Введение в C++

- Тип определяется поведением и набором состояний, в которых может находится экземпляр типа (объект этого типа)
- Вся система представляет собой набор объектов различного типа, которые обмениваются сообщениями
- Основные понятия
  - Абстракция данных
  - Инкапсуляция
  - Полиморфизм
  - Наследование

Введение в C++

Основные понятия. Абстракция

Пусть наш код использует набор случайных чисел

```
int rand (void);
int seed;
void srand (unsigned int seed);

srand(10); // Initialization
int num = rand(); // Using
```

или

```
FILE *rnd = fopen("/dev/random", "r"); //
    Initialization
int num;
fread(rnd, &num, sizeof(num)); // Using
```

Нам нужно только операции создания/уничтожения генератора и операция получения числа

Введение в C++

Основные понятия. Абстракция

Выделяем интерфейс и внутреннее представление

```
typedef struct RandomGenerator
{
    int seed;
    int call_count;
    int (*next) (RandomGenerator *rnd);
    RandomGenerator *(*free)(RandomGenerator *rnd);
} RandomGenerator;

RandomGenerator *random_create(int seed);
```

```
RandomGenerator *rr = random_create(1234);
for (int j = 0; j < 100; ++j) {
  printf("%d\n", rr->next(rr));
}
rr = rr->free(rr);
```

Введение в C++

Основные понятия. Абстракция

```
class RandomGenerator
{
........
    int call_count_;
    int seed_;
.......
    int next();
    RandomGenerator(int seed);
    ~RandomGenerator() {}
};
```

```
{
    RandomGenerator rnd(1234);
    for (int j = 0; j < 100; ++j) {
        printf("%d\n", rnd.next());
    }
}
```

Введение в C++

Основные понятия. Инкапсуляция

Все «отлично» работает, клиент может испортить внутреннее состояние

```
RandomGenerator *rnd = random_create(1234);
for (int j = 0; j < 100; ++j) {
  printf("%d\n", rnd->next(rnd));
  rnd->call_count *= 2;
}
printf("%d\n", rnd->call_count_);
rnd = rnd->free(rnd);
```

Все отлично *не* работает, клиент **не** может испортить внутреннее состояние

```
RandomGenerator rnd(1234);
for (int j = 0; j < 100; ++j) {
    printf("%d\n", rnd.next());
    rnd.call_count_ *= 2;
}
printf("%d\n", rnd.call_count_);
```

```
error: 'int RandomGenerator::call_count_' is private
```

Введение в C++

Основные понятия. Инкапсуляция

```
class RandomGenerator
{
private:
    int call_count_;
    int seed_;
public:
    int next();
    RandomGenerator(int seed);
    ~RandomGenerator() {}
};
```

Квалификаторы доступа

- public
- private
- .....

Введение в C++

Основные понятия. Инкапсуляция

```
class RandomGenerator
{
    int call_count_;
    int seed_;
public:
    int next();
    RandomGenerator(int seed);
    ~RandomGenerator() {}
    int getCallCount() {
        return call_count_;
    }
};
```

```
RandomGenerator rnd(1234);
for (int j = 0; j < 100; ++j) {
    printf("%d\n", rnd.next());
//  rnd.getCallCount() *= 2;
}
printf("%d\n", rnd.getCallCount());
```

Итак, в C у нас был интерфейс генератора случайных чисел и две реализации

```
struct Random;
typedef struct Random Random;

typedef struct RandomOps
{
    Random *(*free)(Random *rnd);
    int (*next)(Random *rnd, int n);
} RandomOps;

Random *random_create();

struct Random
{
    RandomOps *ops;
};
```

Введение в C++

Основные понятия. Наследование. Пример №1 реализации на C

```
typedef struct RandomRand
{
    RandomOps *ops;
    int seed;
} RandomRand;

int random_rand_next(Random *rnd, int n)
{
    return (int)(rand() / (RAND_MAX + 1.0) * n);
}

static RandomOps random_rand_ops = { 0, random_rand_next, };

Random *random_rand_create()
{
    RandomRand *rnd = (RandomRand*) calloc(1, sizeof(*rnd));
    rnd->ops = &random_rand_ops;
    rnd->seed = time(NULL);
    return (Random*)rnd;
}
```

Введение в C++

Основные понятия. Наследование. Пример №2 реализации на C

```
typedef struct RandomDev
{
    RandomOps *ops;
    FILE *f;
} RandomDev;

int random_dev_next(Random *rnd, int n)
{
    unsigned int val;
    fread(&val, sizeof(val), 1, ((RandomDev*)rnd)->f);
    return (int)(val / UINT_MAX * n);
}

static RandomOps random_dev_ops = { 0, random_dev_next, };

Random *random_dev_create()
{
    RandomDev *rnd = (RandomDev*) calloc(1, sizeof(*rnd));
    rnd->ops = &random_dev_ops;
    rnd->f = fopen("/dev/random", "r");
    return (Random*)rnd;
}
```

У нас есть общий интерфейс базового Random и две реализации

```
Random *rnd_first = random_dev_create();
Random *rnd_second = random_rand_create();
Random *rnd_third = random_create(cfg); // Depends on
    config

int one = rnd_first->ops->next(rnd_first, 10);
int two = rnd_second->ops->next(rnd_second, 10);
int three = rnd_third->ops->next(rnd_third, 10);
```

Введение в C++

Основные понятия. Наследование. Интерфейс и пример №1 в C++

```
class Random
{
public:
    int next(int n);
    Random() {};
};
```

```
class RandomRand : Random
{
    int seed_;
public:
    int next(int n)
    {
        return (int)(rand() / (RAND_MAX + 1.0) * n);
    }
    RandomRand()
    {
        seed_ = time(NULL);
    }
};
```

Введение в C++

Основные понятия. Наследование. Пример №2 в C++

```
class RandomDev : Random
{
    FILE *f_;
public:
    int next(int n)
    {
        unsigned int val;
        fread(&val, sizeof(val), 1, f_);
        return (int)(val / UINT_MAX * n);
    }
    RandomDev();
    ~RandomDev()
    {
        fclose(f_);
    }
};

RandomDev::RandomDev()
{
    f_ = fopen("/dev/random", "r");
}
```

```
Random *rnd_first = new RandomRand();
Random *rnd_second = new RandomDev();

int one = ((RandomRand*)rnd_first)->next(10);
int two = ((RandomDev*)rnd_second)->next(10);
int three = ((RandomDev*)rnd_first)->next(10); //
    hm???

delete rnd_first;
delete rnd_second;
```

Проблема в том, что у нас есть приведение типов

Выбор поведения в зависимости от типа.
Какой метод вызывается для объекта решается статически или динамически в зависимости от типа объекта

```
Random *rnd_first = new RandomRand();
Random *rnd_second = new RandomDev();

int one = rnd_first->next(10); // RandomRand::next()
int two = rnd_second->next(10); // RandomDev::next()

delete rnd_first;
delete rnd_second;
```

Введение в C++

Основные понятия. Динамический полиморфизм.

```
class Random {
public: virtual int next(int n);
};

class RandomDev : Random {
public: virtual int next(int n) {
        unsigned int val;
        fread(&val, sizeof(val), 1, f_);
        return (int)(val / UINT_MAX * n);
    }
};

class RandomRand : Random {
public: virtual int next(int n) {
        return (int)(rand() / (RAND_MAX + 1.0) * n);
    }
};
....
Random *rnd_first = new RandomRand();
Random *rnd_second = new RandomDev();
int one = rnd_first->next(10); // RandomRand::next()
int two = rnd_second->next(10); // RandomDev::next()
```

Введение в C++

Основные понятия. Статический полиморфизм.

```
class Complex {
public:
    int re, im;
    Complex(int re, int im) {
        this->re = re;
        this->im = im;
    }
};

Complex sum(Complex &a, Complex &b) {
    return Complex(a.re + b.re, a.im + b.im);
}

int sum(int a, int b) {
    return a + b;
}
....
Complex a(1, 0), b(0, 1);
int c = 1, d = 2;

a = sum(a, b);
int h = sum(c, d);
```

Введение в C++

Основные понятия. Статический полиморфизм, перегрузка операторов

```
class Complex {
public:
    int re, im;
    Complex(int re, int im) : re(re), im(im) { }
    Complex operator+(int a) {
        return Complex(re + a, im);
    }
};

....
Complex a(1, 0);
Complex b = a + 5;
Complex c = a.operator+(17);
```

Введение в C++

Отличия от C (1)

- Типы
  - char, int, float, double, **bool**, **wchar_t**
  - **Интегральные типы** = логический + символьные + целые
  - **Арифметические типы** = интегральные + типы с плавающей точкой
  - **Пользовательские типы** = перечисления (enum) + классы, остальные – встроенные
- Константы
  - true, false
  - символьные константы имеют тип **char**
- Для работы с символьными строками

```
printf("%s\n", (sizeof('a') == sizeof(char)) ? "C++" : "C");
```

Введение в C++

Отличия от C (2)

- Запрет неявного объявления переменных (int) и функций
- Отсутствие неизвестных аргументов в объявлении функции

```
long f(); <=> long f(void);
```

- C $\not\subset$ C++

Введение в C++

Работа с динамической памятью

```
class Complex;
...
// Allocation & Initialization
Complex *pc = new Complex(1, 2);
int *pi = new int(2);
delete pc;
delete pi;

long *pl = new long[10];
delete[] pl;
```

Введение в C++

Ссылочный тип

- **Ссылка** (reference) — альтернативное имя объекта (псевдоним, alias) , можно понимать как «безопасные» указатели

```
X& - reference to variable of type X

int i = 3;
int &ri = i;// Initialization, not assignment
ri = 5; // No reference changes
double &rd; // Error: no initialization
```

- Отличия от указателей
  - При объявлении ссылка обязательно инициализируется ссылкой на уже существующий объект данного типа
  - Ссылка пожизненно указывает на один и тот же адрес
  - При обращении к ссылке операция * производится автоматически