|  |  |  |
| --- | --- | --- |
| **American University of Sharjah**  **College of Engineering**  Dept of Computer Science & Engg  P. O. Box 26666  Sharjah, UAE | A picture containing logo  Description automatically generated | **Instructors:** Dr. Aliaa Moualla  **Lab Instructor:** Sameer Alawnah  **Office:** EB1-0012C  **Phone**: 971-6-515-4940  **e-mail**: salawnah@aus.edu  **Semester**: Spring 2024 |

**CMP 220L – Programming II**

**Lab 9: The Big-5**

**Question 1:**

Given the following incomplete code that used the Book class from the previous lab, implement all the ToDo items to give the expected output:

#include<iostream>

#include<string>

using namespace std;

class Book {

private:

string title, author;

public:

Book() {title=author="None";}

Book(string \_title, string \_author) : title(\_title), author(\_author) {}

std::ostream& operator<<(std::ostream& os) const;

std::istream& operator>>(std::istream& is) ;

~Book() {}

};

std::istream& Book::operator>>(std::istream& is)

{

getline(is, title);

getline(is, author);

return is;

}

std::ostream& Book::operator<<(std::ostream& os) const

{

os << "Title: " << title << ", Author: " << author;

return os;

}

// ToDo #1: Create a function printBook that take a const L-Value reference for a book b, this function should print the following line:

// "Printing using L-Value Book Reference"

// then print the book itself

// ToDo #2: Create a function printBook that take a const R-Value reference for a book b, this function should print the following line:

// "Printing using R-Value Book Reference"

// then print the book itself

// ToDo #3: Create a function createBook that take as input the title and author of the book, this function should create and return a book with the given title and author .

int main()

{

Book lvalue("Book\_Lvalue","Author\_Lvalue");

printBook(lvalue);

printBook(Book("Book\_Rvalue","Book\_Rvalue"));

printBook(createBook("Book\_from\_Function","Author\_from\_Function"));

return 0;

}

The expected output is the following:

**Printing using L-Value Book Reference**

**Title: Book\_Lvalue, Author: Author\_Lvalue**

**Printing using R-Value Book Reference**

**Title: Book\_Rvalue, Author: Book\_Rvalue**

**Printing using R-Value Book Reference**

**Title: Book\_from\_Function, Author: Author\_from\_Function**

#include<iostream>

#include<string>

**using** **namespace** std;

**class** Book {

**private**:

string title, author;

**public**:

Book() {title=author="None";}

Book(string \_title, string \_author) : title(\_title), author(\_author) {}

std::ostream& **operator**<<(std::ostream& os) **const**;

std::istream& **operator**>>(std::istream& is) ;

~Book() {}

};

std::istream& Book::**operator**>>(std::istream& is)

{

getline(is, title);

getline(is, author);

**return** is;

}

std::ostream& Book::**operator**<<(std::ostream& os) **const**

{

os << "Title: " << title << ", Author: " << author;

**return** os;

}

**void** printBook(**const** Book& c){

cout <<"\nPrinting using L-value Book Reference " << endl;

c.**operator**<<(cout);

}

**void** printBook(**const** Book&& c){

cout<<"\nPrinting using R-value Book Reference " << endl;

c.**operator**<<(cout);

}

Book createBook(string title, string author)

{

**return** Book(title, author);

}

**int** main()

{

Book lvalue("Book\_Lvalue","Author\_Lvalue");

printBook(lvalue);

printBook(Book("Book\_Rvalue","Book\_Rvalue"));

printBook(createBook("Book\_from\_Function","Author\_from\_Function"));

**return** 0;

}

A black text on a white background

Description automatically generated

**Question 2:**

Given the following working code that uses the Book and Library classes from the previous lab.

#include<iostream>

#include<string>

#include<vector>

using namespace std;

class Book {

private:

string title, author;

public:

Book() {title=author="None";}

Book(string \_title, string \_author) : title(\_title), author(\_author) {}

std::ostream& operator<<(std::ostream& os) const;

std::istream& operator>>(std::istream& is) ;

~Book() {}

};

std::istream& Book::operator>>(std::istream& is)

{

getline(is, title);

getline(is, author);

return is;

}

std::ostream& Book::operator<<(std::ostream& os) const

{

os << "Title: " << title << ", Author: " << author;

return os;

}

std::ostream& operator<<(std::ostream& os, const Book& book)

{

return book.operator<<(os);

}

std::istream& operator>>(std::istream& is, Book& book)

{

return book.operator>>(is);

}

class Library {

private:

string name;

int numBooks;

int capacity;

Book\* books;

public:

Library() {

name = "None";

capacity = 5;

books = new Book[capacity];

numBooks = 0;

}

Library(string \_name, int \_capacity) : name(\_name), capacity(\_capacity) {

numBooks = 0;

books = new Book[capacity];

}

Library(const Library& obj) {

name = obj.name;

capacity = obj.capacity;

books = new Book[capacity];

numBooks = obj.numBooks;

for (int i = 0; i < numBooks; i++) {

books[i] = obj.books[i];

}

cout<<"Copy const. is called\n";

}

Library& operator=(const Library& obj) {

if (this != &obj)

{

delete[] books; //delete should contain suqare barckets

name = obj.name;

numBooks = obj.numBooks;

capacity = obj.capacity;

books = new Book[capacity];

for (int i = 0; i < numBooks; i++) {

books[i] = obj.books[i];

}

}

cout<<"= operator is called\n";

return \*this;

}

void addBook(const Book& obj)

{

if (numBooks < capacity)

{

books[numBooks] = obj;

numBooks++;

}

else

{

cout << "The capacity is full!" << endl;

}

}

~Library() { if(books!=nullptr) delete[] books; }//delete should contain suqare barckets

std::ostream& operator<<(std::ostream& os) const {

os << "Library Name: " << name << endl;

os << "Books in Library: \n";

for (int i = 0; i < numBooks; i++) {

os<<books[i]<<endl;

}

return os;

}

};

std::ostream& operator<<(std::ostream& os, const Library& lib)

{

return lib.operator<<(os);

}

Library createLibrary(int cap, string name)

{

Library res(name,cap);

for (int i = 0; i < cap; i++)

{

res.addBook(Book("Title "+to\_string(i),"Author "+to\_string(i)));

}

return res;

}

int main()

{

Library lib1 = createLibrary(3, "My Library"); // Testing the copy/move constructor

Library lib2 = lib1; // Testing the copy constrcutor again

Library lib3;

lib3 = lib2; // Testing the = operator

cout << "Library 1 : "<<lib1<<endl;

cout << "Library 2 : "<<lib2<<endl;

cout << "Library 3 : "<<lib3<<endl;

cout << "Library 4 : "<<createLibrary(1,"My Other Library")<<endl; // Will copy/move const. called ?

lib3 = createLibrary(2, "My Third Library"); // Test move = operator

cout << "Library 3 : "<<lib3<<endl;

vector<Library> libs;

libs.push\_back(createLibrary(2, "My Forth Library")); // Testing the move copy const.

cout << "Library 5: "<<libs[0]<<endl;

return 0;

}

Sample output is shown below (Note this output is generated by the g++ compiler, you will get a different number of calls of the Copy const. and = operators when using VS++):

**Copy const. is called**

**= operator is called**

**Library 1 : Library Name: My Library**

**Books in Library:**

**Title: Title 0, Author: Author 0**

**Title: Title 1, Author: Author 1**

**Title: Title 2, Author: Author 2**

**Library 2 : Library Name: My Library**

**Books in Library:**

**Title: Title 0, Author: Author 0**

**Title: Title 1, Author: Author 1**

**Title: Title 2, Author: Author 2**

**Library 3 : Library Name: My Library**

**Books in Library:**

**Title: Title 0, Author: Author 0**

**Title: Title 1, Author: Author 1**

**Title: Title 2, Author: Author 2**

**Library 4 : Library Name: My Other Library**

**Books in Library:**

**Title: Title 0, Author: Author 0**

**= operator is called**

**Library 3 : Library Name: My Third Library**

**Books in Library:**

**Title: Title 0, Author: Author 0**

**Title: Title 1, Author: Author 1**

**Copy const. is called**

**Library 5: Library Name: My Forth Library**

**Books in Library:**

**Title: Title 0, Author: Author 0**

**Title: Title 1, Author: Author 1**

In many cases, an object is created and immediately deleted after being used. For example, in the command:

libs.push\_back(createLibrary(2, "My Forth Library"));, createLibrary will create a library, and pass it to the push\_back function to insert a copy of it into the vector. The object created by the createLibrary will then be deleted. In this case, it is more efficient to transfer the object content to the library without carrying out an actual memory copy. This operation can be accomplished by using the move semantics through the move constructor. The same logic can be used to implement the move assignment operation.

Add the move constructor and the move = operator to the class and test them with the same main. Don’t forget to print “move const. is called” and “move = operator is called” in the respective function.

Some of the calls to the copy constructor should be replaced by the move constructors, the same applies to the call of the = operators.

#include<iostream>

#include<string>

#include<vector>

**using** **namespace** std;

**class** Book {

**private**:

string title, author;

**public**:

Book() { title = author = "None"; }

Book(string \_title, string \_author) : title(\_title), author(\_author) {}

std::ostream& **operator**<<(std::ostream& os) **const**;

std::istream& **operator**>>(std::istream& is);

~Book() {}

};

std::istream& Book::**operator**>>(std::istream& is)

{

getline(is, title);

getline(is, author);

**return** is;

}

std::ostream& Book::**operator**<<(std::ostream& os) **const**

{

os << "Title: " << title << ", Author: " << author;

**return** os;

}

std::ostream& **operator**<<(std::ostream& os, **const** Book& book)

{

**return** book.**operator**<<(os);

}

std::istream& **operator**>>(std::istream& is, Book& book)

{

**return** book.**operator**>>(is);

}

**class** Library {

**private**:

string name;

**int** numBooks;

**int** capacity;

Book\* books;

**public**:

Library() {

name = "None";

capacity = 5;

books = **new** Book[capacity];

numBooks = 0;

}

Library(string \_name, **int** \_capacity) : name(\_name), capacity(\_capacity) {

numBooks = 0;

books = **new** Book[capacity];

}

Library(**const** Library& obj) {

name = obj.name;

capacity = obj.capacity;

books = **new** Book[capacity];

numBooks = obj.numBooks;

**for** (**int** i = 0; i < numBooks; i++) {

books[i] = obj.books[i];

}

cout << "Copy const. is called\n";

}

Library& **operator**=(**const** Library& obj) {

**if** (**this** != &obj)

{

**delete**[] books; //delete should contain suqare barckets

name = obj.name;

numBooks = obj.numBooks;

capacity = obj.capacity;

books = **new** Book[capacity];

**for** (**int** i = 0; i < numBooks; i++) {

books[i] = obj.books[i];

}

}

cout << "= operator is called\n";

**return** \***this**;

}

**void** addBook(**const** Book& obj)

{

**if** (numBooks < capacity)

{

books[numBooks] = obj;

numBooks++;

}

**else**

{

cout << "The capacity is full!" << endl;

}

}

~Library() { **if** (books != **nullptr**) **delete**[] books; }//delete should contain suqare barckets

std::ostream& **operator**<<(std::ostream& os) **const** {

os << "Library Name: " << name << endl;

os << "Books in Library: \n";

**for** (**int** i = 0; i < numBooks; i++) {

os << books[i] << endl;

}

**return** os;

}

Library(Library&& obj) {

name = obj.name;

capacity = obj.capacity;

numBooks = obj.numBooks;

books = obj.books;

obj.books = **nullptr**;

cout << "Move Copy const. is called\n";

}

Library& **operator**=(Library&& obj) {

**if** (**this** != &obj)

{

**delete**[] books; //delete should contain suqare barckets

name = obj.name;

numBooks = obj.numBooks;

capacity = obj.capacity;

books = obj.books;

obj.books = **nullptr**;

}

cout << "= operator is called\n";

**return** \***this**;

}

};

std::ostream& **operator**<<(std::ostream& os, **const** Library& lib)

{

**return** lib.**operator**<<(os);

}

Library createLibrary(**int** cap, string name)

{

Library res(name, cap);

**for** (**int** i = 0; i < cap; i++)

{

res.addBook(Book("Title " + to\_string(i), "Author " + to\_string(i)));

}

**return** res;

}

**int** main()

{

Library lib1 = createLibrary(3, "My Library"); // Testing the copy/move constructor

Library lib2 = lib1; // Testing the copy constrcutor again

Library lib3;

lib3 = lib2; // Testing the = operator

cout << "Library 1 : " << lib1 << endl;

cout << "Library 2 : " << lib2 << endl;

cout << "Library 3 : " << lib3 << endl;

cout << "Library 4 : " << createLibrary(1, "My Other Library") << endl; // Will copy/move const. called ?

lib3 = createLibrary(2, "My Third Library"); // Test move = operator

cout << "Library 3 : " << lib3 << endl;

vector<Library> libs;

libs.push\_back(createLibrary(2, "My Forth Library")); // Testing the move copy const.

cout << "Library 5: " << libs[0] << endl;

**return** 0;

}

Good Luck ☺