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.

short	2 bytes
int	4 bytes
long	4 bytes
float	4 bytes
double	8 bytes

Each byte of memory has a unique address. A variable's address is the address of the first byte allocated to that variable. To find out the address of where a variable is stored in memory we use the **address operator** (**&**) symbol. When the **&** symbol is placed in front of a variable name (such as &amount), it returns the location 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 memory size allocated for the variable, and the actual content of the variable.

```
// The address operator & returns the memory address of a variable
// The sizeof function returns the amount of storage allocated for the variable

#include <iostream>
using namespace std;

int main() {
   int x = 25;
   double y = 34;

   cout << "The value stored in x is " << x << endl;
   cout << "The size of x is " << sizeof(x) << " bytes" << endl;
   cout << "The address of x is " << &x << endl;

   cout << endl;
   cout << "The value stored in y is " << y << endl;
   cout << "The size of y is " << sizeof(y) << " bytes" << endl;
   cout << "The address of y is " << &y << endl;

   return 0;
}
```

Sample output:

```
The value stored in x is 25
The size of x is 4 bytes
The address of x is 000000C057AFFBD4

The value stored in y is 34
```



```
The size of y is 8 bytes
The address of y is 000000C057AFFBF8
```

2. Pointer Basics 2

Memory addresses can also be stored in variables. These pointer variables for storing memory addresses are called **pointers**. With pointer variables you can indirectly manipulate data stored in other variables.

We use the * symbol to declare a pointer variable. For example, to declare a pointer variable that points to a location for storing an integer we do

```
int *ptr;  // declare a pointer variable called ptr
```

And to declare a pointer variable that points to a location for storing a float we do

```
float *ptr;  // declare a pointer variable called ptr
```

```
// Pointer variables, which are often just called pointers, are designed to
// hold memory addresses. With pointer variables you can indirectly
// manipulate data stored in other variables.

// This program stores the address of a variable in a pointer variable

#include <iostream>
using namespace std;

int main() {
   int x = 25;
   int *ptr;  // declare a pointer called ptr

   ptr = &x;  // store the address of x in ptr

   cout << "The value stored in x is " << x << endl;
   cout << "The address of x is " << ptr << endl;
}
```

Sample output:

```
The value stored in x is 25
The address of x is 000000830FF5F544
```

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. Note that the use of the * here is different from the use of the * in Pointer Basics 2 above.

```
// This program demonstrates the use of the indirection operator *

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



```
int main() {
   int x = 25;
   int *ptr;   // declare a pointer called ptr
   ptr = &x;   // store the address of x in ptr

   cout << "The value stored in x is " << *ptr << endl;
   *ptr = 73;      // store 73 in x
   cout << "The value stored in x after assignment is " << *ptr << endl;

   return 0;
}
```

Sample output:

```
The value stored in x is 25
The value stored in x after assignment is 73
```

4. Pointer Basics 4

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

```
// This program demonstrates the use of the indirection operator *

#include <iostream>
using namespace std;

int main() {
   int x = 25, y = 43, z = 18;
   int* ptr;

   cout << "The original values: x=" << x << ", y=" << y << ", z=" << z << endl;

   ptr = &x;              // store the address of x in ptr
   *ptr = *ptr * 2;// multiply the value in x by 2

   ptr = &y;              // store the address of y in ptr
   *ptr = *ptr * 2;// multiply the value in y by 2

   ptr = &z;              // store the address of z in ptr
   *ptr = *ptr * 2;// multiply the value in z by 2

   cout << "The new values: x=" << x << ", y=" << y << ", z=" << z << endl;

   return 0;
}
```

Sample output:

```
The original values: x=25, y=43, z=18
The new values: x=50, y=86, z=36
```


As you've seen there are three different uses of the asterisk:

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

So be careful and make sure that you understand the different usage.

5. Arrays and Pointers

Array names can be used as pointers, and pointers can be used as array names. Array names, without brackets and a subscript, actually represent the starting address of the array. This means that an array name is really a pointer. So we can use the * operator to dereference or get the content of the location pointed to by the array name pointer.

```
// This program shows an array name being dereference with the * operator
#include <iostream>
using namespace std;

int main() {
   int A[] = { 2,4,6,8 };
   int dummy = 23;

   cout << "The first three elements of the array are: " << A[0] << ", " << A[1] <<
", " << A[2] << endl;
   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.
   cout << "Outside the array bounds:  " << *(A+3) << ", " << *(A+4) << endl;

   return 0;
}
```

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:  8, -858993460
```


6. Pointer Arithmetic

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

#include <iostream>
using namespace std;

int main() {
   int A[] = { 2,4,6,8 };
   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 << ", " << ptrA << endl;
   ptrA++;
   cout << *ptrA << ", " << ptrA << endl;
   ptrA++;
   cout << *ptrA << ", " << ptrA << endl;
   ptrA++;
   cout << *ptrA << ", " << ptrA << endl;
   ptrA++;
   cout << *ptrA << ", " << ptrA << endl;


   return 0;
}
```

Sample output:

```
Size of each array element is 4
Elements of the array and their addresses are:
2, 000000DE714FF8C8
4, 000000DE714FF8CC
6, 000000DE714FF8D0
8, 000000DE714FF8D4
-858993460, 000000DE714FF8D8
```


7. Dynamic Memory Allocation

Dynamically allocate memory space for data storage using the `new` command.

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

int main() {
   int* ptr;       // declare a pointer to an integer
   // dynamically allocate memory for storing an integer and
   // assign the address of this memory space to ptr
   ptr = new(int);
   *ptr = 68;      // assign a value to the integer
   cout << "Size of an integer memory location is " << sizeof(*ptr) << endl;
   cout << "Content in this memory location is " << *ptr << endl;

   double* ptr2;   // declare a pointer to a double
   // dynamically allocate memory for storing a double and
   // assign the address of this memory space to ptr2
   ptr2 = new(double);
   *ptr2 = 3.1415;        // assign a value to the double
   cout << "Size of a double memory location is " << sizeof(*ptr2) << endl;
   cout << "Content in this memory location is " << *ptr2 << endl;

   return 0;
}
```

Sample output.

```
Size of an integer memory location is 4
Content in this memory location is 68
Size of a double memory location is 8
Content in this memory location is 3.1415
```

