Pointers

1. Pointer Basics 1

Variables are stored in distinct memory locations. Each variable is allocated a section of memory large enough to hold a value of that type. Some common types and the amount of memory space allocated are shown next.

<table>
<thead>
<tr>
<th>Type</th>
<th>Memory Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>short</td>
<td>2 bytes</td>
</tr>
<tr>
<td>int</td>
<td>4 bytes</td>
</tr>
<tr>
<td>long</td>
<td>4 bytes</td>
</tr>
<tr>
<td>float</td>
<td>4 bytes</td>
</tr>
<tr>
<td>double</td>
<td>8 bytes</td>
</tr>
</tbody>
</table>

Each byte of memory has a unique address. A variable’s address is the address of the first byte allocated to that variable. When the address operator (&) is placed in front of a variable name (such as &amount), it returns the address of that variable.

The following program shows the use of the address operator & to display the address of a variable, the sizeof function to display the size of the variable, and the actual content of the variable.

```cpp
#include <iostream>
using namespace std;

void main (int argc, char * const argv[]) {
    int x = 1;
    short y = 2;
    int z = 3;

    cout << "The value of x is " << x << endl;
    cout << "The value of y is " << y << endl;
    cout << "The value of z is " << z << endl;

    cout << "The size of x is " << sizeof(x) << " bytes" << endl;
    cout << "The size of y is " << sizeof(y) << " bytes" << endl;
    cout << "The size of z is " << sizeof(z) << " bytes" << endl;

    cout << "The address of x is " << &x << endl;
    cout << "The address of y is " << &y << endl;
    cout << "The address of z is " << &z << endl;

    system("pause");
}
```
Sample output:

The value of x is 1
The value of y is 2
The value of z is 3
The size of x is 4 bytes
The size of y is 2 bytes
The size of z is 4 bytes
The address of x is 0029FB50
The address of y is 0029FB44
The address of z is 0029FB38
2. **Pointer Basics 2**

Variables for storing memory addresses are called pointers.

```cpp
#include <iostream>
using namespace std;

void main (int argc, char * const argv[]) {
    int x = 25;
    int *ptr; // declare a pointer called ptr
    ptr = &x; // store the address of x into ptr
    cout << "The value of x is " << x << endl;
    cout << "The address of x is " << ptr << endl;
    system("pause");
}
```

Sample output:

The value in x is 25
The address of x is 0016F72C
3. Pointer Basics 3

The indirection operator * is used to access the content of the memory location that the pointer is pointing to. So if ptr points to a memory location, then *ptr accesses the content of that location.

```cpp
// This program demonstrates the use of the indirection operator *
#include <iostream>
using namespace std;

void main (int argc, char * const argv[]) {
    int x = 25;
    int *ptr; // declare a pointer called ptr
    ptr = &x; // store the address of x in ptr

    cout << "The value in x printed twice: " << x << " and " << *ptr << endl;
    *ptr = 73; // store 73 in x

    cout << "The value in x printed twice after assigning to *ptr: " << x << " and " << *ptr << endl;
    system("pause");
}
```

Sample output:

The value in x printed twice: 25 and 25

The value in x printed twice after assigning to *ptr: 73 and 73
4. Pointer Basics 4

A pointer can point to different variables (at different times) by assigning different variable addresses to it.

```cpp
// This program demonstrates the use of the indirection operator *
#include <iostream>
using namespace std;

void main (int argc, char * const argv[]) {
    int x = 25, y = 50, z = 75;
    int *ptr;  // declare a pointer called ptr
    ptr = &x;  // store the address of x in ptr
    cout << "Here are the values of x, y, and z: " << x << ", " << y << ", and " << z << endl;
    ptr = &x;  // store the address of x in ptr
    *ptr *= 2;  // multiply the value in x by 2
    ptr = &y;  // store the address of y in ptr
    *ptr *= 2;  // multiply the value in y by 2
    ptr = &z;  // store the address of z in ptr
    *ptr *= 2;  // multiply the value in z by 2
    cout << "Here are the new values of x, y, and z: " << x << ", " << y << ", and " << z << endl;
    system("pause");
}
```

Sample output:

Here are the values of x, y, and z: 25, 50, and 75
Here are the new values of x, y, and z: 50, 100, and 150

So far you've seen three different uses of the asterisk:

1. As the multiplication operator: `distance = speed * time;`
2. In the definition of a pointer: `int *ptr;`
3. As the indirection operator: `*ptr = 100;`
5. Arrays and Pointers

Array names can be used as pointers, and pointers can be used as array names.

```cpp
// Array name, without brackets and a subscript, actually represents the
// starting address of the array. This means that an array name is really
// a pointer.

// This program shows an array name being dereferenced with the * operator

#include <iostream>
using namespace std;

void main (int argc, char * const argv[]) {
    int A[] = {2,4,6,8,10};
    int dummy = 23;

    cout << "Here they are again using the * operator: " << *A << " " << *(A+1) << " " << *(A+2) << endl;
    // So A[index] is equivalent to *(A+index)
    // Be careful that C++ performs no bounds checking with arrays, so it is
    // possible to assign the pointer an address that is outside the array.
    // Note: Visual C++ will give you a run-time warning. Click Continue to ignore the
    // warning and run the program

    cout << "Outside the array bounds: " << *(A+4) << " " << *(A+5) << " " << *(A+24) << endl;
    system("pause");
}
```

Sample output:

```
The first three elements of the array are: 2  4  6
Here they are again using the * operator:  2  4  6
Outside the array bounds: 10  23  -1073743728
```
6. Pointer Arithmetics

// Array name, without brackets and a subscript, actually represents the // starting address of the array. This means that an array name is really // a pointer.

// This program shows how to use a pointer to access an array and doing // pointer arithmetic

#include <iostream>
using namespace std;

void main (int argc, char * const argv[]) {
    int A[] = {2,4,6,8,10};
    int dummy = 23;
    int *ptrA;

    ptrA = A; // assign the starting location of A to the pointer

    cout << "Size of each array element is " << sizeof(A[0]) << endl;
    cout << "Elements of the array and their addresses are:" << endl;
    cout << " " << ptrA << endl;
    ptrA++;
    cout << " " << ptrA << endl;
    ptrA++;
    cout << " " << ptrA << endl;
    ptrA++;
    cout << " " << ptrA << endl;
    ptrA++;
    cout << " " << ptrA << endl;
    ptrA++;
    cout << " " << ptrA << endl;
    ptrA++;
    cout << " " << ptrA << endl;
    ptrA++;
    cout << " " << ptrA << endl;
    ptrA++;
    cout << " " << ptrA << endl;
    ptrA++;
    cout << " " << ptrA << endl;
    ptrA++;
    cout << " " << ptrA << endl;
    ptrA++;
    cout << " " << ptrA << endl;
    system("pause");
}

Sample output:

Size of each array element is 4
Elements of the array and their addresses are:
  2  0xbfffff634
  4  0xbfffff638
  6  0xbfffff63c
  8  0xbfffff640
 10  0xbfffff644
 23  0xbfffff648
-1073744308  0xbfffff64c