**Source Code**

#include <iostream>

#include <ostream>

using *namespace* std;

*int* main()

{

    // declare varibales for inputs and pointers

*int* \* ptr\_one;

*int* \* ptr\_two;

*int* \* ptr\_three;

*int* input\_one = 0;

*int* input\_two = 0;

*int* input\_three = 0;    // get integer inputs from user

    cout << "Please enter 3 integer values: ";

    cin >> input\_one >> input\_two >> input\_three;

    // create and assign integer pointers for each variable

    ptr\_one = &input\_one;

    ptr\_two = &input\_two;

    ptr\_three = &input\_three;

// display the contents of the varibles and pointers

    cout << "\nFor the First Input: " << '\n';

    cout << "Value stored at the pointers location: " << \*ptr\_one << '\n';

    cout << "Memory address of pointer: " << ptr\_one << '\n';

    cout << "Value of input variable: "  << input\_one << '\n';

    cout << '\n';

    // second input

    cout << "For the Second Input: " << '\n';

    cout << "Value stored at the pointers location: " << \*ptr\_two << '\n';

    cout << "Memory address of pointer: " << ptr\_two << '\n';

    cout << "Value of input variable: "  << input\_two << '\n';

    cout << '\n';

    // thrid input

    cout << "For the Third Input: " << '\n';

    cout << "Value stored at the pointers location: " << \*ptr\_three << '\n';

    cout << "Memory address of pointer: " << ptr\_three << '\n';

    cout << "Value of input variable: "  << input\_three << '\n';

    cout << '\n';

    // retun main

    return 0;

}

**Screenshots:**

**Text

Description automatically generated**

**Analysis**

A major goal of the software development process is to deliver high quality and secure products on time. One of the most common security concerns with C++ is a buffer overflow. Buffer overflows are caused by not having built-in boundary check features in C++, which reduce the risk of overwriting memory. Writing outside our allocated memory can cause program crashes and corrupt data. It can even lead to the execution of malicious code (Top, 2022). One of the main things I want to focus on while writing c++ programs with pointers is avoiding buffer overflow. Lets look at the program to see what we can find.

Main:

*int* main()

{

    // declare varibales for inputs and pointers

*int* \* ptr\_one;

*int* \* ptr\_two;

*int* \* ptr\_three;

*int* input\_one = 0;

*int* input\_two = 0;

*int* input\_three = 0;    // get integer inputs from user

    cout << "Please enter 3 integer values: ";

    cin >> input\_one >> input\_two >> input\_three;

I see my ptr\_one, ptr\_two, and ptr\_three integer pointers are declared correctly, and input\_one, input\_two, and input\_three are declared correctly and taken as an input. Next let’s look at what’s being done with the input.

ptr\_one = &input\_one;

    ptr\_two = &input\_two;

    ptr\_three = &input\_three;

we are successfully assigning integer pointers for each variable. Let’s proceed to the final step of the program.

// display the contents of the varibles and pointers

    cout << "\nFor the First Input: " << '\n';

    cout << "Value stored at the pointers location: " << \*ptr\_one << '\n';

    cout << "Memory address of pointer: " << ptr\_one << '\n';

    cout << "Value of input variable: "  << input\_one << '\n';

    cout << '\n';

    // second input

    cout << "For the Second Input: " << '\n';

    cout << "Value stored at the pointers location: " << \*ptr\_two << '\n';

    cout << "Memory address of pointer: " << ptr\_two << '\n';

    cout << "Value of input variable: "  << input\_two << '\n';

    cout << '\n';

    // thrid input

    cout << "For the Third Input: " << '\n';

    cout << "Value stored at the pointers location: " << \*ptr\_three << '\n';

    cout << "Memory address of pointer: " << ptr\_three << '\n';

    cout << "Value of input variable: "  << input\_three << '\n';

    cout << '\n';

    // return main

    return 0;

We see all I did was display the contents of the variables and pointers using the cout statement. Again, I don’t see anything wrong with this section of the program.

Finally, I believe this code looks secure, only thing I would add to improve functionality and security of the code are exception handling and the use of functions. Fortunately, in this code if a user inputs strings instead of integers then the program won’t crash it simply won’t register as an int and it will leave inputs with the value of zero. But, if the program expects a certain data type but another is used without safety measures in play then we could run into potential issues.

**References**

Top 5 C++ security risks. (2022, August 16). Snyk. <https://snyk.io/blog/top-5-c-security-risks/>