Implementeerimine Võistlusprogrameerimisel

Oliver-Matis Lill

March 11, 2017

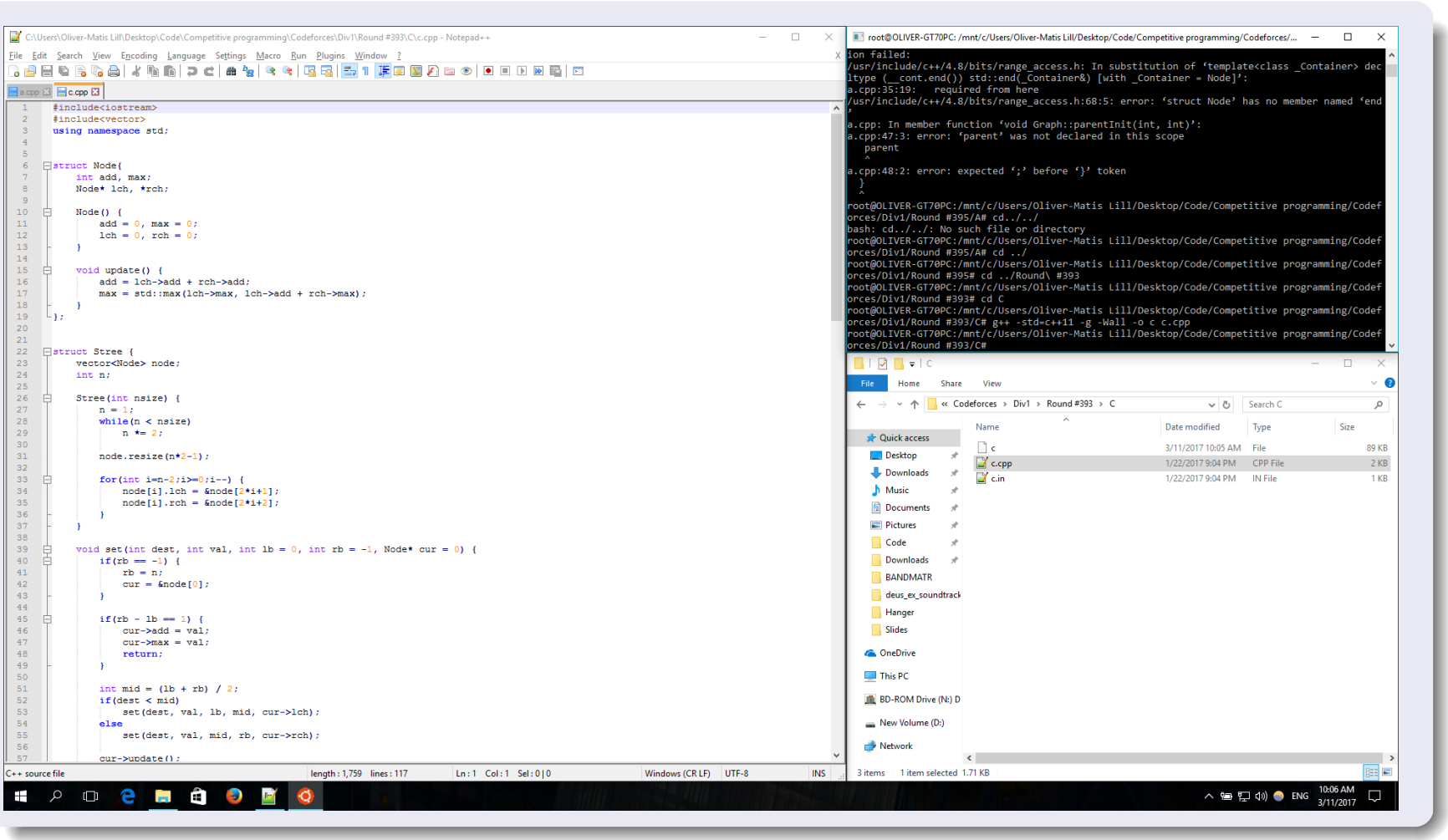
Sessiooni Struktuur

- Minu blokk on implementeerimise harjutamiseks, järgnevad blokid käsitlevad testimist
- Alguses toimub loeng, ülejäänud aeg on minivõistluse lahendamiseks
- Tehke palju vigu, et oleks järgmistes blokkides mida testida

Teemad

- Töökeskkond
- Standardteek
- Implementeerimisnipid

Minu programmeerimiskeskkond



Minu tööriistad

- Bash on Ubuntu on Windows - Linux-i käsurida windowsil, mugavam kui virtuaalmasin
- Notepad++ - Korralik textiredaktor windows-ile
- Windows Explorer - Aadressiriba saab kasutada et samas kaustas bash-i konsool käivitada
- g++ - Kompileerib C++ failid
- gdb - Debug-imise tööriist

Standardteek

- Enamus keeltel on oma standardteek (Standard library)
- Sisaldab suur hunnik funktsioone, objekte jms asju mis võimaldavad palju korda saata
- Kättesaadav praktiliselt igas keskkonnas (kaasaarvatud programmeerimisvõistlustel)
- Võistlustel väga kasulik

Standardteegi elemendid

Mõned kasulikud c++ standardteegi funktsioonid/klassid:

1. vector - muudetava suurusega jada
2. sort - sorteerimisfunktsioon, saab oma võrdlusfunktsiooni järjestamiseks kasutada
3. lower_bound\upper_bound - kahendotsing
4. priority_queue - kuhi (heap)
5. set\map - Väga võimas ennast tasakaalustav kahendpuu. Kiire asümptootiline keerukus, aga praktikas palju aeglasem kui näiteks sorteerimine ja N kahendotsingut

Standardteegi kokkuvõtte

- Õppige kasutama mingit standardteegi juhist. Minu lemmik on: `http://www.cplusplus.com/reference/`
- Katsetage standardteegi kasutamist nii palju kui võimalik (eriti võistlusprogrammeerimise raames)
- Standardteegi tundmine tuleb kasuks igal pool

Implementeerimine

- Implementeerimine on loominguline tegevus
- On palju erinevaid viise sama asja implementeerida, mõned paremad kui teised
- Rõhuge oma koodi loetavusele ja elegantsusele. Need omadused on väga kasulikud näiteks:
  1. Vigade vältimisel ja parandamisel
  2. Väga keerukate asjade implementeerimisel
  3. Keskendumisele koodi kirjutamisel
  4. Koodi jagamisel teistega
- Järgnevad nipid võiksid olla kasulikud selle saavutamisel

Lokaalsus

- Deklareerige oma muutujad võimalikult väikeses skoobis
- Teeb selgemaks, kus ja kuidas neid muutujaid kasutatakse, suurendab koodi loetavust
- Võimaldab muutujanimesi paremini taaskasutada
- Aitab vältida muutujate segiajamist

Näide

```
//... includes, etc ...
int ind, a, b;
long long dp[20][20];

int main() {
    //... some code ...
    if(something) {
        //... use the variables ...
    }
}
```

→

```
//... includes, etc ...

int main() {
    //... some code ...
    if(something) {
        int ind, a, b;
        long long dp[20][20];
        //... use the variables ...
    }
}
```

Skoop

- Skoobi saab luua ka ilma if/while jms võtmesõnata
- Kasulik lokaalsuse loomiseks

Näide

```
//... includes, etc ...
int main() {
    //... some code ...

    int x, y, dx, dy;
    //... use those variables ...

    //... some unrelated code ...

    double xd, yd, dxd, dyd;
    //... use those variables ...
}
```

→

```
//... includes, etc ...
int main() {
    //... some code ...
    {
        int x, y, dx, dy;
        //... use those variables ...
    }
    //... some unrelated code ...
    {
        double x, y, dx, dy;
        //... use those variables ...
    }
}
```

Võtmesõna static

- Võimaldab deklareerida globaalseid muutujaid lokaalses skoobis, andes neile ka lokaalsuse eeliseid

Näide

```
//... includes, etc ...
int dp1[1001][1001];
double dp2[101][50001];

int function1() {
    //... calculation on dp1 ...
    return dp1[1000][1000]
}
double function2() {
    //... calculation on dp2 ...
    return dp2[100][50000];
}
// ... rest of the code ...
```

→

```
//... includes, etc ...

int function1() {
    static int dp[1001][1001];
    //... calculation on dp ...
    return dp[1000][1000]
}
double function2() {
    static double dp[101][50001];
    //... calculation on dp ...
    return dp[100][50000];
}
// ... rest of the code ...
```

Objekt Orienteeritud Programmeerimine

- OOP (Objekt Orienteeritud Programmeerimine) on väga võimas tööriist. Õppige seda kasutama!
- Lubab muutujaid, funktsioone jms loogiliselt ühendada
- Annab rohkem võimalusi lokaalsuse ärakasutamiseks
- Lubab funktsioone lokaalselt deklareerida

Näide

```
//... includes, etc ...
vector<int> arc[2][100000];
int weight[2][100000];

void construct(int i, int seed) {
    //uses arc[i] and weight[i]
}
int calculate(int i) {
    //uses arc[i] and weight[i]
}
int main() {
    construct(0, 15);
    construct(1, 2017);
    cout<<calculate(0)<<' ';
    cout<<calculate(1)<<'\n';
}
```

→

```
//... includes, etc ...
struct Graph {
    vector<int> arc[100000];
    int weight[100000];
    //constructor
    Graph(int seed) {
        //uses arc and weight
    }
    int calculate() {
        //uses arc and weight
    }
};
int main() {
    static Graph g1(15), g2(2107);
    cout<<g1.calculate()<<' ';
    cout<<g2.calculate()<<'\n';
}
```

Algväärtustusjada

- Annab väga elegantse viisi objektide väärtustamiseks
- Saab kasutada ka standardteegi objektide peal

Näide

```
//... includes, etc ...
struct Object {
    int cnt, val, size;
};
int main() {
    Object cur;
    cur.cnt = 1, cur.val = 10;
    cur.size = 2;
    vector<int> arr(3);
    arr[0] = 2, arr[1] = 15;
    arr[2] = 52;
}
```

→

```
//... includes, etc ...
struct Object {
    int cnt, val, size;
};
int main() {
    Object cur = {1, 10, 2};
    vector<int> arr = {2, 15, 52};
}
```

Lambda funktsioonid

- Võimaldab luua ühekordseid, nimetuid lokaalseid funktsioone
- Teeb funktsiooniga sorteerimise lihtsamaks ja loetavamaks

Näide

```
//... includes, etc ...
bool pred(Object l, Object r) {
    return l.cnt*l.val <
           r.cnt*r.val;
}
int main() {
    //... some code ...
    vector<Object> objects;
    //... construct objects ...
    sort(objects.begin(),
         objects.end(),
         pred);
}
```

→

```
//... includes, etc ...
int main() {
    //... some code ...
    vector<Object> objects;
    //... construct objects ...
    sort(objects.begin(),
         objects.end(),
         [](Object l, Object r)
        {return l.cnt*l.val <
                r.cnt*r.val;});
}
```

Pointerid

- Mugavam viis objektile viidata kui indeksid. Nt cur->next[1]->next[3] on parem kui next[next[cur][1]][3]
- Kasulik, kui on vaja leida objektide jadale erinevaid järjestusi

Näide

```
vector<Object*> byVal(n);
for(int i=0;i<n;i++) byVal[i] = &object[i];
sort(byVal.begin(), byVal.end(),
        [](Object* l, Object* r) {return l->val < r->val;});

vector<Object*> odd(n/2);
for(int i=1;i<n;i+=2) odd[i/2] = byVal[i];
sort(odd.begin(), odd.end(),
        [](Object* l, Object* r) {return l->size < r->size;});

for(int i=0;i<n/2;i++) odd[i]->result += i;
```

Kokkuvõte

- Võistlusprogrammeerimine aitab suunata teid palju koodi kirjutama
- Kasutage seda ära et arendada oma implementeerimisoskust ja et õppida kirjutama elegantsemat ja loetavamat koodi
- Eelmainitud nipid on ainult tööriistad, kasutage neid intelligentselt. Näiteks ärge üritage jõuga kõike OOP-iga teha, vaid kasutage seda siis kui sellest on kasu
- Implementeerimine on loominguline tegevus ja võib olla päris huvitav