Array Example: Searching

Imagine being given an array of values, searching for a particular one. In an unsorted array, we can do this as follows:

Go through each element one by one, comparing it to what we are searching for. If it's found, we're done. If we go all the way through the array and we don't find it, it's not there.

Let's write a function to take care of this task. Here's the prototype:

```
// Pre-conditions: length is the length of the array values.
// Post-condition: The function returns 1 if val is stored in the
//                        array values and 0 otherwise.
int Search(int values[], int length, int val);
```

Here is the code:

```
int Search(int values[], int length, int val) {

     int index;
     for (index = 0; index<length; index++)
          if (values[index] == val)
               return 1;

     return 0;
}
```

Could you adjust this to return the number of occurences of val in the array values?

Array Example: Sorting

Included below is a small program that illustrates an algorithm to sort an array of 20 integers. We will now develop this program, step by step.

```
// Arup Guha
// 10/15/01
// A simple sorting program
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

#define     SIZE  20

void Print_Array(int values[], int length);
void Fill_Array(int values[], int length, int max);
void swap(int *num1, int *num2);
void Move_Max(int values[], int max_index);
void Simple_Sort(int values[], int length);

int main() {

  int my_vals[SIZE];
  srand(time(NULL)); // Initalize the random number generator.

  // Initialize the array with random values and print these.
  Fill_Array(my_vals, SIZE, 100);
  printf("original array : ");
  Print_Array(my_vals, SIZE);

  // Sort this array and print the result.
  Simple_Sort(my_vals, SIZE);
  printf("sorted array : ");
  Print_Array(my_vals, SIZE);

  return 0;
}

// Pre-condtions: the second parameter is the length of the array
//                specified by the first parameter.
// Post-conditions: The first parameter will be sorted from lowest to
//                  highest values.
void Simple_Sort(int values[], int length) {

  int i;
  // Find the largest value and put that in its correct location,
  // successively.
  for (i=length-1; i> 0; i--) {
    Move_Max(values, i);
  }
}
```

```c
// Pre-conditions: max_index is a valid index to the array values.
// Post-condition: The largest value in the array stored in between
//                 indexes 0 and max_index inclusive will be swapped
//                 into the max_index location of the array.
void Move_Max(int values[], int max_index) {

  int max, i, maxi;
  max = values[0]; // Set up current max and max index.
  maxi = 0;

  // Loop through all possible candidates, updating the max and the
  // index that stores that maximum, if necessary.
  for (i=1; i<=max_index; i++) {

    if (max < values[i]) {
      max = values[i];
      maxi = i;
    }
  }

  // Swap the maximum value in range to the appropriate spot in the array.
  swap(&values[maxi], &values[max_index]);
}

// Pre-condition: i and j are valid indexes to the array values.
// Post_condition: The values stored in indexes i and j of values will be
//                 swapped.
void swap(int *num1, int *num2) {

  int temp;

  temp = *num1;
  *num1 = *num2;
  *num2 = temp;
}

// Pre-condition: length is the length of the array values.
// Post-condition: all the numbers stored in values will be printed out,
//                 from the values stored in index 0 to index length-1.
void Print_Array(int values[], int length) {

  int i;
  for (i=0; i<length; i++)
    printf("%d ", values[i]);
  printf("\n");
}

// Pre-condition: length is the length of the array values and max<32767.
// Post-condition: the array values will be initialized with random values
//                 in between 1 and max.
void Fill_Array(int values[], int length, int max) {

  int i;
  for (i=0; i<length; i++)
    values[i] = (rand()%max) + 1;
}
```

Basic design for a sorting algorithm

Anytime you are trying to come up with an algorithm, a good question to ask yourself is, "is this a task that I complete, normally, every day?" If the answer is yes, then all you need to do to come up with your algorithm is analyze what you do to solve your everyday problem, and translate that into a specific set of steps that can be implemented in a programming language.

Sorting does come often in our every day lives. Whenever I arrange papers to hand back in class, I typically sort them so it's easier for me to enter my grades in the spreadsheet, which already has all my students sorted alphabetically by last name. Whenever I play cards, I sort the cards in a particular order.

With reference to sorting cards, the method that I use to do so is that I search for the "largest" card and then swap that to its correct location (in my hand, that's to the far left). I look at the rest of the cards and grab the largest one left. Then, I swap that card directly to the right of the previous largest card. I repeat these sets of steps until my entire hand of cards is sorted.

What we need to do is find a way to utilize this "algorithm" in c. Keep in mind that our "cards" will simply be integers stored in an array. Our first major subtask is as follows:

Find a way to find the maximum card in a subset of the cards, and then "swap" that card to its correct location. Converting that into c, we want to write a function that does the following:

Takes in an array, and a maximum index, finds the largest value, stored in the array upto that maximum index, and then swaps that value into the maximum index location, ie. the correct place for that number.

Move_Max function

In order to write this function, we first need a prototype:

```
// Pre-conditions: max_index is a valid index to the array values.
// Post-condition: The largest value in the array stored in between
//              indexes 0 and max_index inclusive will be swapped
//              into the max_index location of the array.
void Move_Max(int values[], int max_index);
```

Furthermore, it looks like we will need to swap values inside the array. Here's the swap function we've used previously:

```
// Pre-condition: i and j are valid indexes to the array values.
// Post_condition: The values stored in indexes i and j of values will be
//              swapped.
void swap(int *num1, int *num2) {

    int temp;

    temp = *num1;
    *num1 = *num2;
    *num2 = temp;
}
```

Now we are ready to tackle our Move_Max function.

Here is the function. The key to writing this function is to keep in mind that we always need to store the maximum value seen in the array "thus far", as well as the index that value is stored in. In my implementation below, the local variables max and maxi store these values.

```
void Move_Max(int values[], int max_index) {

    int max, i, maxi;
    max = values[0]; // Set up current max and max index.
    maxi = 0;

    // Loop through all possible candidates, updating the max and the
    // index that stores that maximum, if necessary.
    for (i=1; i<=max_index; i++) {

        if (max < values[i]) {
            max = values[i];
            maxi = i;
        }
    }
    swap(&values[maxi], &values[max_index]);
}
```

Now, we must find a way to use this function to sort our whole entire array. The basic idea is as follows:

We need to first call this function on the entire array, so to speak. Once that is done, we know that the maximum value is stored in the correct location in our array, thus we are really left with the subproblem of sorting the REST of the array!!! Although this could be handled recursively, it is more straightforward to handle this through a for loop.

The Sort function

Here is an implementation of the idea discussed. Notice that we MUST "run" our loop backwards. Why is that?

```
// Pre-condtions: the second parameter is the length of the array
//              specified by the first parameter.
// Post-conditions: The first parameter will be sorted from lowest to
//               highest values.
void Simple_Sort(int values[], int length) {
     int i;
     // Find the largest value and put that in its correct location,
     // successively.
     for (i=length-1; i > 0; i--) {
        Move_Max(values, i);
     }
}
```

Now, in order to test our sort function, it helps to have a couple other functions in place. In particular, it's nice to be able to initialize an array with random values. Furthermore, it is nice to be able to print out the contents of an array at any time.

Here are a couple prototypes of functions to allow us to do these two items:

```
// Pre-condition: length is the length of the array values.
// Post-condition: all the numbers stored in values will be printed out,
//               from the values stored in index 0 to index length-1.
void Print_Array(int values[], int length);

// Pre-condition: length is the length of the array values and max<32767.
// Post-condition: the array values will be initialized with random values
//               in between 1 and max.
void Fill_Array(int values[], int length, int max);
```

Print_Array and Fill_Array functions

The code for both of these functions is fairly straight-forward. You loop through the entire array, and execute a small task for each array element.

```
void Print_Array(int values[], int length) {

    int i;
    for (i=0; i<length; i++)
        printf("%d ", values[i]);
    printf("\n");
}


void Fill_Array(int values[], int length, int max) {

    int i;
    for (i=0; i<length; i++)
         values[i] = (rand()%max) + 1;
}
```

We can put this all together by testing the functions we have written as follows:

```
#define     SIZE        20

int main() {

    int my_vals[SIZE];
    srand(time(NULL)); // Initalize the random number generator.

    // Initialize the array with random values and print these.
    Fill_Array(my_vals, SIZE, 100);
    printf("original array : ");
    Print_Array(my_vals, SIZE);

    // Sort this array and print the result.
    Simple_Sort(my_vals, SIZE);
    printf("sorted array : ");
    Print_Array(my_vals, SIZE);

    return 0;
}
```

Let's trace through an example of this program running. For simplicity's sake, let us trace the program through with an array of size 5, instead of 20.