```cpp
// Lecture 48: Copy Constructor I
#include <iostream>
class Integer {
    int *m_pInt; // pointer as member
public:
    Integer(); // default
    Integer(int value); //parameterised
    //Integer(const Integer &obj);
    int GetValue()const; // gets
    void SetValue(int value); // set
    ~Integer(); // destructor - free the memory allocated for the integer pointer.

};

// Implementation
Integer::Integer() {
    std::cout << "Integer()" << std::endl;
    m_pInt = new int(0); // default
}

Integer::Integer(int value) {
    std::cout << "Integer(int)" << std::endl;
    m_pInt = new int(value); /
}

Integer::Integer(const Integer & obj) {
    std::cout << "Integer(const Integer&)" << std::endl;
    m_pInt = new int(*obj.m_pInt);
}


int Integer::GetValue() const {
    return *m_pInt;
}

void Integer::SetValue(int value) {
    *m_pInt = value;
}

Integer::~Integer() {
    std::cout << "~Integer()" << std::endl;
    delete m_pInt;
}
// Case 2: Copy of the object is created because we are passing by value.
void Print(Integer i){}

// Case 3: copy of the object is created because we are returning by value.
Integer Add(int x, int y){ return Integer(x+y);}
// Driver code
int main(void)
{
    Integer i (5); // creates an integer i

    // Case 1: Invoking copy constructor directly
    Integer i2(i);// this would cause the constructor to synthesize a copy constructor for our class even though we have not created it. This is kust 1 case, the other case is - assume this function that prints the integer that prints an integer or a function that adds two integers.
}
// This seems ok but if you run this code - it crashes and it crashes in some library function.

```