An Introduction to C++ Pointers

Why Study Pointers?

Pointers are needed frequently when programming in C. Pointers are needed less often when working in C++, but nonetheless, it is essential for a C++ programmer to have a clear understanding of pointers and addresses. It has been said that 90% of the bugs that appear in C and C++ programs are due to the misuse of pointers!

What is a Pointer?

A **pointer** (variable) is a variable that holds the address of another variable of same data type.

Declaring Pointer Variables

To declare a pointer variable, one simply puts an "*****" after the name of the data type and before the variable name. For example;

```
int *P1;    //declares P1 to be a pointer to an int,  "int* P1;" also works
```

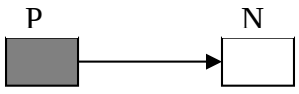
This means that instead of holding an int, `P1` holds the address of an int.

Aiming Pointers

Often one wishes to "point" or "aim" a pointer variable at a specific ordinary variable. To do this the **address operator**, **&**, is used. For example;

```
int N = 1;
int *P = &N;     //declare a pointer and aim it at N.
                 //this means that P holds the address of N
```

Notation: when P holds the address of N, we draw the following diagram



Getting at Information Via Pointers

Once P points at N, the information can be accessed via P by using the **dereference operator**, *****. For example, assuming the declarations above,

```
cout << *P;      // output: 1

*P = 123;        // N now holds 123

cout << N;       // output: 123
```

Why Are Pointers Hard to Use?

1) They are indirect, rather than a direct way of accessing information.

2) Pointer notation generally tends to be confusing. Unlike the clean, mathematical notation that is used for simple variables, pointer notation is often ill-chosen by language designers. Note for example, that in the declaration "int *P;", and the statement "*****P= 123;", the "*" has two different meanings.

So Why Do We Have to Use Pointers?

1) C passes all parameters by value, so addresses (pointers) must be passed to functions if the function is to change a value of an actual parameter. Thus, there is no escape from pointers if one programs in C. The function below shows how this is done. The function merely interchanges the values of two int memory locations.

```
void Swap(int *X, int *Y)
{
    int Temp = *X;    //save contents of what X points at
    *X = *Y;          //*X now holds what *Y has
    *Y = Temp         //copy saved value to where Y points
}
```

2) Arrays are not suitable for solving certain problems and dynamic data structures such as linked lists are needed. Pointers are indispensable when working with dynamic data structures.

3) Pointers and arrays have a close relationship in C and C++. An understanding of pointers is required in order to work with arrays. This relationship is partially demonstrated in the program below.

```
void main()
{
   int A[] = {3, 5, 7, 11, 13, 17, 19};
   int *P = A;

   cout << "*P = " << *P << endl;         //output: *P = 3
   cout << "P[1] = " << P[1] << endl;     //output: P[1] = 5
   P = &A[2];
   cout << "P[0] = " << P[0] << endl;     //output: P[0] = 7
   ++P;
   cout << *P << endl;                   //output:  11
   cout << *A;                           //output:  3
 }
```

Note that although P is declared to be a pointer, one is allowed to use array notation to reference data pointed at by P. Also, the last line shows that pointer notation can be used with arrays. The major difference between the two is that the value of a pointer variable can be changed, while arrays are constant pointers, and the array name always holds the address of subscript zero.