```
#include <iomanip>
#include <iostream>
#include "IntegerBuffer.h"
using namespace std;


IntegerBuffer::IntegerBuffer(int dataCapacity) {
    this->dataLength = 0;
    try {
        this->data = new int[dataCapacity];
        this->dataCapacity = dataCapacity;
    }
    catch (const bad_alloc& e) {
        this->data = nullptr;
        this->dataCapacity = 0;
        cout << "Allocation of memory failed";
    }
}

IntegerBuffer::IntegerBuffer() {
    this->dataLength = 0;
    try {
        this->data = new int[dataCapacity];
        this->dataCapacity = 32;
    }
    catch (const bad_alloc& e) {
        this->data = nullptr;
        this->dataCapacity = 0;
        cout << "Allocation of memory failed";
    }
}

IntegerBuffer::~IntegerBuffer() {
    dataCapacity = 0;
    dataLength = 0;
    if (data != nullptr) {
        int* memory = data;
        data = nullptr;
        delete[] memory;
    }
}

IntegerBuffer::IntegerBuffer(const IntegerBuffer& other) :
IntegerBuffer(other.dataCapacity) {
    copy(other);
}

IntegerBuffer* IntegerBuffer::clone() {
    //Cant call copy function here, needs to create a new instance of
IntegerBuffer?
    return new IntegerBuffer(*this);
}

void IntegerBuffer::clear() {
    //Setting dataLength to zero will work because there will be no attempt to read
from the buffer
    dataLength = 0;
}
```

```
int IntegerBuffer::add(int value) {
    if (dataLength < dataCapacity) {
        data[dataLength] = value;
        ++dataLength;
        return 1;
    }
    else {
        return 0;
    }
}

int IntegerBuffer::add(const int* array, int arrayLength) {
    int count = 0;
    for (int i = 0; i < arrayLength; ++i) {
        count += add(array[i]);
    }
    return count;
}

int IntegerBuffer::removeFast(int index) {
    if (index >= 0 && index < dataLength) {
        --dataLength;
        data[index] = data[dataLength];
        return 1;
    }
    else {
        return 0;
    }
}

int IntegerBuffer::removeStable(int index) {
    if (index >= 0 && index < dataLength) {
        --dataLength;
        for (int i = index + 1; i <= dataLength; ++i) {
            data[i - 1] = data[i];
        }
        return 1;
    }
    else {
        return 0;
    }
}

int IntegerBuffer::index(int value) const {
    for (int i = 0; i < dataLength; ++i) {
        if (data[i] == value) {
            return i;
        }
    }
    return -1;
}

int IntegerBuffer::rindex(int value) const {
    for (int i = dataLength - 1; i >= 0; --i) {
        if (data[i] == value) {
            return i;
        }
    }
    return -1;
```

```
}
//I want this to take column width and number of colums, but the test code shows no
inputs so I preset the numbers
void IntegerBuffer::print() const {
    for (int i = 0; i < dataLength; ++i) {
        if (i > 0 && i % 10 == 0) {
            cout << "\n";
        }
        cout << setw(4) << data[i];
    }
    cout << endl;
}

int IntegerBuffer::copy(const IntegerBuffer& other) {
    if (&other == this) {
        return 0;
    }
    else {
        this->clear();
        return this->add(other.data, other.dataLength);
    }
}

int IntegerBuffer::copy(const int* arr, int arrLength) {
    this->clear();
    return this->add(arr, arrLength);
}

bool IntegerBuffer::compare(const IntegerBuffer& other) const {
    int count = 0;
    for (int i = 0; i < this->dataLength; ++i) {
        if (this->data[i] == other.data[i])
            ++count;
    }
    if (this->dataLength == other.dataLength && count == this->dataLength) {
        return true;
    }
    else {
        return false;
    }
}
```