Pointers and arrays

There are two ways of working with data in C that go beyond what you would normally think of as a single variable. These are pointers and arrays. They are two very different concepts, but we will treat them together.

When a C/C++ program passes a variable to a function, it passes the value of the variable, but not the memory location. What this means is that you cannot change the value of the variable within a function.[1]

The way that you can work around this is to use *pointers* to the location in memory. Then inside the function, you can tell the computer to look up the value stored at that memory location, rather than the memory location itself. This manner of dealing with variables stems from the fact that C was originally developed with the goal of writing operating systems, rather than the goal of doing numerical calculations – for running an operating system, it is useful to be able to work with specific memory locations. It takes a bit of getting used to, but after you get used to it, it's not so bad.

A pointer to a variable's memory location is specified by putting an ampersand (i.e. the & symbol)in front of it. The value of the variable stored in a memory location is specific by putting an asterisk in front of the pointer.

Some basic rules on how to deal with pointers:

1. When you're in the main program, and you want to send a variable to a function, but don't want to change it, use the variable name. This is the simplest case – everything is intuitive. When you get back from the function, the variable will have the same value it had before you passed it, *even if you do something to the variable within the function.*

2. When you're in the main program, and you want to send a variable to a function and do want the function to be able to change the value, send a pointer – the variable name with the ampersand in front of it.

3. When you're then in the function and you wish to work with the variable to which you have a pointer, you need to remember that it is a pointer that you have in that function and not the variable itself. If you want to change the value of the variable, rather than the value of the memory location, you must then put an asterisk in front of the variable name.

Here is an example, the program `read_numbers.cc`. One of the lines is just in there to print out both the pointer and the data value, and to show you that the pointer is generally a very large integer which is meaningless unless you have a lot of expertise in dealing with the innards of operating systems.

```
   /* Simple program to read in number */
/* Really just demonstrates pointers*/
#include<iostream>
using namespace std;
```

[1] You can declare global variables, which can be used in all subroutines without being passed, and do things like that, but it is considered by many people to be bad programming practice – it definitely increases the risk that you will make certain types of mistakes.

```
#define pi 3.141592653505
void read_start_vals(float *x, float*y,float *v, float *theta)
{
/* We use pointers here, because we are changing the variable's value
and sending that back to the other routine */
/* If we do not use pointers here, we will only change the values of
x,y,v,theta within the subroutine here */
cout <<"What is the initial value of x in m?\n";
cin >> *x;
cout <<"What is the initial value of y in m?\n";
cin >> *y;
cout <<"What is the initial value of theta in degrees?\n";
cin >> *theta;
theta=*theta*pi/180.0;
/*This is to convert theta into radians, since C/C++ does trig in rads
*/
cout <<"What is the initial value of v in m/s?\n";
cin >> *v;
cout << "For x, the memory value is" << x << "and the actual value
is " <<*x <<endl;
return;
}

main()
{
int i,imax;
float x,y,v,theta,vx,vy,t,fx,fy;
read_start_vals(&x,&y,&v,&theta);
cout << x <<" "<< y<<" " << v <<" "<< theta << endl;
}
```

Arrays

The other type of data that we have to learn to deal with is data stored in arrays. An array is basically the computer equivalent of a vector or a matrix (or tensors when you eventually get to them). It's a set of numbers sorted in a particular way and stored in a single variable.

There are a few things to remember. First, in C, the vectors start from element number 0, rather than from element number 1. In many other programming languages vectors start from element 1. This will be an important thing to check on when/if you start to program in other languages. Second, the way that we pass arrays to functions is as the whole array, without using symbols indicating they're pointers. When we have the whole array, the compiler assumes that it is a pointer to the first element of the array, and then you can work with the variables as though you had passed a pointer. However, when

you send a single element of an array to a function, you need to use a pointer symbol. Take a look at this program, which reads in two arrays of user-defined size, and then takes their dot product:

```
/*This is a simple program to read two vectors and compute their dot
products */

#include<iostream>
using namespace std;
#define MAX_DIM 10

void readinvalues(double vector1[], double vector2[],int *dimensions)
{ /*At the top, we just use vector1[] because C knows we want to use
*/
/*a pointer to the whole array */
int i;
do
{
cout << "How many dimensions do the vectors have?" << endl;
cin >> *dimensions;
/* We used a pointer to dimensions because we want to be able to send
back to the main loop how many dimensions our vector has */
}
while (*dimensions>MAX_DIM);
for (i=0;i<*dimensions; i=i+1)
{
cout <<"Enter vector 1, element" << i << endl;
cin >> vector1[i];
}
for (i=0;i<*dimensions; i=i+1)

   {
cout <<"Enter vector 2, element" << i << endl;
cin >> vector2[i];
}

return;
}

double dotproduct(double vector1[],double vector2[],int dimensions)
{
int i;
double dp;
dp=0.0;
for (i=0;i<dimensions;i=i+1)
{
dp=dp+vector1[i]*vector2[i];
/*Here, we leave vector1[i] alone, rather than putting an */
/*asterisk before it, because we are not passing it to */
/*another function.  */
```

```
cout << "i,vector1,vector2=" << " "<< i << " " << vector1[i] << " "
<< vector2[i] << endl;
}
return(dp);
}
main()
{
int dimensions;
int i;
double vector1[MAX_DIM];
double vector2[MAX_DIM];
double dp;
readinvalues(vector1,vector2,&dimensions);
/* Dimensions needs an & but vector1 and vector2 don't because the
full arrays are being passed.*/ dp=dotproduct(vector1,vector2,dimensions);
cout << "The dot product is:  " << dp <<endl;
}



}
```

You can also specify multi-dimensional arrays – for example to do matrix calculations. It can also be a convenient way to keep the position and velocity for an object – that could be a 2×3 array, with:

`x[0,0] =` x
`x[0,1] =` y
`x[0,2] =` z
`x[1,0] =` v_x
`x[1,1] =` v_y
`x[1,2] =` v_z

Choices like this are often a matter of taste, rather than a matter of what is the best way to do a programming problem. The choice to put both the position and velocity into the same variable is likely to make it a little bit easier to get the code written without making a mistake, but a little bit harder for someone knew coming to the problem to change the code if you do not comment it well.