*Notes 11/8*

Const int MAXSIZE = 5;  
double da[MAXSIZE];  
int k;  
double\* dp;  
  
for (k = 0; k < MAXSIZE; k++)  
 da[k] = 3.6;  
  
for (dp = &da[0]; dp < da + MAXSIZE; dp++)  
 \*dp = 3.6;  
  
**\*&x 🡺 x**

&a[i] + j 🡺 &a[i+j] &a[i] – j 🡺 &a[i-j]  
&a[i] < &a[j] 🡺 i < j  
a <==> &a[0]  
p[i] <==> \*(p + i)  
  
As a parameter to a function, T p[] and T\* p are interchangeable.  
  
int m = lookup(people+2, 3, “grant”); // 2  
int m = lookup(&people[2], 3, “grant”); // 2  
  
char s[10] = “jellyfish”;  
cout << strlen(s); // writes 9  
cout << strlen(s+5); // writes 4  
  
cout << s; // writes *jellyfish*cout << (s+5); // writes *fish*

Strcpy(s+5, “duck”); //s is now *jellyduck*

Int findFirstNegative(double a[], int n)  
{  
 for (int k = 0; k < n; k++)  
 if (a[k] < 0)  
 return k;  
 return -1;  
}  
  
int main()  
{  
 double da[5];  
 …  
 int pos = findFirstNegative(da,5);  
 if (pos == -1)  
 cout << “There are no negative elements” << endl;  
 else  
 {  
 cout << “The first negative value is “ << da[pos] << endl;  
 cout << “It’s at position “ << pos << endl;  
 }  
}  
  
The null pointer value is a pointer value that is guaranteed not to point to anything valid. If you try to follow a null pointer the program will typically crash right away.

double\* findFirstNegative(double a[], int n)  
{  
 for (double\* dp = a; dp < a + n; dp++)  
 if (\*dp < 0)  
 return dp;  
 return NULL;  
}  
  
int main()  
{  
 double da[5];  
…  
 double\* p = findFirstNegative(da,5);  
 if (p == NULL)  
 cout << “There are no negative elements” << endl;  
 else  
 {   
 cout << “The first negative value is “ << \*p << endl;  
 cout << “It’s at position “ << (p – da) << endl;  
 }  
}  
  
Null pointers are a common way for the program to express that it did not work correctly or did not do its job. If it returns a value, it worked; if it returns NULL, something went wrong.