Programiranje 2

Dobre preporuke C++

dr Đorđe Obradović

Singidunum - Centar Novi Sad

Interfejsi

Interfejsi treba da budu eksplicitni

```
0  int round(double d)
1  {
2      return (round_up) ? ceil(d) : d;     // don't: "invisible" dependency
3  }
```

Više poziva iste funkcije sa istim parametrima neće dati iste rezultate.

Izbegavajte globalne varijable

- teško je pratiti zavisnosti u velikom projektu

Interfejsi bi trebalo da budu tipizirani

Primer 1. LOŠE

```
0 void pass(void* data);
```

Primer 2. LOŠE

```
0 void draw_rect(int, int, int, int);
1
2 draw_rect(p.x, p.y, 10, 20);
```

Interfejsi bi trebalo da budu tipizirani

Primer 3. DOBRO

```
0 void draw_rectangle(Point top_left, Point bottom_right);
```

Interfejsi bi trebalo da budu tipizirani

Primer 4. LOŠE

```
0  void blink_led(int time_to_blink)
1  {
2      // ...
3      // do something with time_to_blink
4      // ...
5  }
6
7  void use()
8  {
9      blink_led(2);
10 };
```

Interfejsi bi trebalo da budu tipizirani

Primer 5. DOBRO

```
0  void blink_led(milliseconds time_to_blink)
1  {
2      // ...
3      // do something with time_to_blink
4      // ...
5  }
6
7  void use()
8  {
9      blink_led(1500ms);
10 }
```

I:5 Ograničenja na argumentima

```
0 double sqrt(double x);
1
2 double sqrt(double x) { Expects(x >= 0); /* ... */ }
```

I:7 Ograničenja na povratnoj vrednosti

```
0  int area(int height, int width) {
1      return height * width;
2  }
3
4  int area(int height, int width)
5  {
6      auto res = height * width;
7      Ensures(res > 0);
8      return res;
9  }
```

I:7 Stanje sistema nakon izvršenja

```
0  mutex m;
1
2  void manipulate(Record& r)     // don't
3  {
4      m.lock();
5      // ... no m.unlock() ...
6  }
```

I:10 Izuzeci

```
try {
   // protected code
} catch( ExceptionName e1 ) {
   // catch block
} catch( ExceptionName e2 ) {
   // catch block
} catch( ExceptionName eN ) {
   // catch block
}
```

I:10 Kodovi greške

```
0  int val;
1  int error_code;
2  tie(val, error_code) = do_something();
3  if (error_code) {
4      // ... handle the error or exit ...
5  }
```

I:11 Prenos vlasništva

```
0 X* compute(args)     // don't
1 {
2     X* res = new X{};
3     // ...
4     return res;
5 }
```

I:11 Prenos vlasništva

```
0 vector<double> compute(args)  // good
1 {
2     vector<double> res(10000);
3     // ...
4     return res;
5 }
```

I:13 Prenos niza kao pokazivača

```
0 void copy_n(const T* p, T* q, int n);
1 // copy from [p:p+n) to [q:q+n)
```

I:13 Prenos niza kao pokazivača

```
0 void draw(Shape* p, int n);  // poor interface; poor code
1 Circle arr[10];
2 // ...
3 draw(arr, 10);
```

I:23 Minimalan broj argumenata

```
0 template<class InputIterator1,
1 class InputIterator2, class OutputIterator,
2 class Compare>
3 OutputIterator merge(InputIterator1 first1, InputIterator1 last1,
4
5 InputIterator2 first2, InputIterator2 last2,
6 OutputIterator result, Compare comp);
```

Template funkcije

```
0  #include <iostream>
1  using namespace std;
2
3  template <class T>
4  T GetMax (T a, T b) {
5    T result;
6    result = (a>b)? a : b;
7    return (result);
8  }
9
10 int main () {
11   int i=5, j=6, k;
12   long l=10, m=5, n;
13   k=GetMax<int>(i,j);
14   n=GetMax<long>(l,m);
15   cout << k << endl;
16   cout << n << endl;
```

Abstraktne klase

```
0  class Shape {  // bad: interface class loaded with data
1  public:
2      Point center() const { return c; }
3      virtual void draw() const;
4      virtual void rotate(int);
5      // ...
6  private:
7      Point c;
8      vector<Point> outline;
9      Color col;
10 };
```

Abstraktne klase

```
0  class Shape {    // better: Shape is a pure interface
1  public:
2      virtual Point center() const = 0;   // pure virtual function
3      virtual void draw() const = 0;
4      virtual void rotate(int) = 0;
5      // ...
6      // ... no data members ...
7  };
```

Funkcije

Uvod

- Šta su funkcije?
- Kako?
- Imenovanje funkcija
- Funkcije bi trebalo da izvršavaju jednu logičku operaciju
- Jednostavnost u pisanju funkcija
- Inline funkcije
- Funkcije koje mogu da se izvršavaju za vreme kompajliranja

Prenos parametara

- prenos preko vrednosti
- prenos preko adrese
- prenos preko pokazivača

Kako eksplicitno naznačiti ulazne i izlazne parametre

Prenos parametara

```
0  void f1(const string& s);  // OK: pass by reference to const; always cheap
1
2  void f2(string s);         // bad: potentially expensive
3
4  void f3(int x);            // OK: Unbeatable
5
6  void f4(const int& x);     // bad: overhead on access in f4()
```

Prenos parametara

```
0  Matrix operator+(const Matrix& a, const Matrix& b)
1  {
2      Matrix res;
3      // ... fill res with the sum ...
4      return res;
5  }
6
7  Matrix x = m1 + m2;  // move constructor
8
9  y = m3 + m3;         // move assignment
```

Prenos parametara

```
0  void update(Record& r);
1  // assume that update writes to r
```

Prenos parametara

```
0  int f(const string& input, /*output only*/ string& output_data)
1  {
2      // ...
3      output_data = something();
4      return status;
5  }
```

Prenos parametara

```
0  // GOOD: self-documenting
1  tuple<int, string> f(const string& input)
2  {
3      // ...
4      return make_tuple(status, something());
5  }
```

Prenos parametara

```
0  void use(int* p, int n, char* s, int* q)
1  {
2      p[n - 1] = 666; // Bad: we don't know if p points to n elements;
3                      // assume it does not or use span<int>
4      cout << s;      // Bad: we don't know if that s points to a zero-terminated array of char;
5                      // assume it does not or use zstring
6      delete q;       // Bad: we don't know if *q is allocated on the free store;
7                      // assume it does not or use owner
8  }
```

Nikada ne vracati pokazivač ili referencu na lokalni objekat

```
0  int* f()
1  {
2      int fx = 9;
3      return &fx;  // BAD
4  }
5
6  void h()
7  {
8      int* p = f();
9      int z = *p;  // read from abandoned stack frame (bad)
10     g(p);        // pass pointer to abandoned stack frame to function (bad)
11 }
```

Nikada ne vracati pokazivač ili referencu na lokalni objekat

```
0  int& f()
1  {
2      int x = 7;
3      // ...
4      return x;  // Bad: returns reference to object that is about to be destroyed
5  }
```

Nikada ne vracati pokazivač ili referencu na lokalni objekat

```
0  class Car
1  {
2      array<wheel, 4> w;
3      // ...
4  public:
5      wheel& get_wheel(int i) { Expects(i < w.size()); return w[i]; }
6      // ...
7  };
8
9  void use()
10 {
11     Car c;
12     wheel& w0 = c.get_wheel(0); // w0 has the same lifetime as c
13 }
```

Klase

Uvod

- Bolja organizacija koda
- Većina prethodnih problema se lakše rešava
- Bolja dokumentacija koda
- Lakše testiranje i ispravljanje grešaka

Sintaksa

```
0  class Rectangle {
1      int width, height;
2    public:
3      void set_values (int,int);
4      int area (void);
5  } rect;
```

Sintaksa

```
0  #include <iostream>
1  using namespace std;
2
3  class Rectangle {
4      int width, height;
5    public:
6      void set_values (int,int);
7      int area() {return width*height;}
8  };
9
10 void Rectangle::set_values (int x, int y) {
11   width = x;
12   height = y;
13 }
14
15 int main () {
16   Rectangle rect;
17   rect.set_values (3,4);
```