

Язык C++

Reference, initialization, function overloading
namespaces

Ссылки (reference)

```
void swap(int* pi, int* pj) {  
    int temp = *pi;  
    *pi = *pj;  
    *pj = temp;  
}
```

Reference (lvalue reference)

- “Псевдоним” для уже существующего объекта
- Обязательно инициализирована
- Не занимает дополнительную память
- Нельзя сделать указатель на ссылку
- Продлевают “жизнь” временным переменным

Reference

```
int main() {  
    int i = 10;  
    int& j = i;  
    //int& k; // error: 'k' declared as reference but not initialized  
    j = 20;  
    std::cout << i << std::endl;  
    std::cout << &i << " " << &j << std::endl;  
  
    const int& r = i;  
    // r = 21; // error assignment of read-only reference 'r'  
    return 0;  
}
```

Reference

```
void swap(int& i, int& j) {  
    int temp = i;  
    i = j;  
    j = temp;  
}
```

Rvalue reference

```
int&& rv = 1;
```

```
// soon...
```

Reference

```
int& foo() {  
    int i = 20;  
    return i;  
}  
  
int main() {  
    int& x = foo();  
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9 ,10, 11, 12};  
  
    std::cout << x; // !OOps  
    return 0;  
}
```

Передача аргументов в функцию

- By reference
 - В общем случае наиболее “дешевый” и простой способ
- By pointer
- By value
 - Для встроенных и небольших типов

Передача аргументов в функцию

```
struct SomeStruct ;
```

```
void funcA(const SomeStruct& value); // by reference
```

```
void funcB(SomeStruct& value); // by reference
```

```
void funcC(SomeStruct* value); // by pointer
```

```
void funcD(int value); // by value
```

Перегрузка функций

```
void print() {  
    std::cout << std::endl;  
}  
  
void print(int x) {  
    std::cout << x << std::endl;  
}
```

Перегрузка функций

```
void print(const char* str) {  
    std::cout << str << std::endl;  
}  
  
void print(int x, int y) {  
    std::cout << x << " " << y << std::endl;  
}
```

Перегрузка функций

```
int main() {  
    print();  
    print(10);  
    print("Hello world!");  
    print(10, 20);  
    // print(20.1);  
  
    return 0;  
}
```

Перегрузка функций

```
int test(int i) {  
    return i;  
}  
  
float test(float f) {  
    return f;  
}  
  
int main() {  
    test(10.2f);  
    return 0;  
}
```

Перегрузка функций

```
int test(int i) {  
    return i;  
}  
  
float test(int f) { // error: ambiguating new declaration of 'float test(int)'  
    return (float)f;  
}  
  
int main() {  
    test(10.2f);  
    return 0;  
}
```

nullptr

```
#include <iostream>

void func(int a) {
    std::cout << "int" << std::endl;
}

void func(const char* str) {
    std::cout << "char*" << std::endl;
}

int main() {
    func(10);           // int
    func("aa");         // char*
    //func(NULL);       // call is ambiguous
    func(0);            // int
    func(nullptr);      //char*
}
```

nullptr

```
#include <iostream>

void func(char* str) {
    std::cout << "char*" << std::endl;
}

void func(int* i) {
    std::cout << "int*" << std::endl;
}

int main() {
    int* i = nullptr;
    func(i);           // int*
    func("aa");        // char*
    func(nullptr);     // call is ambiguous
}
```


nullptr

```
void func(const char* str) {  
    std::cout << "char*" << std::endl;  
}  
  
void func(int* i) {  
    std::cout << "int*" << std::endl;  
}  
  
void func(std::nullptr_t null_pointer) {  
    std::cout << "std::nullptr" << std::endl;  
}
```

Аргументы по умолчанию

```
struct SPoint {  
    int x;  
    int y;  
};  
  
SPoint make_point(int x = 0, int y = 0) {  
    return {x,y};  
}  
  
int main(int argc, char const *argv[]) {  
    SPoint p1 = make_point(10, 20);  
    SPoint p2 = make_point();  
    return 0;  
}
```

Аргументы по умолчанию

```
#include <iostream>
#include <vector>

void printVector(const std::vector<int>& data, char delimiter = ',') {
    for (const int& i: data) {
        std::cout << i << delimiter;
    }
}

int main() {
    std::vector<int> v {1,2,3,4,5};
    printVector(v);
}
```

Инициализация

- Default initialization
- Value initialization
- Direct initialization
- Copy initialization
- List initialization
- Aggregate initialization

<https://en.cppreference.com/w/cpp/language/initialization>

Инициализация

```
int main() {  
    int i();    // function  
    int k = 2; // copy initialization  
    int j(2);   // direct initialization  
    int l{};    //value initialization  
    std::string str {'a', 'b', 'c'}; // list initialization  
    SCircle c {{1,2}, 3}; // aggregate initialization  
}
```

namespace

- Предотвращают конфликт имен
- Могут состоять из нескольких блоков
- Упрощают “читабельность” кода
- Unnamed namespace
- Namespace alias

namespace

```
namespace Foo {  
    void f() {  
        std::cout << "Foo" << std::endl;  
    }  
}  
  
namespace Boo {  
    void f() {  
        std::cout << "Boo" << std::endl;  
    }  
}
```

namespace

```
int main() {  
    Foo::f();  
    Boo::f();  
  
    return 0;  
}
```


namespace

```
using namespace Foo;  
int main() {  
    f();  
    Boo::f();  
  
    return 0;  
}
```

namespace

```
using namespace Foo;
using namespace Boo;

int main() {
    f();          // call to 'f' is ambiguous
    return 0;
}
```

namespace

```
using namespace Foo;
using namespace Boo;

int main() {
    f();          // call to 'f' is ambiguous
    return 0;
}
```

namespace

```
int main() {  
    using namespace Foo;  
  
    f();  
    return 0;  
}
```

namespace

```
namespace Foo {  
    namespace SomeLogNamespaceName {  
        void f() {  
            std::cout << "Foo" << std::endl;  
        }  
    }  
}  
  
int main() {  
    Foo::SomeLogNamespaceName::f();  
  
    return 0;  
}
```

namespace alias

```
namespace Foo {  
    namespace SomeLogNamespaceName {  
        void f() {  
            std::cout << "Foo" << std::endl;  
        }  
    }  
}  
  
int main() {  
    namespace SN = Foo::SomeLogNamespaceName;  
  
    SN::f();  
    return 0;  
}
```

namespace

```
#include <iostream>

namespace {
    void f () {
        std::cout << "Unnamed namespace function\n" ;
    }
}

int main() {
    f();
    return 0;
}
```

namespace

```
namespace {  
    int counter;  
}  
void increment() {  
    counter++;  
}  
namespace A {  
    namespace {  
        int counter;  
    }  
    void increment() { counter++; }  
}
```


namespace

```
int main() {  
    counter++;  
    increment();  
    A::increment();  
  
    std::cout << counter << std::endl;  
    std::cout << A::counter << std::endl;  
}
```