Move Operators

In cases where an object of a class with resources will no longer be referred to once it has been copied or assigned, we may move that object's resources by a simple copying of their addresses.  This copying is an efficient alternative to copying the members of the to-be-moved object to new locations in the current object.

The C++ move facilities implement this alternative solution for objects that are near the end of their lifetime.

The prototype for a move-constructor takes the form

*Class-name*(*Class-name&&*);

The prototype for a move-assignment operator takes the form

*Class-name&* operator=(*Class-name&&*);

These special member functions receive rvalue references to the source object (the object that is to be moved) and perform the swapping of addresses between the objects' resources.

For example, these operators are defined below for the copy\_assign.cpp program listed above.

|  |  |
| --- | --- |
| // Copy and Move  // copy\_move.cpp  #include <iostream>  #include <utility>  class Array {  int\* a = nullptr;  unsigned n = 0u;  int dummy = 0;  public:  Array(){}  Array(unsigned no) : a(new int[no]), n(no){}  Array(const Array& src) { \*this = src; }  Array(Array&& src) { \*this = std::move(src); }  Array& operator=(const Array& src) {  if (this != &src) {  delete [] a;  a = new int[src.n];  for (unsigned i = 0u; i < src.n; ++i)  a[i] = src.a[i];  n = src.n;  }  return \*this;  }  Array& operator=(Array&& src) {  if (this != &src) {  delete [] a; // deallocate current resource  a = src.a; // copy address to current object  src.a = nullptr; // initialize source resource  // swap instance variables  n = src.n;  dummy = src.dummy;  src.n = 0u;  src.dummy = 0;  }  return \*this;  }  ~Array() { delete [] a; }  int& operator[](unsigned i) {  return n > 0u && i < n ? a[i] : dummy; }  int operator[](unsigned i) const {  return n > 0u && i < n ? a[i] : dummy; }  unsigned size() const { return n; }  };  int main() {  const unsigned size = 5;  Array a(size), b;  for (unsigned i = 0u; i < a.size(); ++i)  a[i] = 3 \* i;  std::cout << "Copy-Assignment\n";  std::cout << "a : ";  for (unsigned i = 0u; i < a.size(); ++i)  std::cout << a[i] << ' ';  std::cout << std::endl;  b = a; // calls copy-assignment  std::cout << "b : ";  for (unsigned i = 0u; i < b.size(); ++i)  std::cout << b[i] << ' ';  std::cout << std::endl;  std::cout << "a : ";  for (unsigned i = 0u; i < a.size(); ++i)  std::cout << a[i] << ' ';  std::cout << std::endl;  std::cout << "Move-Assignment\n";  std::cout << "a : ";  for (unsigned i = 0u; i < a.size(); ++i)  std::cout << a[i] << ' ';  std::cout << std::endl;  b = std::move(a); // calls move-assignment  std::cout << "b : ";  for (unsigned i = 0u; i < b.size(); ++i)  std::cout << b[i] << ' ';  std::cout << std::endl;  std::cout << "a : ";  for (unsigned i = 0u; i < a.size(); ++i)  std::cout << a[i] << ' ';  std::cout << std::endl;  } | Copy-Assignment  a : 0 3 6 9 12  b : 0 3 6 9 12  a : 0 3 6 9 12  Move-Assignment  a : 0 3 6 9 12  b : 0 3 6 9 12  a : |