Указатели на функции, методы и члены данных

Александр Смаль

CS центр
24 октября 2017
Санкт-Петербург

Указатели на функции

Кроме указателей на значения в C++ присутствуют три особенных типа указателей:

1. указатели на функции (унаследованно из C),
2. указатели на методы,
3. указатели на поля классов.

Указатели на функции (и методы) используются для

1. параметризация алгоритмов,
2. обратных вызовов (callback),
3. подписки на события (шаблон Listener),
4. создание очередей событий/заданий.

Указатели на функции: параметризация алгоритмов

```
void qsort (void* base, size_t num, size_t size,
            int (*compar)(const void*,const void*));
```

Как это работает?

```
bool less   (double a, double b) { return a < b; }
bool greater(double a, double b) { return a > b; }

void sort(double * p, double * q, bool (*cmp)(double, double)) {
    for (double * m = p; m != q; ++m)
        for (double * r = m; r + 1 != q; ++r)
            if ( cmp(*(r + 1), *r) )
                swap(*r, *(r + 1));
}
int main() {
    double m[100];
    sort(m, m + 100, &less);
    sort(m, m + 100, &greater);
}
```

Указатели на функции: создание потока

```
#include <pthread.h>
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void *(*start_routine) (void *), void *arg);

double somedata[10][100];

void * thread_fun(void * data) {
    size_t const N = *((size_t *) data);
    sort(somedata[N], somedata[N] + 100);
    return 0;
}
int main( ) {
    pthread_t threads[10];
    size_t id[] = {0,1,2,3,4,5,6,7,8,9};
    for (size_t i = 0; i != 10; ++i)
        pthread_create(&threads[i], 0, &thread_fun, &id[i]);
    ...
}
```

Сразу о полезности typedef

Что здесь объявлено?

```
char * (*func(int, int))(int, int, int *, float);
```

Сразу о полезности typedef

Что здесь объявлено?

```
char * (*func(int, int))(int, int, int *, float);
```

Функция двух целочисленных параметров, возвращающая указатель на функцию, которая возвращает указатель на `char` и имеет собственный список формальных параметров вида: `(int, int, int *, float)`

Сразу о полезности typedef

Что здесь объявлено?

```
char * (*func(int, int))(int, int, int *, float);
```

Функция двух целочисленных параметров, возвращающая указатель на функцию, которая возвращает указатель на `char` и имеет собственный список формальных параметров вида: `(int, int, int *, float)`

Как стоило это написать:

```
typedef char* (*SomeFunction)(int, int, int *, float);

SomeFunction func(int, int);
```

Указатели на методы

В отличие от указателей на функции требуют экземпляр класса.

```
struct Person {
        string name()       const;
        string surname()    const;
        string address()    const;
};
typedef string (Person::*person_method)() const;

void print(Person const& p) {
    static person_method im[3] =
        {&Person::name, &Person::surname, &Person::address};

    for (size_t i = 0; i != 3; ++i)
        cout << (p.*im[i])();
}                                   // person_method mem
void sort(Person * p, Person * q, string (Person::*mem) () const);

sort(persons, persons + 100, &Person::name);
sort(persons, persons + 100, &Person::surname);
```

Указатели на члены данных

Похожи на указатели на методы.

```
struct Person {
        string name;
        string surname;
        string address;
};
typedef string Person::*person_field;

void print(Person const& p) {
    static person_field im[3] =
        {&Person::name, &Person::surname, &Person::address};

    for (size_t i = 0; i != 3; ++i)
        cout << (p.*im[i]);
}                                   // person_field mem
void sort(Person * p, Person * q, string Person::*mem);

sort(persons, persons + 100, &Person::name);
sort(persons, persons + 100, &Person::surname);
```

Резюме по синтаксису

Указатели на функции.

```
int foo(double d) { return 0; }
int main() {
    int (*fptr)(double) = foo;
    return fptr(3.5);
}
```

Указатели на методы и поля класса.

```
struct Student {
    string name () const { return name_; }
    string name_;
};
int main() {
    string (Student::*mptr)() const = &Student::name;
    string  Student::*dptr          = &Student::name_;
    Student s;
    Student * p = &s;
    (s.*mptr)() == (p->*mptr)();
    (s.*dptr)   == (p->*dptr);
}
```

Шаблон Listener

Решение с помощью ООП:

```
struct Button;

struct ButtonListener {
    virtual void onButtonClick(Button * b, bool down) = 0;
    virtual ~ButtonListener(){}
};
struct Button {
    void subscribe( ButtonListener * bl );
};
```

Решение с помощью указателей на функции:

```
struct Button;
typedef void (*ButtonProc)(Button *, bool, void *);
struct Button {
    void subscribe( ButtonProc bp, void * arg );
};
```

Как такие указатели устроены?

Что хранится в указателе на функцию?

Как такие указатели устроены?

Что хранится в указателе на функцию?

Хранится адрес функции.

Что хранится в указателе на поле класса?

Как такие указатели устроены?

Что хранится в указателе на функцию?

Хранится адрес функции.

Что хранится в указателе на поле класса?

Хранится смещение поля от начала объекта.

Что хранится в указателе на метод?

Как такие указатели устроены?

Что хранится в указателе на функцию?

Хранится адрес функции.

Что хранится в указателе на поле класса?

Хранится смещение поля от начала объекта.

Что хранится в указателе на метод?

Там хранятся:

1. адрес метода,
2. номер в таблице виртуальных методов,
3. смещение.

Важные моменты

- Использование неинициализированных указателей на функции и методы влечёт неопределённое поведение.
- В шаблонном коде указатель на функцию ведёт себя так же, как и объект класса с оператором `()`. Это позволяет использовать указатели на функции в качестве функторов.
- Для использования указателей на методы и поля классов нужны экземпляры этих классов.
- Указатели на методы и поля класса ни к чему не приводятся (используется для safe bool).
- Используйте `typedef`! $=)$.