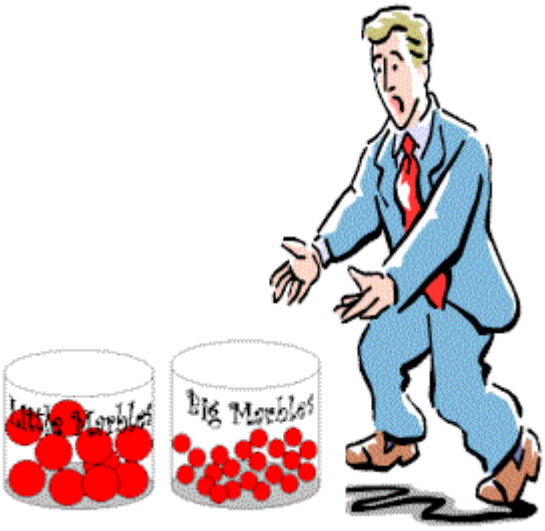
Sorting

Sorting means to arrange a list of values in numerical or alphabetical order. One of the important functions in any sorting program is a swap. Swap can be passed 2 values and exchange them. In order for swap to work the values must be passed by reference.

Swap

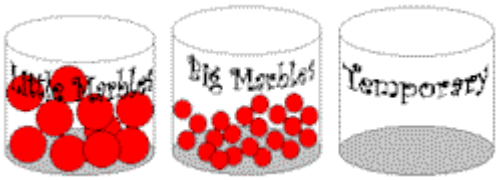


Pour Al! He just realized that he has his marbles in the wrong containers!

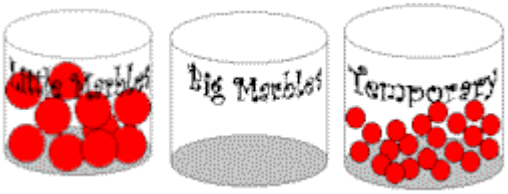
He better swap them fast!

Can he develop an algorithm?

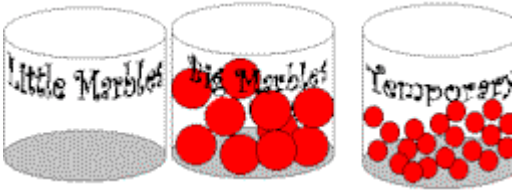
First, he needs another container.
We will refer to the extra container as temporary.



He pours the marbles from "Big" into "Temp".

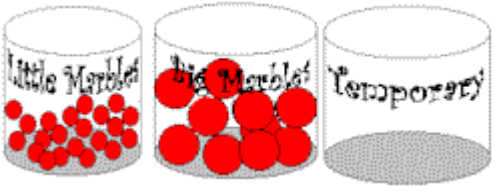


After the marbles from "Big" are stored safely away in "Temp", he pours the marbles from "Little" into "Big".



Next he pours the marbles from "temp" into "Little".

The marbles are now in the right containers.



The algorithm can be expressed as:

```
Temp = Value1
Value1 = Value2
Value2 = Temp
```

Swap in C++

We will start with a small program that reads in 2 numbers and swaps them if they are out of order.

```
1  //Sort 2 numbers
2  #include<iostream.h>
3  void swap(int& n1, int& n2)
4  //receive  n1 and n2 by reference, exchange
5  { int temp;
6    temp=n1;
7    n1=n2;
8    n2=temp;
9  } //swap
10
11 void main() //input 2 numbers, sort, print
12 { int first, second;
13   cout<<"enter a number ";
14   cin>>first;
15   cout<<"enter a number ";
16   cin>>second;
17   if (first>second) swap(first, second);
18   cout<<"In order: "<<first<<" "<<second<<"\n";
19 } //main
```

Explanation:

3. Swap receives 2 integers by reference. We know it is by reference because of the &.
5. The local variable temp must be the same type as the variables received.
6. Store n1 in temp
7. Copy n2 into n1
8. Copy temp into n2
9. The values of the variables that n1 and n2 reference have now been exchanged.
12. The variables first and second are local to main.
17. Swap is called only if the values are out of order.

Good test data would include:

- first< second
- second< first
- first = second

Sort 3 Integers

The next example will read in and swap 3 integers. This example is important to illustrate that any two values can be refered to as n1 and n2 in swap.

```
1  //Sort 3 numbers
2  #include<iostream.h>
3  void swap(int& n1, int& n2)
4  //receive  n1 and n2 by reference, exchange
5  { int temp;
6    temp=n1;
7    n1=n2;
8    n2=temp;
9  } //swap
10
11 void main() //input 3 numbers, sort, print
12 { int num1, num2, num3;
13   cout<<"enter a number ";
14   cin>>num1;
15   cout<<"enter a number ";
16   cin>>num2;
17   cout<<"enter a number ";
18   cin>>num3;
19   if (num1>num2) swap(num1, num2);
20   if (num2>num3) swap(num2, num3);
21   if (num1>num2) swap(num1, num2);
22   cout<<"In order: "<<num1<<" "<<num2<<" "<<num3<<"\n";
23 } //main
```

Explanation:

If swap is called in line 19 or 21 then n1 will refer to num1 and n2 will refer to num2.
If swap is called in line 20 then n1 will refer to num2 and n2 will refer to num3.
The comparison in line 21 will occur if the numbers are in reverse order. Suppose the values of num1, num2, and num3 are as shown below:

num1	num2	num3	
6	5	4	Original values
5	6	4	Values after line 19 and swap is executed.
5	4	6	Values after line 20 and swap is executed.
4	5	6	Values after line 21 and swap is executed.

An example of good test data for this program would be each set of numbers: 3 4 5; 3 5 4; 4 3 5; 4 5 3; 5 3 4; 5 4 3; 3 3 4; 3 4 3; 4 3 3; 3 3 3.
Of course, it does not have to be just 3, 4 and 5. The numbers shown represent every possible relationship.

Sorting 3 integers in an Array

Our next example is identical to the one above except that an array is used to store the 3 values. Because it is an array, loops are used wherever appropriate.

```
1  //Sort array of 3 numbers
2  #include<iostream.h>
3  void swap(int& n1, int& n2)
4  //receive  n1 and n2 by reference, exchange
5  { int temp;
6    temp=n1;
7    n1=n2;
8    n2=temp;
9  } //swap
10
11 void main() //input an array of 3 numbers, sort, print
12 { int num[3];
13   int index;
14   for (index=0; index<3; index++)
15   { cout<<"enter a number ";
16     cin>>num[index];
17   } //for
18   if (num[0]>num[1]) swap(num[0], num[1]);
19   if (num[1]>num[2]) swap(num[1], num[2]);
20   if (num[0]>num[1]) swap(num[0], num[1]);
21   cout<<"In order: ";
22   for (index=0; index<3; index++)
23     cout<<num[index]<<" ";
24   cout<<"\n";
25 } //main
```

Swapping Strings

```
1  //Sort 2 names
2  #include<iostream.h>
3  #include<string.h>
4  void swap(char n1[], char n2[])
5  //receive  strings n1 and n2, exchange
6  { char temp[20];
7    strcpy(temp,n1);
8    strcpy(n1,n2);
9    strcpy(n2,temp);
10 } //swap
```

Let's look at the swap first, then main.

3. Don't forget to include **`string.h`** for the string functions.

4. Arrays are always passed by reference, so we don't need an &.

7. This is the line that would be **`temp=n1;`** if we were using integers. To do the same thing with strings, we need to use strcpy. (string copy)

This program is continued on the next page.

```
11 void main() //input 2 names, sort, print
12 { char first[20], second[20];
13   cout<<"enter a name: ";
14   cin.getline(first,20); //allow first and last
15   //cin.getline stops when it gets to '\n' or max.
16   //if max, there may be something in buffer
17   if (strlen(first)==19)
18      cin.ignore(100,'\n');  //clear the input buffer
19   cout<<"enter a name: ";
20   cin.getline(second,20);
21   if (strcmp(first,second)>0) //if first ">" second
22     swap(first, second);
23   cout<<"In order: "<<first<<" "<<second<<"\n";
24 } //main
```

12. each name can have a maximum of 19 letters. One of the characters must be reserved for the '\0'.
14. We use cin.getline so that we can read a name with spaces, and so it doesn't crash if we type a really long name. It will stop when it it hits the maximum (20 here), or when it hits a line feed (\n)
17. If the length of the name is 19 there may be something still in the input buffer. If so, we will use cin.ignore to empty the buffer.
21. This is the line that said **`if (first > second) swap(first, second);`** when we were working with integers. We need **`strcmp`** for strings.

An Array of 3 Names

The next program sorts an array of 3 names. Compare this program to the previous program. Notice that swap is exactly the same.

```
1  //Sort an array of 3 names
2  #include<iostream.h>
3  #include<string.h>
4  void swap(char n1[], char n2[])
5  //receive  strings n1 and n2, exchange
6  { char temp[20];
7    strcpy(temp,n1);
8    strcpy(n1,n2);
9    strcpy(n2,temp);
10 } //swap
11
12 void main() //input 3 names, sort, print
13 { char names[3][20]; //3 names: [0],[1],[2]
14   int index;
15   for(index=0; index<3; index++)
16   {  cout<<"enter a name: ";
17      cin.getline(names[index],20); //allow first and last
18      if (strlen(names[index])==19)
19      cin.ignore(100,'\n');  //clear the input buffer
20   } //for loop
21   if (strcmp(names[0],names[1])>0)
22     swap(names[0],names[1]);
```

```
23    if (strcmp(names[1],names[2])>0)
24      swap(names[1],names[2]);
25    if (strcmp(names[0],names[1])>0)
26      swap(names[0],names[1]);
27    for(index=0; index<3; index++)
28      cout<<names[index]<<"\n";
29  } //main
```

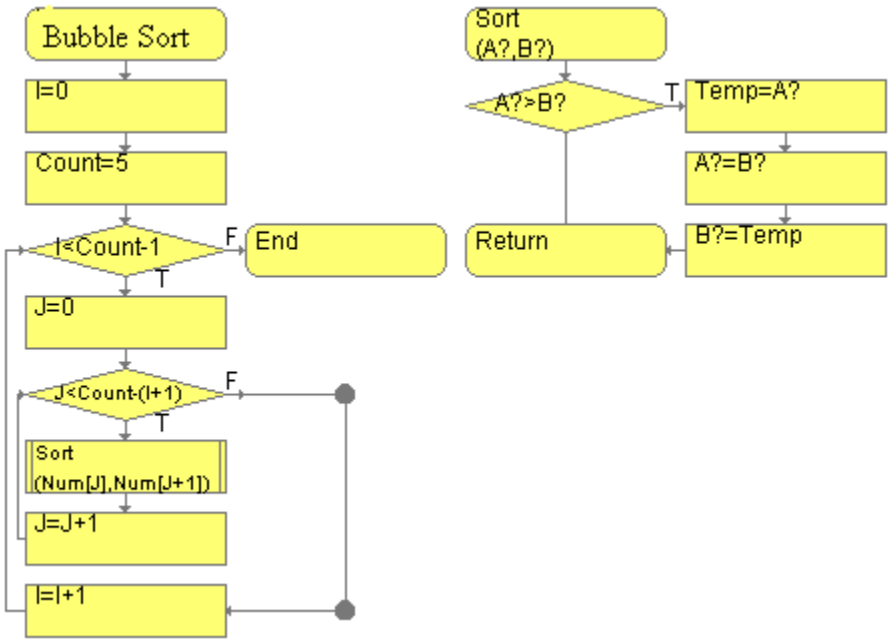
Big Arrays

An array with just 3 values is somewhat easy to sort. The programs above gave you an understanding of swap for integers and strings and showed how to call swap with arrays of values. With more than 3 values we will need a different algorithm for performing the sort.

There are several algorithms for sorting. (the algorithm is the same no matter what kind of array you use.)

Bubble Sort

Bubble sort is a popular sort because it is fairly easy for beginners to write (compared to the other methods). The basic idea is to loop through i=0 to count-1 and compare array[i] to array[i+1], swapping if out of order. When the loop finishes, the largest value will have "sunk" to the bottom. This is called one pass. On the next pass the last element can be ignored so on the second pass the loop only goes from i=0 to count-2. A nested loop is used to repeat the passes.



One problem with a Bubble sort is that it is extremely slow. The only time it performs fairly well is when the array is already in order. Part of the reason that it is so slow is that by comparing array[I] to array[I+1] the sort takes on the order of n^2 comparisons. It can also require n^2 swaps, and swaps take even more time than compares.

Selection Sort

The selection sort is another sort that is fairly easy to write. The basic idea is to find the position of smallest value from 0 to last. Swap array[0] and array[smallest]. This is the first pass. On the second pass, find the position of smallest value from 1 to last. Swap array[1] and array[smallest]. Continue until the whole array is sorted. Sometimes the selection sort is described as finding the largest from 0 to last and swap array[last] and array[largest]. On the next pass find the largest from 0 to last-1. Continue until the array is sorted.