコストが小さい順に、始点および終点とそのコストを出力してください。距離が同じ場合は、始点の番号が小さい順に、始点も同じ場合は終点の番号が小さい順に出力してください。
- Eはそこそこでかいくらいの制約

解法

- 辺を適切にソートする
- だけなんだけど、素早くコーディングできますか？

解答コード例

```
void graph() {
    int V, E;
    cin >> V >> E;

    struct Edge {
        int from, to, cost;

        void dump(ostream& os) const { os << from << "->" << to << " : " << cost << endl; }

        bool operator < (const Edge& e) const {
            if (cost == e.cost) {
                if (from == e.from) return to < e.to;
                return from < e.from;
            }
            return cost < e.cost;
        }
    };

    Vec< Edge > edges;
    for_(i,0,E) {
        int u, v, c;
        cin >> u >> v >> c;
        edges.push_back(Edge{u,v,c});
    }

    sort(allof(edges));
    for(const Edge& e : edges) e.dump(cout);
}
```

解答コード例

```
void graph() {
    int V, E;
    cin >> V >> E;

    struct Edge {
        int from, to, cost;

        void dump(ostream& os) const { os << from << "->" << to << " : " << cost << endl; }

        bool operator < (const Edge& e) const {
            if (cost == e.cost) {
                if (from == e.from) return to < e.to;
                return from < e.from;
            }
            return cost < e.cost;
        }
    };

    Vec< Edge > edges;
    for_(i,0,E) {
        int u, v, c;
        cin >> u >> v >> c;
        edges.push_back(Edge{u,v,c});
    }

    sort(allof(edges));
    for(const Edge& e : edges) e.dump(cout);
}
```

!?

構造体

- 文法（クラス (class) との差はほぼ無い）
  - struct [名前] { 変数 … 関数 … 演算子 };
  - 詳しい使い方は調べてみてください
- 変数の管理とそれに関わる処理をまとめて担当

```
struct Edge {
    int from, to, cost;

    void dump(ostream& os) const { os << from << "->" << to << " : " << cost << endl; }

    bool operator < (const Edge& e) const {
        if (cost == e.cost) {
            if (from == e.from) return to < e.to;
            return from < e.from;
        }
        return cost < e.cost;
    }
};
```

Edge という名前で新しい構造体を定義

from, to, cost という名前でそれぞれ int 型変数を保持

出力のための関数を定義

from, to, cost という変数郡について
比較演算子 < のルールを定義

おしまい

割と詳細を省いているけど、調べる力も大事だよ！

キーワード：マクロ、エイリアス、STL、構造体、クラス