Collection or array in c

So far, we are only able to store numbers as individual variables. What if we need a collection of numbers? Say, 30 numbers? Numbers that represent the grades of the students in a class. And we want to compute for the average grade of the class. What are we going to do?

Let's look a different scenario. Say we are in a gymnastics competition and there are 20 judges. The scores they give the gymnasts have to be recorded. Are we going to use 20 different variables? Sure we can! And then, we compute for the average score.

```
double sum =
↪  s1+s2+s3+s4+s5+s6+s7+s8+s9+s10+s11+s12+s13+s14+s15+s16+s17+s18+s19+s20;
double ave = sum/20;
```

In a gymnastics competition, the lowest and highest scores the gymnasts get are dropped in computing for the average ("eliminates" bias). This means looking for smallest score and the biggest score. Recall how it was done for only 3 values. Are we going to compare the 20 variables with each other?

For sure we can, again! It is going to be very inconvenient to do this, wouldn't it?

The solution to this problem is to store the scores as one unit, very much like a set of numbers.

There is a construct in c called arrays. Essentially, they are a collection of entities. Like an array of gold jewelry, an array of red roses, an array of chocolates! In our case, an array of double values (the scores gymnasts get are double values - both for difficulty and execution scores).

This array requires that the entities in the collection should be of the same data type. Let us work with an array of integers first.

Declaring an array of integers (or of any type) in c requires 2 things to distinguish it from an ordinary integer (or the corresponding type of the array). These are the square brackets (this tells the compiler that the identifier is an array). And the second is a constant positive integer (literal) that specifies the size of the array. The size is tells the compiler how many elements or items the array (collection) can hold.

```
int array1[5];
int array2[10];
```

Both $array1$ and $array2$ above are arrays of type int. The square brackets are used to declare an array in c. Again, this is what distinguishes $array1$ and $array2$ from regular integer identifiers.

In the sample, $array1$ can now contain up to a maximum of 5 ints. $array2$ on the other can contain up to a maximum of 10 ints. Maximum because it is not required to fill them up to brim! When declaring arrays, make sure you allocate only what the solution needs so that memory is not wasted.

We are now ready to use them. We illustrate this by using $array1$. Since $array1$ has the capacity to hold a maximum of 5 ints, these 5 ints must be stored in a systematic manner. The items stored in arrays are called elements. And these elements are stored in sequential order. Storage locations in arrays are identified and referred to as indices. The indices of the arrays go from 0 to $size - 1$. In the case of $array1$ here, the indices go from 0 to 4.

This means that the first element is stored in index 0, the second element in index 1, the third in index 2, the fourth in index 3 and finally, the 5th element in index 4! Whew!

The index cannot be less than 0, and it shouldn't go beyond $size-1$. When we go beyond $size-1$, this is an index-out-of-bounds error. Remember that we only requested for 5 integers in the case of $array1$. The assignment `array1[5] = 17;` is problematic. This means we are accessing and changing the value of the 6th element of the array. But there is no 6th element since the size or the capacity of $array1$ is just 5. `array1[10] = 17;` is problematic as well like the previous one.

To access the elements in a particular index, **indexing** is used. This is through the use of the square brackets and specifying a valid index as shown below.

```
printf("%i %i %i %i
↪  %i\n",array1[0],array1[1],array1[2],array1[3],array1[4]);
```

The code above will display all the 5 elements of $array1$ in one line. Notice, though, that the indices have a fixed pattern (they go from 0 to $size-1$, always). This means we can iterate through the elements of $array1$ via indexing as demonstrated (down) below.

```
for(i=0; i<5; i++)
    printf("%i ",array1[i]);
printf("\n");
```

Assigning values to the elements of the array requires indexing as well, with the use of the assignment operator, of course. This is shown below.

```
array1[0] = 10;
array1[1] = 96;
array1[2] = -23;
array1[3] = 0;
array1[4] = array1[1] + array1[2];
```

The values assigned as elements of the array may come from the standard input as. This is illustrated (down) below.

```
scanf("%i%i",&array1[0],&arry1[4]);
```

The code above asks for 2 integer inputs from the standard input and stores this as array1's first and fifth elements.

We may also iterate throught the elements of the array similar to what we did in the printing.

```
for(i=0; i<5; i++)
    scanf("%i ",&array1[i]);
```

Let's get back to the gymnastics competition. Leggo!

```c
#include <stdio.h>

int main(){
    double scores[20];
    double sum=0, average;

    for(int i=0; i<20; i++){ // asks for the 20 scores
        do{
            printf("Enter score from Judge %i: ", i+1);
            scanf("%i",&scores[i]); //stores the scores in
↪   the array
        }while(scores[i]<=0);   //makes sure inputs are
↪   positive.
    }

    for(i=0; i<20; i++)
        sum = sum + scores[i];  //sums up the scores

    average = sum/20;           //computes the average

    printf("The average score of the gymnast is
↪   %lf.\n",average);
}
```

Searching in Arrays

We are almost done. In gymnastics competition, one of the lowest scores and one of the highest scores are dropped or are not included in the computation of the final score of the gymnast. How do we search for the smallest item in an array? How do we search for the biggest element in an array?

Before we head on and look for the minimum element in an array, one of the core problems discussed in problem solving is searching. Given a collection of data or an array of data, and search element x, we answer the question whether x is in the array or not.

Check the solution (down) below.

```c
#include <stdio.h>

int main(){
    int nums[20];
    int x, i;

    for(int i=0; i<20; i++){ // asks for 20 numbers
        printf("Enter number %i: ", i+1);
        scanf("%i",&nums[i]);   //stores the numbers in the
↪   array
    }

    printf("Enter the search item: ");
    scanf("%i",&x); //x is item we are going to look for in
↪   array nums

    //the strategy is to scan the items of nums one by one
    //while scanning we verify if the item is x
    for(i=0; i<20; i++){
        if(scores[i] == x) //checks if a score is smaller
        ↪  than the current lowest
            break; //if this is the case, then a new lowest
            ↪  is found
    }
    //when x is not in the array the search would have
    ↪  exhausted all elements in the array
    //this means that i has reached 20, and the loop
    ↪  terminates
    //if i is less than 20, then x is found in the array
    if(i < 20)
        printf("%i is found in index %i.\n",x,i);
    else
        printf("%i is not in the array.\n",x)
}
```

Minimum Element

Going back to the search for the lowest element, the same strategy is used. But only in terms of exhausting all the elements in the array. This means we need to scan all the elements of the array to find it.

```c
#include <stdio.h>

int main(){
    int i;
    double scores[20];
    double sum=0, average, lowest;

    for(int i=0; i<20; i++){ // asks for the 20 scores
        do{
            printf("Enter score from Judge %i: ", i+1);
            scanf("%i",&scores[i]); //stores the scores in
↪  the array
        }while(scores[i]<=0);   //makes sure inputs are
↪  positive.
    }

    lowest = scores[0]; //assume the first element to be
↪  the lowest score

    //then start looking for scores that might be lesser
    ↪  than the current lowest
    //if this is found, then it becomes the new lowest
    ↪  score
    //search begins with index 1 since we already assumed
    ↪  the first element is the lowest
    for(i=1; i<20; i++){
        if(scores[i] < lowest) //checks if a score is
        ↪  smaller than the current lowest
            lowest = scores[i]; //if this is the case, then
↪  a new lowest is found
    }
}
```

Maximum Element

The same strategy is used to find the biggest element. It is demonstrated (down) below.

```c
#include <stdio.h>

int main(){
    int i;
    double scores[20];
    double sum=0, average, biggest;

    for(i=0; i<20; i++){ // asks for the 20 scores
        do{
            printf("Enter score from Judge %i: ", i+1);
            scanf("%i",&scores[i]); //stores the scores in
↪  the array
        }while(scores[i]<=0);   //makes sure inputs are
↪  positive.
    }

    biggest = scores[0]; //assume the first element is the
↪  biggest score

    //then start looking for scores that might be greater
    ↪  than the current biggest
    //if this is found, then it becomes the new biggest
    ↪  score
    //search begins with index 1 since we already assumed
    ↪  the first element is the biggest
    for(i=1; i<20; i++){
        if(scores[i] > biggest)     //checks if a score is
        ↪  bigger than the current biggest
            biggest = scores[i];    //if this is the case,
↪  then a new biggest is found
    }
}
```

Merging all these strategies in solving the gymnasts average score we have:

```c
#include <stdio.h>

int main(){
    double scores[20];
    double sum=0, average, lowest, biggest;
    int i;

    for(i=0; i<20; i++){        // asks for the 20 scores
        do{
            printf("Enter score from Judge %i: ", i+1);
            scanf("%i",&scores[i]); //stores the scores in
↪   the array
        }while(scores[i]<=0);       //makes sure inputs are
↪   positive.
        sum = sum + scores[i];      //sum can be computed
↪   while asking for input
    }

    //lowest gets assigned with scores[0] first, then
    ↪  biggest is assigned the value of lowest
    biggest = lowest = scores[0]; //assume the first
↪   element as the lowest and lowest score

    for(i=1; i<20; i++){
        if(scores[i] < lowest)      //checks if a score is
        ↪  smaller than the current lowest
            lowest = scores[i];     //if this is the case,
↪   then a new lowest is found
        if(scores[i] > biggest)     //checks if a score is
        ↪  bigger than the current biggest
            biggest = scores[i];    //if this is the case,
↪   then a new biggest is found
    }

    average =  (sum - biggest - lowest)/18; //divided by 18
↪   since 2 scores have been dropped.

    printf("The average score of the gymnast sans the
↪   lowest and biggest is %lf.\n",average);

    return 0;
}
```

The search for the lowest item and the biggest item are two separate tasks. But we can look for the "simultaneously" as shown above. Do not place `if(scores[i] > biggest)` in an else statement. Again, they are independent of each other. If we did this, then the check for a new biggest item will only be executed when no new lowest item is found.

Sorting

Since we just discussed the search for the smallest and the biggest items in an array, check the series of codes below. This time around, we will be working with an array with size 5 only so it is manageable. Compile and run each of the codes below. And observe what is happening. We will also not ask any input from the user and instead initialize the array with literal ints.

```
#include <stdio.h>
int main(){
    int arr[] = {23, 12, 9, 68, -2};
    int min = 0, i = 1;

    while(i<5){
        if(arr[i] < arr[min])
            min = i;
        i++;
    }
    printf("The smallest item/element in the array is
↪   %i.\n",arr[min]);

    return 0;
}
```

`int arr[] = {23, 12, 9, 68, -2};` this initializes the array with the corresponding elements enclosed in the curly braces. The firt int, 23, is sotred in a `arr[0]` , 12 in `arr[1]` , -2 in `arr[4]` . We did mention in an earlier section that apart from the square brackets, a size is required in declaring an array. Leaving the array size is allowed only when there is initialization. The number of items listed inside the curly braces is the size of the array. In the case above, the size of the array is 5. We still can put the size if we wish to, like this: `int arr[] = {23, 12, 9, 68, -2};` . This is also possible: `int arr[10] = {23, 12, 9, 68, -2};` . This would mean that the size of the array is 10, but only the first 5 elements are initialized, the next 5 are not. This, though, is problematic: `int arr[3] = {23, 12, 9, 68, -2};` . The size of the array is set to 3 but there are 5 listed items.

The code above should look familiar. It is searching for the smallest item in the array. And this information is stored in *min*. But unlike our original version of

finding the smallest item in an array, what is stored in *min* is the index of the elements and not the elements themselves. Look, *min* is initially zero (instead of `arr[0]`). `if(arr[i] < arr[min])` insteand of `if(arr[i] < min)`. `printf("The smallest item/element in the array is %i.\n",arr[min]);` instead of `printf("The smallest item/element in the array is %i.\n",min);`

Carefully observe the code (down) below.

```
#include <stdio.h>
int main(){
    int arr[] = {23, 12, 9, 68, -2};
    int min, i, tmp;

    // start of searching for the smallest item
    min = 0;
    i = 1;
    while(i<5){
        if(arr[i] < arr[min])
            min = i;
        i++;
    }
    printf("The smallest item/element in the array is
↪   %i.\n",arr[min]);
    // end of searching for the smallest item

    // start of swapping arr[0] and arr[min]
    tmp = arr[0];
    arr[0] = arr[min];
    arr[min] = tmp;
    // end of swapping arr[0] and arr[min]

    // start of printing the items of arr
    i=0;
    while(i<5){
        printf("%i ",arr[i]);
        i++;
    }
    printf("\n");
    // end of printing the items of arr

    return 0;
}
```

Running the program should result in the display of `-2 12 9 68 23`. The smallest item from among the 5 items is -2. Once this is found, it is swapped with the first element of the array, which, in this case, is 23.

Again, carefully study the code (down) below.

```
#include <stdio.h>
int main(){
    int arr[] = {23, 12, 9, 68, -2};
    int min, i, tmp;

    // start of searching for the smallest item from among
    ↪  the 5
    min = 0;
    i = 1;
    while(i<5){
        if(arr[i] < arr[min])
            min = i;
        i++;
    }
    printf("The smallest item/element in the array is
↪   %i.\n",arr[min]);
    // end of searching for the smallest item

    // start of swapping arr[0] and arr[min]
    tmp = arr[0];
    arr[0] = arr[min];
    arr[min] = tmp;
    // end of swapping arr[0] and arr[min]

    // start of printing the items of arr
    i=0;
    while(i<5){
        printf("%i ",arr[i]);
        i++;
    }
    printf("\n");
    // end of printing the items of arr
```

```
    // start of searching for the smallest item from among
    ↪  the 4
    // elements from index 1 to index 4
    min = 1;
    i=2;
    while(i<5){
        if(arr[i] < arr[min])
            min = i;
        i++;
    }
    printf("The smallest item/element of the array is
↪   %i.\n",arr[min]);
    // end of searching for the smallest item

    // start of swapping arr[1] and arr[min]
    tmp = arr[1];
    arr[1] = arr[min];
    arr[min] = tmp;
    // end of swapping arr[0] and arr[min]

    // start of printing the items of arr
    i=0;
    while(i<5){
        printf("%i ",arr[i]);
        i++;
    }
    printf("\n");
    // end of printing the items of arr

    return 0;
}
```

Running the program above should result in the final display of `-2 9 12 68 23`. The smallest item from among the 4 items left is 9. Once this is found, it is swapped with the second element of the array, which, in this case, is 12.

Again, carefully study the code (down) below.

```c
#include <stdio.h>
int main(){
    int arr[] = {23, 12, 9, 68, -2};
    int min, i, tmp;

    // start of searching for the smallest item from among
    ↪  the 5
    min = 0;
    i = 1;
    while(i<5){
        if(arr[i] < arr[min])
            min = i;
        i++;
    }
    printf("The smallest item/element in the array is
↪   %i.\n",arr[min]);
    // end of searching for the smallest item

    // start of swapping arr[0] and arr[min]
    tmp = arr[0];
    arr[0] = arr[min];
    arr[min] = tmp;
    // end of swapping arr[0] and arr[min]

    // start of printing the items of arr
    i=0;
    while(i<5){
        printf("%i ",arr[i]);
        i++;
    }
    printf("\n");
    // end of printing the items of arr
```

```
    // start of searching for the smallest item from among
    ↪  the 4
    // elements from index 1 to index 4
    min = 1;
    i=2;
    while(i<5){
        if(arr[i] < arr[min])
            min = i;
        i++;
    }
    printf("The smallest item/element of the array is
↪   %i.\n",arr[min]);
    // end of searching for the smallest item

    // start of swapping arr[1] and arr[min]
    tmp = arr[1];
    arr[1] = arr[min];
    arr[min] = tmp;
    // end of swapping arr[1] and arr[min]

    // start of printing the items of arr
    i=0;
    while(i<5){
        printf("%i ",arr[i]);
        i++;
    }
    printf("\n");
    // end of printing the items of arr
```

```
    // start of searching for the smallest item from among
    ↪  the 3
    // elements from index 2 to index 4
    min = 2;
    i=3;
    while(i<5){
        if(arr[i] < arr[min])
            min = i;
        i++;
    }
    printf("The smallest item/element of the array is
↪   %i.\n",arr[min]);
    // end of searching for the smallest item

    // start of swapping arr[2] and arr[min]
    tmp = arr[2];
    arr[2] = arr[min];
    arr[min] = tmp;
    // end of swapping arr[2] and arr[min]

    // start of printing the items of arr
    i=0;
    while(i<5){
        printf("%i ",arr[i]);
        i++;
    }
    printf("\n");
    // end of printing the items of arr

    return 0;
}
```

Running the program above should result in the final display of `-2 9 12 68 23`. The smallest item from among the 3 items left is 12. Once this is found, it is swapped with the third element of the array, which, in this case, is 12 (it got swapped with itself).

Again, check and student the code (down) below.

```c
#include <stdio.h>
int main(){
    int arr[] = {23, 12, 9, 68, -2};
    int min, i, tmp;

    // start of searching for the smallest item from among
    ↪  the 5
    min = 0;
    i = 1;
    while(i<5){
        if(arr[i] < arr[min])
            min = i;
        i++;
    }
    printf("The smallest item/element in the array is
↪   %i.\n",arr[min]);
    // end of searching for the smallest item

    // start of swapping arr[0] and arr[min]
    tmp = arr[0];
    arr[0] = arr[min];
    arr[min] = tmp;
    // end of swapping arr[0] and arr[min]

    // start of printing the items of arr
    i=0;
    while(i<5){
        printf("%i ",arr[i]);
        i++;
    }
    printf("\n");
    // end of printing the items of arr
```

```
    // start of searching for the smallest item from among
    ↪  the 4
    // elements from index 1 to index 4
    min = 1;
    i=2;
    while(i<5){
        if(arr[i] < arr[min])
            min = i;
        i++;
    }
    printf("The smallest item/element of the array is
↪   %i.\n",arr[min]);
    // end of searching for the smallest item

    // start of swapping arr[1] and arr[min]
    tmp = arr[1];
    arr[1] = arr[min];
    arr[min] = tmp;
    // end of swapping arr[1] and arr[min]

    // start of printing the items of arr
    i=0;
    while(i<5){
        printf("%i ",arr[i]);
        i++;
    }
    printf("\n");
    // end of printing the items of arr
```

```
    // start of searching for the smallest item from among
    ↪  the 3
    // elements from index 2 to index 4
    min = 2;
    i=3;
    while(i<5){
        if(arr[i] < arr[min])
            min = i;
        i++;
    }
    printf("The smallest item/element of the array is
↪   %i.\n",arr[min]);
    // end of searching for the smallest item

    // start of swapping arr[2] and arr[min]
    tmp = arr[2];
    arr[2] = arr[min];
    arr[min] = tmp;
    // end of swapping arr[2] and arr[min]

    // start of printing the items of arr
    i=0;
    while(i<5){
        printf("%i ",arr[i]);
        i++;
    }
    printf("\n");
    // end of printing the items of arr
```

```
    // start of searching for the smallest item from among
    ↪  the 3
    // elements from index 3 to index 4
    min = 3;
    i=4;
    while(i<5){
        if(arr[i] < arr[min])
            min = i;
        i++;
    }
    printf("The smallest item/element of the array is
↪   %i.\n",arr[min]);
    // end of searching for the smallest item

    // start of swapping arr[3] and arr[min]
    tmp = arr[3];
    arr[3] = arr[min];
    arr[min] = tmp;
    // end of swapping arr[2] and arr[min]

    // start of printing the items of arr
    i=0;
    while(i<5){
        printf("%i ",arr[i]);
        i++;
    }
    printf("\n");
    // end of printing the items of arr

    return 0;
}
```

Running the program above should result in the final display of `-2 9 12 23 68`. The smallest item from among the last 2 items left is 23. Once this is found, it is swapped with the fourth element of the array, which, in this case, is 68.

Initially, the elements are `23 12 9 68 -2`, and eventually reconfigured to `-2 9 12 23 68`. The items are now arranged from the smallest to the biggest. This is called sorting! Specifically, this is selection sort. Aptly called so, since at each step, the smallest item is selected and placed in its correct position in terms of sorted order.

And this is the reason why the search area gets reduced every time. Initially, the search area are the 5 elements. Once the smalles is swapped with the first element, the search for the next round is reduced to the last 4 elements (index 1

to 4). We no longer include the first element, since we know it is already the smallest item and it is now stored in the correct position in terms of sorted order. The same goes for the next rounds.

But obviously, our solution is not that elegant at all. What we can deduce from our solution is that the searching for the smallest item, the swapping and the display are repeated many times over. We can then extend this to n elements in general and not just 5.

You might have observed that *min* started with 0 and in the last round of the searching and swapping, it is 3. We didn't proceed with the last element since the solution will not have anything to compare it with (it is the last element, after all). We can simluate this by having an iterator that goes $0 - 3$. And at each iteration, *min* is set to the iterator. And for each iteration, a search for the minimum is executed. This search always begins from the next element after where the iterator is, all the way to the last element.

Let's do this.

```c
#include <stdio.h>
int main(){
    int arr[] = {23, 12, 9, 68, -2}, min, i, j, tmp;
    printf("Enter 10 numbers: ");
    for(i=0; i<10; i++)
        scanf("%i",&arr[i]);
    i=0; // this is the iterator mentioned above
    while(i<4){ //this is the main loop that will perform a
    ↪  number of minimum searches and swaps, including
    ↪  printing the items
        min = i;    //assumes that the smallest item is
↪  found in index i
        // start of the search for the smallest (the next
        ↪  element after where the iterattor is (i+ 1))
        j=i+1;
        while(j<5){
            if(arr[j] < arr[min])
                min = j;
            j++;
        }
        // end of search
        //start of swap
        tmp = arr[i];
        arr[i] = arr[min];
        arr[min] = tmp;
        //end of swap
        //start of print
        j=0;
        while(j<5){
            printf("%i ",arr[j]);
            j++;
        }
        printf("\n");
        //end of print
        i++; //used to go to the next round of searching
↪  and swapping
    }
    return 0;
}
```

We mentioned that we can extend this to n elements. So let's do this. This time, the elements come from the user.

```c
#include <stdio.h>
int main(){
    int arr[10], n = 10, min, i, j, tmp;
    printf("Enter 10 numbers: ");
    for(i=0; i<10; i++)
        scanf("%i",&arr[i]);
    i=0; // this is the iterator mentioned above
    while(i<n){ //this is the main loop that will perform a
    ↪  number of minimum searches and swaps, including
    ↪  printing the items
        min = i;    //assumes that the smallest item is
↪  found in index i
        // start of the search for the smallest (the next
        ↪  element after where the iterattor is (i+ 1))
        j=i+1;
        while(j<n){
            if(arr[j] < arr[min])
                min = j;
            j++;
        }
        // end of search
        //start of swap
        tmp = arr[i];
        arr[i] = arr[min];
        arr[min] = tmp;
        //end of swap

        //start of print
        j=0;
        while(j<n){
            printf("%i ",arr[j]);
            j++;
        }
        printf("\n");
        //end of print
        i++; //used to go to the next round of searching
↪  and swapping
    }
    return 0;
}
```