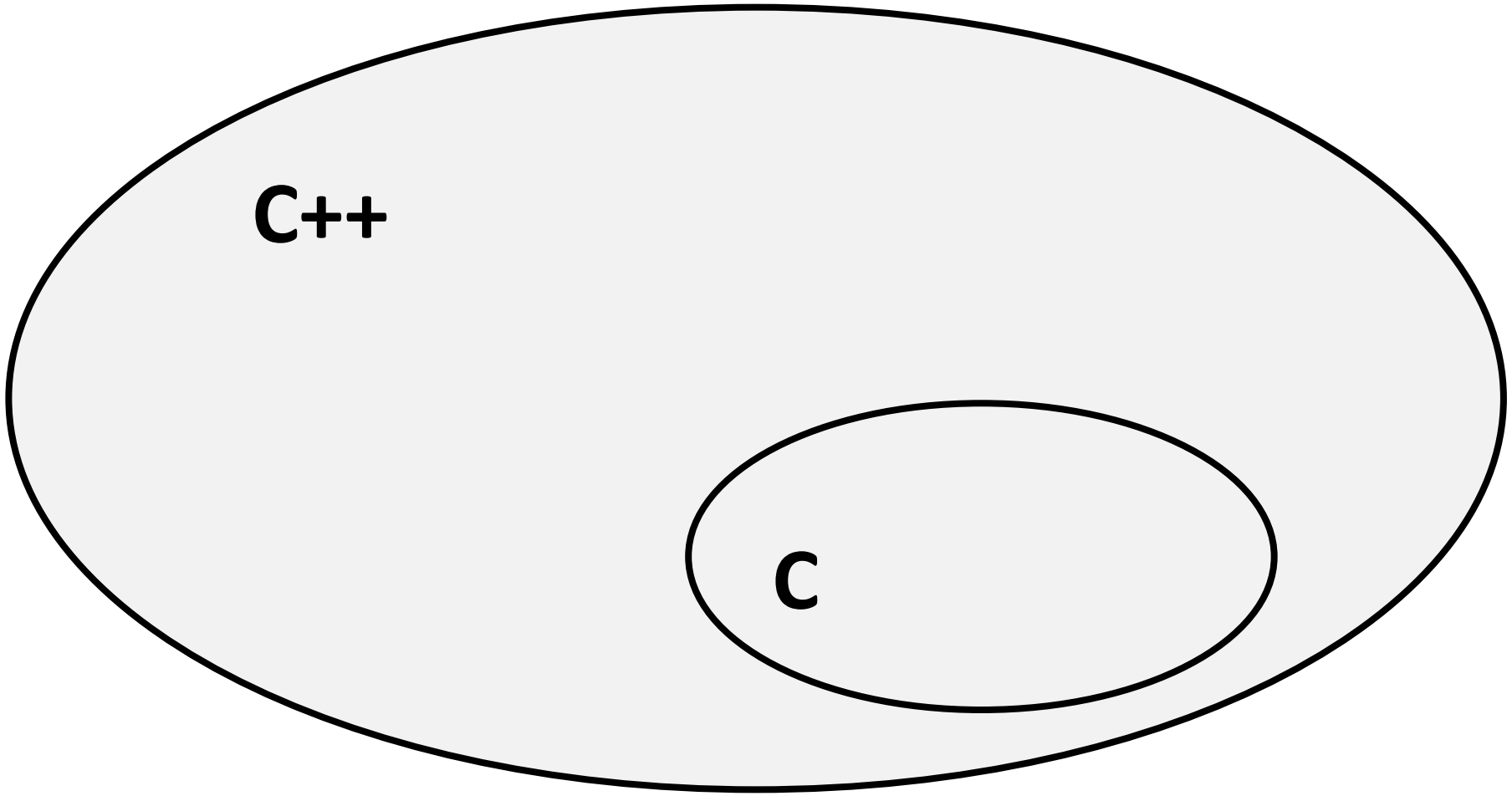
Основы программирования на C++

Дерево языка

Основы C++



Пример программы

C

```c
#include <stdio.h>

int main()
{
    printf("Hello World!\n");
}
```

C++

```cpp
#include <iostream>

int main()
{
    std::cout << "Hello World!\n";
}
```

Пространство имен

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello World!\n";
}
```

Пространство имен — это декларативная область, в рамках которой определяются различные идентификаторы (имена типов, функций, переменных, и т. д.).

Пространства имен

```cpp
#include <iostream>

namespace Summ
{
    int Func(int a, int b) {
        return a + b;
    }
}

namespace Mult
{
    int Func(int a, int b) {
        return a * b;
    }
}

int main()
{
        std::cout << "Mult = " << Mult::Func(5, 5) << std::endl;
        std::cout << "Summ = " << Summ::Func(5, 5) << std::endl;
}
```

```
Mult = 25
Summ = 10
```

Потоки ввода/вывода

C

```c
#include <stdio.h>

int main()
{
    int x;
    scanf_s("%d", &x);
    printf("X = %d\n", x);
    return 0;
}
```

C++

```cpp
#include <iostream>

int main()
{
    int x;
    std::cin >> x;
    std::cout << "X = " << x << std::endl;
}
```

Выделение динамической памяти

C

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
        int N;
        scanf_s("%d", &N);
        int* p = (int*)malloc(N * sizeof(int));
        for (size_t i = 0; i < N; ++i)
        {
            p[i] = i;
        }
        for (size_t i = 0; i < N; ++i)
        {
            printf("%d ",p[i]);
        }
        printf("\n");
        free(p);
        return 0;
}
```

C++

```cpp
#include <iostream>

int main()
{
        int N;
        std::cin >> N;
        int* p = new int[N];
        for (size_t i = 0; i < N; ++i)
        {
            p[i] = i;
        }
        for (size_t i = 0; i < N; ++i)
        {
            std::cout << p[i] << " ";
        }
        std::cout << std::endl;
        delete []p;
        return 0;
}
```

Ссылки

```
#include <iostream>
using namespace std;

void func_1(int& x){
    x++;
}

void func_2(int x){
    x++;
}

void main()
{
    int a = 10;
    cout << a << endl;
    func_1(a);
    cout << a << endl;
    func_2(a);
    cout << a << endl;
}
```

Ссылка в C++ -- это альтернативное имя объекта.

```
10
11
11
```

Rvalue / Lvalue

Rvalue, Lvalue – свойство синтаксической записи выражения

Lvalue	Rvalue
Идентификатор (имя переменной) Результат присваивания Префиксный инкремент/декремент Результат разыменования Результат функции, если она возвращает lvalue-reference	Литерал (присваиваемое значение) Результат выражения Постфиксный инкремент/декремент Результат функции, если она возвращает rvalue-reference

```
int value1 = 7;
int* pValue1 = &++value1;  //OK
int* pValue1 = &value1++;  //NO OK
```

X=5

Y=1+1

++X

Y++

Rvalue – временный объект, у него нет адреса

Lvalue-reference

```
int x = 10;
int& y = x;
```

Lvalue-reference

```
int& z = 2;          //нельзя. Пытаемся Lvalue-reference инициализировать rvalue
const int& z = 2;     //А так можно
```

Rvalue-reference

```
int&& y = 10;
```

Rvalue-reference

```
int&& y = x;   //Нельзя. Пытаемся в Rvalue-reference инициализировать lvalue
int&& y = 10; //Можно
```

Ссылка на число 10 будет видна до выхода из области видимости.

Move семантика

```
template < typename T>
typename remove_reference<T>::type&&
move(T&& param)
{
    using ReturnType =
    typename remove_reference<T>::type&&;
    return static_cast<ReturnType>(param);
}
```

std::move используется для указания того, что объект может быть «перемещен» (не скопирован)

В частности, std::move создает выражение xvalue, которое идентифицирует его аргумент.

std : : move выполняет безусловное приведение своего аргумента к **rvalue**

Сам по себе move ничего не перемещает

```
remove_reference<T> – удаляем все возможные ссылки, навешанные до
type&& – возвращаемое значение rvalue-reference
```

Move семантика

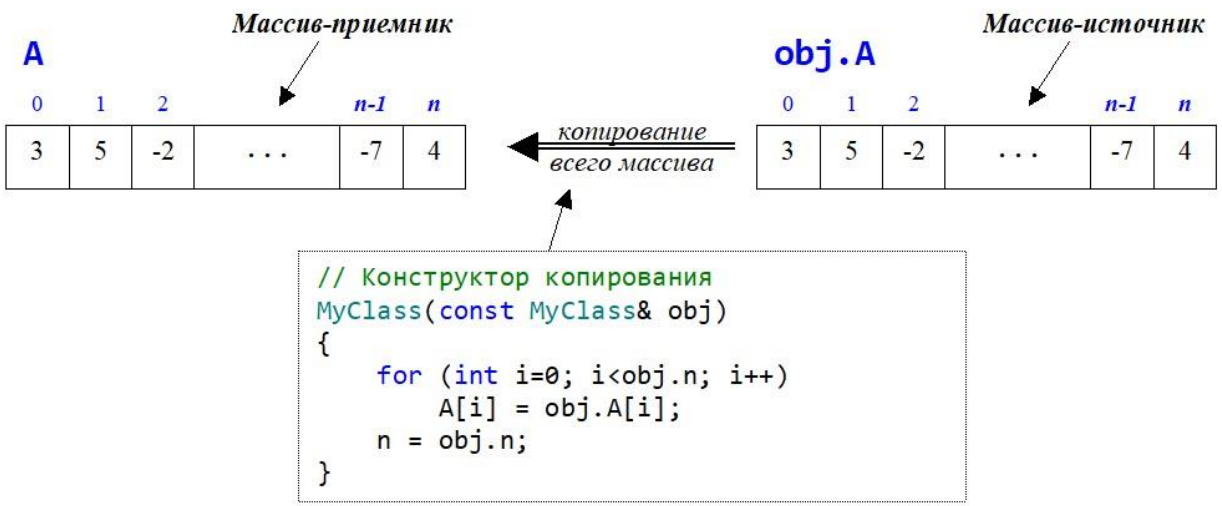
```cpp
#include <utility>      // std::move
#include <iostream>     // std::cout
#include <vector>       // std::vector
#include <string>       // std::string

int main() {
    std::string foo = "Hello";
    std::string bar = "world!";
    std::vector<std::string> myvector;
    myvector.push_back(foo);                    // copies
    myvector.push_back(std::move(bar));         // moves

    std::cout << "myvector contains:";
    for (std::string& x : myvector) std::cout << ' ' << x;
    std::cout << '\n';
    std::cout << foo <<'\n';
    std::cout << bar << '\n';
    return 0;
}
```

```
myvector contains: Hello world!
Hello
```

Значение bar было перемещено и в итоге не вывелось на экран

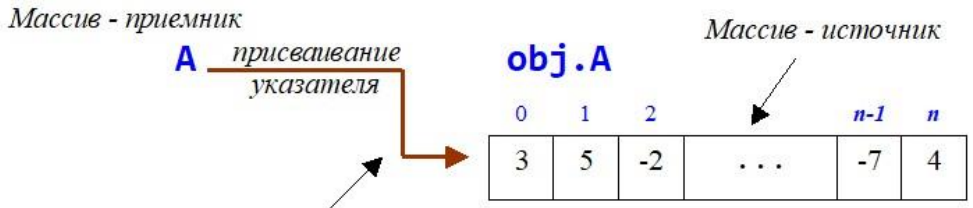
В репозитории пример с расчетом времени

Move семантика




```
// Конструктор копирования
MyClass(const MyClass& obj)
{
    for (int i=0; i<obj.n; i++)
        A[i] = obj.A[i];
    n = obj.n;
}
```

Вызов конструктора копирования




```
// Конструктор переноса
MyClass(MyClass&& obj)
{
    A = obj.A;
    n = obj.n;
    obj.n = 0; // это нужно обязательно!
}
```

Вызов конструктора перемещения