Содержание

RAII

Объекты классов могут на протяжении всего своего существования использовать различные ресурсы - динамически выделенная память, файлы, сетевые подключения и тд. В этом случае в C++ применяется так называемый принцип/идиома RAII (**r**esource **a**cquisition **i**s **i**nitialization). RAII предполагает, что получение ресурса производится при инициализации объекта. А освобождение ресурса производится в деструкторе объекта.

Пример, использующий RAII:

```
#include <iostream>

class IntArray{
public:
  IntArray(unsigned size) : data{ new int[size] } {} // выделяем память в конструкторе
  ~IntArray(){
    if(data){
      std::cout<<"Freeing memory..."<< std::endl;
      delete[] data; // освобождаем память в диструкторе
    }
  }

  // Удаляем конструктор копирования и оператор присваивания
  IntArray(const IntArray&) = delete;
  IntArray& operator=(const IntArray&) = delete;

  // оператор индексирования для доступа к элементам
  int& operator[](unsigned index) { return data[index]; }

  // возвращаем инкапсулированный ресурс
  int* get() const { return data; }

  // передаем ресурс другогому объекту
  int* release(){
    int* result = data;
    data = nullptr;
    return result;
  }

private:
  int* data;
};

int main(){
  const unsigned count {5}; // количество элементов
  IntArray values{count}; // создаем объект, который управляет ресурсом

  // изменяем элементы динамического массива
```

```
  for (unsigned i {}; i < count; ++i){
    values[i] = i;
  }

  // выводим элементы динамического массива на консоль`
  for (unsigned i {}; i < count; ++i){
    std::cout<<values[i]<<"\t";
  }

  std::cout<<std::endl;
}
```

При этом важно, чтобы ресурс (в данном случае динамическая память) освобождался только один раз. Для этой цели в классе удалены конструктор копирования и оператор присваивания, что позволяет избежать ситуации, когда два объекта хранят указатель на одну и ту же область динамической памяти и соответственно потом в деструкторе будут пытаться освободить эту память.

```
IntArray(const IntArray&) = delete;
IntArray& operator=(const IntArray&) = delete;
```

Применение кода, написанного выше:

```
int main(){
  const unsigned count {5}; // количество элементов
  IntArray array{count}; // создаем объект, который управляет ресурсом

  // изменяем элементы динамического массива
  for(unsigned i {}; i < count; ++i){
    array[i] = i;
  }

  // получаем указатель
  int* data = array.release(); // теперь функция  main  обязана освободить память
  for(unsigned i {}; i < count; ++i){
    std::cout<<data[i]<<"\t";
  }

  std::cout << std::endl;

  //освобождаем память
  delete[] data;
}
```