Module 3: Critical Thinking Assignment

Professor H. Pensado, CSC450-1 Programming III, Colorado State University – Global Campus

Integers Pointers Program

Hugo Martinez

February 27, 2026

Integers Pointers Program – C++

In this assignment, I will demonstrate my understanding of integer pointers in C++ and how dynamic memory allocation works. The goal of this program is to collect three integer values from the user, store those values in regular variables, and then use pointers to dynamically allocate memory and manage those values separately on the heap.

Throughout this paper, I will explain how the program works step by step, including how memory is allocated using the new operator and properly released using the delete operator. I will also show how to display both the values and memory addresses of variables and pointers in order to better understand the relationship between stack memory and heap memory.

Pseudocode

The purpose of this program is to collect three integer values from the user and store them in regular variables. After that, the program dynamically allocates memory on the heap using integer pointers and copies those values into the allocated memory.

The pseudocode focuses on the overall structure and flow of the program rather than specific syntax. It outlines how user input is handled, how memory is allocated using the new operator, how values are accessed through dereferencing, and how dynamically allocated memory is properly released using the delete operator. This step helps ensure the logic is clear before implementing the final code.

```
BEGIN Program

    Declare integer variables:
        num1, num2, num3

    Prompt user to enter first integer
    Read value into num1

    Prompt user to enter second integer
    Read value into num2

    Prompt user to enter third integer
    Read value into num3

    Declare integer pointers:
        ptr1, ptr2, ptr3

    Allocate dynamic memory for each pointer:
        ptr1 = new integer
        ptr2 = new integer
        ptr3 = new integer

    Store values from variables into dynamic memory:
        *ptr1 = num1
        *ptr2 = num2
        *ptr3 = num3

    Display for each number:
        Value of variable (numX)
        Address of variable (&numX)
        Address stored in pointer (ptrX)
        Value stored at pointer (*ptrX)

    Deallocate dynamic memory:
        delete ptr1
        delete ptr2
        delete ptr3

    Set pointers to nullptr

END Program
```

Source Code

The program begins by including the <iostream> library, which allows for input and output using cin and cout. The main() function contains all of the program's logic.

First, three integer variables (num1, num2, and num3) are declared. These variables are stored in stack memory and are used to collect input from the user. The program prompts the user to enter three integer values, which are read using cin and stored in the respective variables.

Next, three integer pointers (ptr1, ptr2, and ptr3) are declared. These pointers will be used to manage dynamically allocated memory on the heap. Using the new operator, memory is allocated for each pointer. At this point, the pointers store the memory addresses of newly allocated integer locations on the heap.

The values entered by the user are then copied into the dynamically allocated memory by dereferencing each pointer. For example, *ptr1 = num1; assigns the value stored in num1 to the memory location pointed to by ptr1.

The program then displays detailed information for both the variables and the pointers. For the variables, it prints the stored value and the memory address using the address-of operator (&). For the pointers, it prints the memory address stored in the pointer and the value at that address using the dereference operator (*). This demonstrates the difference between stack memory and heap memory.

Finally, the program properly releases the dynamically allocated memory using the delete operator for each pointer. After deletion, each pointer is set to nullptr to prevent it from referencing invalid memory. This step is important for avoiding dangling pointers and ensuring responsible memory management.

```
#include <iostream>


using namespace std;


int main()

{

    // Declare three integer variables (stack memory)

    int num1, num2, num3;


    // Prompt user for input

    cout << "Enter the first integer: ";

    cin >> num1;
```

```
cout << "Enter the second integer: ";
cin >> num2;

cout << "Enter the third integer: ";
cin >> num3;

// Declare three integer pointers
int* ptr1;
int* ptr2;
int* ptr3;

// Dynamically allocate memory (heap memory)
ptr1 = new int;
ptr2 = new int;
ptr3 = new int;

// Store variable values into dynamically allocated memory
*ptr1 = num1;
*ptr2 = num2;
*ptr3 = num3;

// Display values and addresses
cout << "\n--- Variable Information ---" << endl;
cout << "num1 value: " << num1 << endl;
cout << "num1 address: " << &num1 << endl;

cout << "\nnum2 value: " << num2 << endl;
cout << "num2 address: " << &num2 << endl;

cout << "\nnum3 value: " << num3 << endl;
cout << "num3 address: " << &num3 << endl;
```

```
    cout << "\n--- Pointer Information ---" << endl;

    cout << "ptr1 address stored: " << ptr1 << endl;

    cout << "ptr1 dereferenced value: " << *ptr1 << endl;


    cout << "\nptr2 address stored: " << ptr2 << endl;

    cout << "ptr2 dereferenced value: " << *ptr2 << endl;


    cout << "\nptr3 address stored: " << ptr3 << endl;

    cout << "ptr3 dereferenced value: " << *ptr3 << endl;


    // Free dynamically allocated memory

    delete ptr1;

    delete ptr2;

    delete ptr3;


    // Set pointers to nullptr to avoid dangling pointers

    ptr1 = nullptr;

    ptr2 = nullptr;

    ptr3 = nullptr;


    return 0;

}
```

Program Analysis

Working with integer pointers and dynamic memory allocation introduces additional responsibility compared to using regular variables alone. While the program functions correctly, there are several potential risks that must be considered when using pointers.

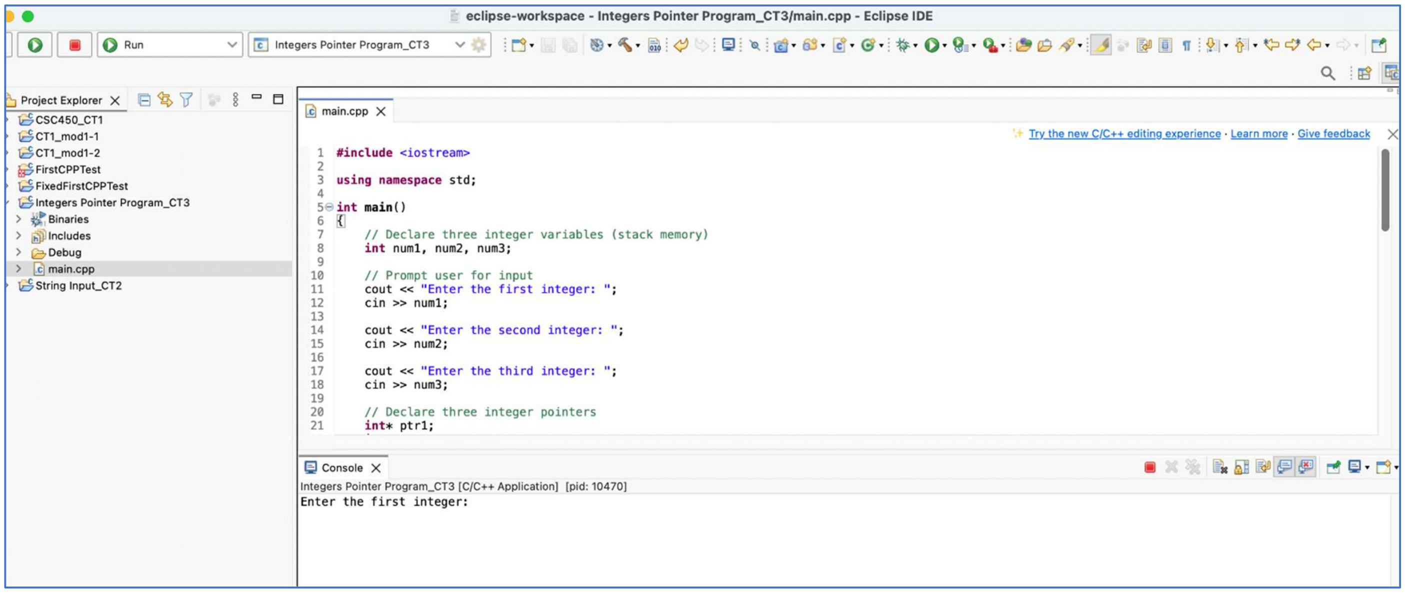
One of the primary concerns is a memory leak. A memory leak occurs when dynamically allocated memory is not properly released using the delete operator. If the program allocates memory with new but never calls delete, that memory remains reserved even after it is no longer needed. Over time, repeated memory leaks in larger programs can reduce available system memory and negatively impact performance.

Another potential issue is a dangling pointer. A dangling pointer occurs when a pointer continues to reference a memory location that has already been freed. Attempting to dereference such a pointer can lead to undefined behavior, program crashes, or corrupted data. In this program, setting each pointer to nullptr after using delete helps reduce this risk by ensuring the pointer no longer references invalid memory.

There is also the possibility of dereferencing a null or uninitialized pointer. If a pointer is not properly initialized before being dereferenced, the program may attempt to access an invalid memory address, which can result in a segmentation fault. In this implementation, memory is allocated before any dereferencing occurs, which prevents that issue.

Compared to regular stack variables, dynamically allocated memory requires manual management. Stack variables are automatically cleaned up when they go out of scope, but heap memory must be explicitly released. This difference highlights why careful memory management is essential when working with pointers in C++.

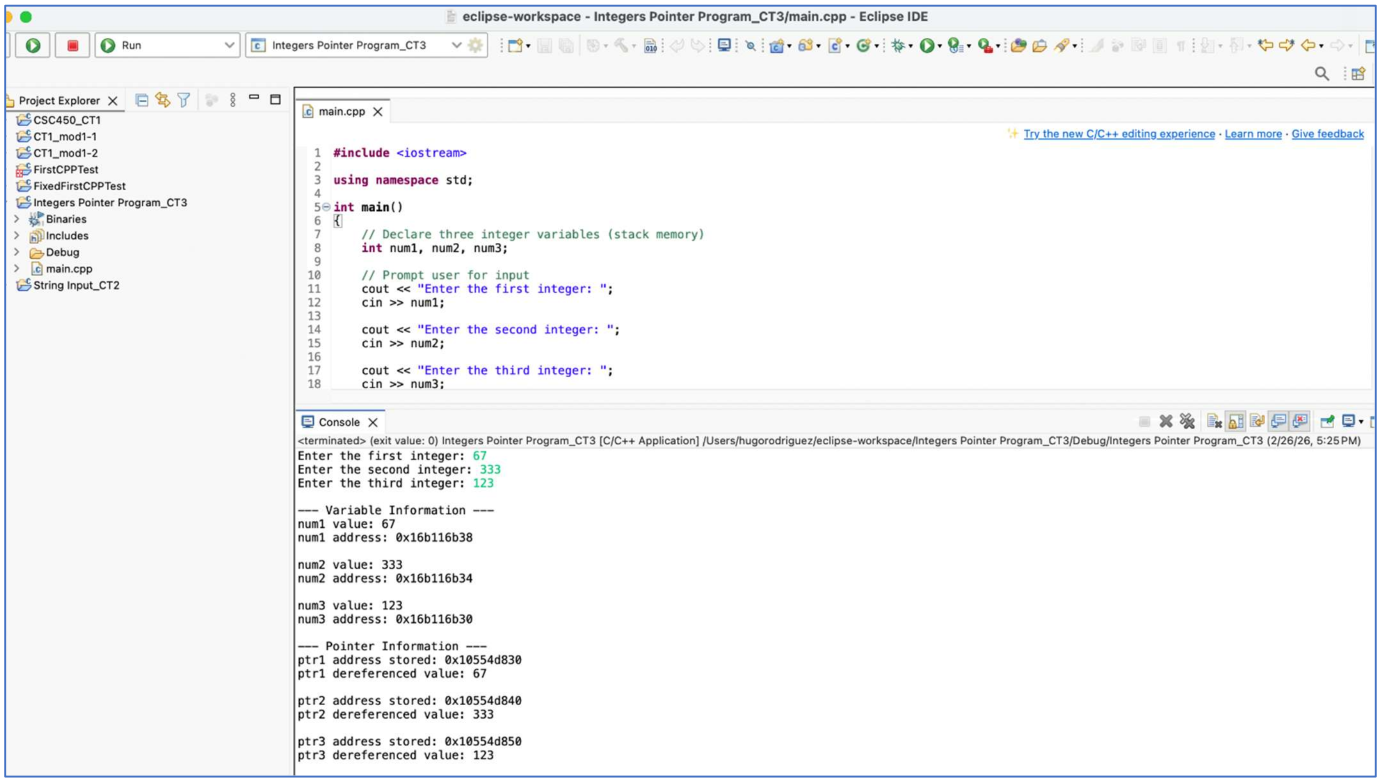
Program Execution


eclipse-workspace - Integers Pointer Program_CT3/main.cpp - Eclipse IDE
Run
Integers Pointer Program_CT3
Project Explorer
CSC450_CT1
CT1_mod1-1
CT1_mod1-2
FirstCPPTest
FixedFirstCPPTest
Integers Pointer Program_CT3
Binaries
Includes
Debug
main.cpp
String Input_CT2
main.cpp
Try the new C/C++ editing experience · Learn more · Give feedback
Console

```cpp
#include <iostream>

using namespace std;

int main()
{
    // Declare three integer variables (stack memory)
    int num1, num2, num3;

    // Prompt user for input
    cout << "Enter the first integer: ";
    cin >> num1;

    cout << "Enter the second integer: ";
    cin >> num2;

    cout << "Enter the third integer: ";
    cin >> num3;

    // Declare three integer pointers
    int* ptr1;
```

```
Integers Pointer Program_CT3 [C/C++ Application]  [pid: 10470]
Enter the first integer:
```

Figure 1. Program output showing the completed main.cpp file for the Integer Pointers Program along with the console window prompting the user to enter integer values.


eclipse-workspace - Integers Pointer Program_CT3/main.cpp - Eclipse IDE
Run
Integers Pointer Program_CT3
Project Explorer
CSC450_CT1
CT1_mod1-1
CT1_mod1-2
FirstCPPTest
FixedFirstCPPTest
Integers Pointer Program_CT3
Binaries
Includes
Debug
main.cpp
String Input_CT2
main.cpp
Try the new C/C++ editing experience · Learn more · Give feedback
Console

```cpp
#include <iostream>

using namespace std;

int main()
{
    // Declare three integer variables (stack memory)
    int num1, num2, num3;

    // Prompt user for input
    cout << "Enter the first integer: ";
    cin >> num1;

    cout << "Enter the second integer: ";
    cin >> num2;

    cout << "Enter the third integer: ";
    cin >> num3;
```

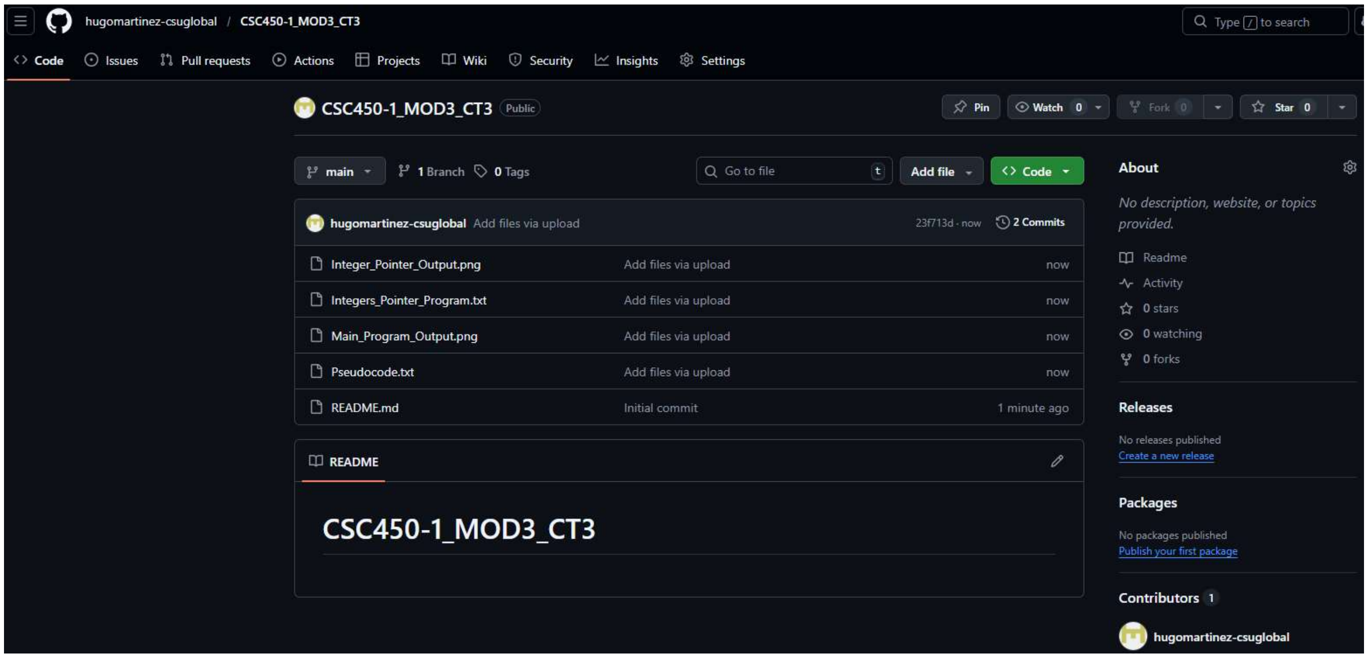
```
<terminated> (exit value: 0) Integers Pointer Program_CT3 [C/C++ Application] /Users/hugorodriguez/eclipse-workspace/Integers Pointer Program_CT3/Debug/Integers Pointer Program_CT3 (2/26/26, 5:25 PM)
Enter the first integer: 67
Enter the second integer: 333
Enter the third integer: 123

--- Variable Information ---
num1 value: 67
num1 address: 0x16b116b38

num2 value: 333
num2 address: 0x16b116b34

num3 value: 123
num3 address: 0x16b116b30

--- Pointer Information ---
ptr1 address stored: 0x10554d830
ptr1 dereferenced value: 67

ptr2 address stored: 0x10554d840
ptr2 dereferenced value: 333

ptr3 address stored: 0x10554d850
ptr3 dereferenced value: 123
```

Figure 2. Program execution results showing the three integer inputs along with their corresponding stack and heap memory addresses and dereferenced pointer values.

GitHub Repository

Submission details can be found at the following GitHub Repository link:

https://github.com/hugomartinez-csuglobal/CSC450-1_MOD3_CT3



hugomartinez-csuglobal Add files via upload 23f713d · now 2 Commits

Integer_Pointer_Output.png	Add files via upload	now
Integers_Pointer_Program.txt	Add files via upload	now
Main_Program_Output.png	Add files via upload	now
Pseudocode.txt	Add files via upload	now
README.md	Initial commit	1 minute ago

README

CSC450-1_MOD3_CT3

No description, website, or topics provided.

No releases published
Create a new release

No packages published
Publish your first package

Conclusion

This assignment provided a clear demonstration of how integer pointers and dynamic memory allocation work in C++. By collecting user input, allocating memory on the heap using the new operator, and properly releasing that memory with delete, the program illustrates the difference between stack and heap memory in a practical way.

It highlighted the responsibility that comes with using dynamic memory. Unlike stack variables, heap memory must be manually managed. Failing to release memory or improperly handling pointers can result in memory leaks, dangling pointers, or program instability. By carefully allocating and deallocating memory and setting pointers to nullptr, this program demonstrates safe and responsible pointer usage.