Pointers

What is a Pointer Variable?

A pointer variable is a variable whose value is the address of a location in memory. A pointer has a variable name just like any other variable and also has a type that designates what kind of variables its contents refer to. A variable that is a pointer that can contain addresses of locations in memory containing values of type int, is of type ‘pointer to int’.

Declaring Pointers

The declaration for a pointer is similar to that of an ordinary variable, except that the pointer name has an asterisk in front of it to indicate that it’s a variable that is a pointer. For example, to declare a pointer pnumber of type int, you could use the following statement:

int* pnumber;

This declaration has been written with the asterisk close to the type name. If you want, you can also write it as:

int *pnumber;

The compiler won’t mind at all; however, the type of the variable pnumber is ‘pointer to int’, which is often indicated by placing the asterisk close to the type name. You can mix declarations of ordinary variables and pointers in the same statement. For example:

int* pnumber, number = 99;

This declares the pointer pnumber of type ‘pointer to int’ as before, and also declares the variable number, of type int. On balance, it’s probably better to declare pointers separately from other variables. The following statements certainly look clearer and putting declarations on separate lines enables you to add comments for them individually, making for a program that is easier to read.

```
int number = 99; // Declaration and initialization of int variable
int* pnumber; // Declaration of variable of type pointer to int
```

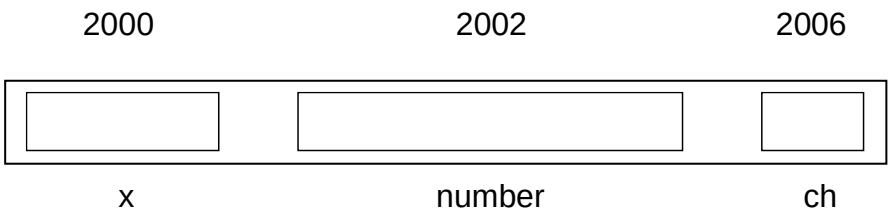
It’s a common convention in C++ to use variable names beginning with p to denote pointers. This makes it easier to see which variables in a program are pointers, which in turn can make a program easier to follow.

The Address-Of Operator

When a variable is declared, enough memory to hold a value of that type is allocated for it at an unused memory location. This is the address of the variable.

Example:

short int x; float number; char ch;



What you need is the **address-of operator**, &. This is a unary operator that obtains the address of a variable.
It's also called the reference operator. To set up the pointer that we have just discussed, you could write this assignment statement:

pnumber = &number; // Store address of number in pnumber

Using Pointers

Taking the address of a variable and storing it in a pointer is all very well, but the really interesting aspect is how you can use it. Fundamental to using a pointer is accessing the data value in the variable to which a pointer points. This is done using the **indirection operator**, *.

The Indirection Operator

You use the **indirection operator**, *, with a pointer to access the contents of the variable that it points to.
The name 'indirection operator' stems from the fact that the data is accessed indirectly.
One aspect of this operator that can seem confusing is the fact that you now have several different uses for the same symbol, *. It is the multiply operator, it also serves as the indirection operator, and it is used in the declaration of a pointer. Each time you use *, the compiler is able to distinguish its meaning by the context.

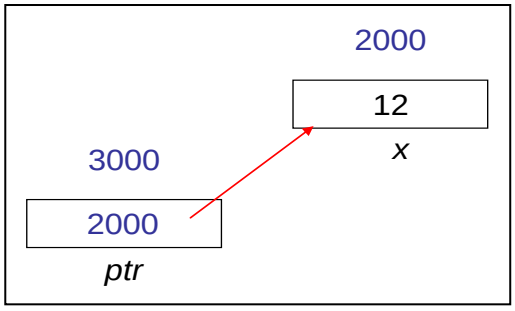
Example:

```
int* ptr;        // ptr will hold the address of an int
char* q;         // q will hold the address of a char
```

Using a pointer variable:

Example:

```
int x;
x = 12;
int* ptr;
ptr = &x;
```



// ptr holds the address of x, i.e. ptr "points to" x

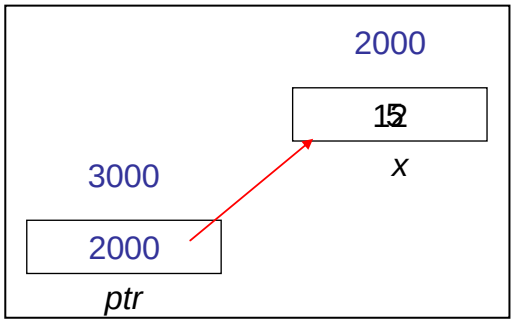
Using the deference operator:

Example(1):

```
int x;
x = 12;
int* ptr;
ptr = &x;

*ptr = 5;
```

// Changes the value at address ptr to 5



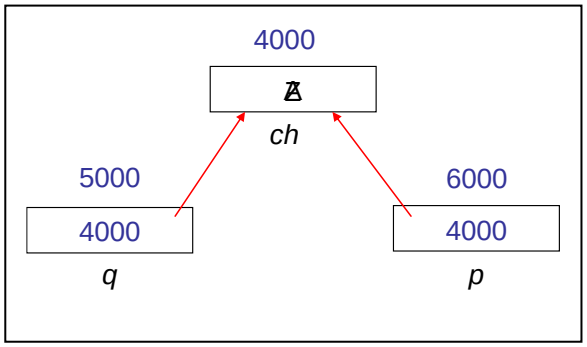
Example(2):

```
char ch;
ch = 'A';

char* q;
q = &ch;

*q = 'Z';
char* p;
p = q;
```

// The rhs has value 4000 now p & q both point to ch



Assignment:

Run the following code then check the results:

```
#include<iostream>
using namespace std;
int main()
{
	int x=3;
	int* ptr1;
	int* ptr2;
	cout<<endl<<&x<<endl;		//  Address of X
	ptr1=&x;
	x+=10;
	cout<<endl<<x<<endl;
	*ptr1=x*2;
	cout<<endl<<x<<endl;
	cout<<endl<<ptr1<<endl;   // Contents of pointer ptr1
	cout<<endl<<&ptr1<<endl;		// Address of ptr1
	ptr2=ptr1;
	cout<<endl<<ptr2<<endl;   // Contents of pointer ptr2
	cout<<endl<<&ptr2<<endl;  // Address of ptr2
	*ptr2=x+5;
	cout<<endl<<x<<"\t"<<*ptr1<<"\t"<<*ptr2<<endl;
	return 0;
}
```