

# PERFORMANCE, DATA STRUCTURES AND ALGORITHMS

Exercise 07

Exercise 07

Sort an array of doubles

# Name: Minakov Nikita

PURPOSE

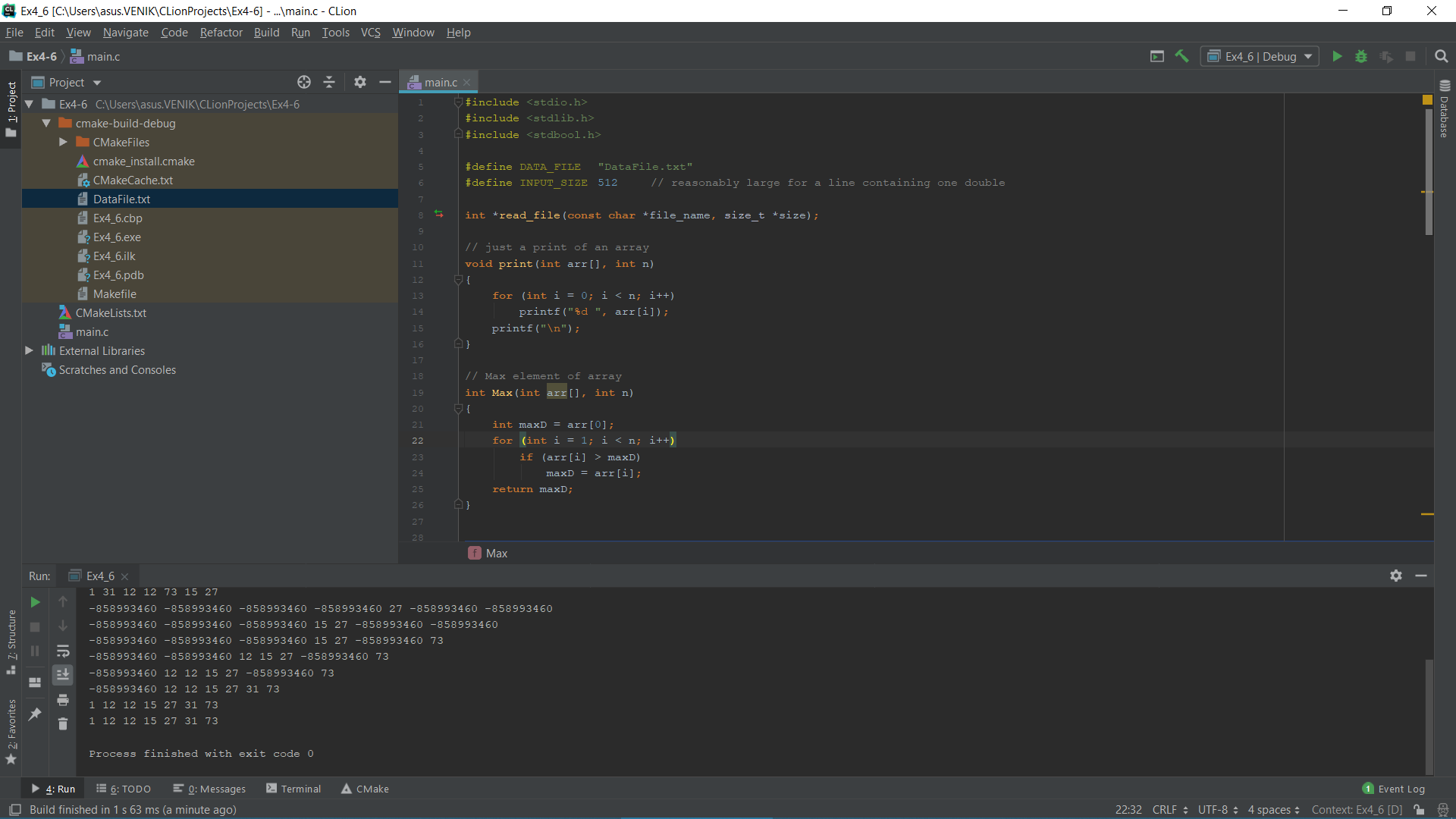
If your program needs to store a limited amount of things (numbers, data, or records for example) a simple and effective approach is to use a list-type data structure. In this project you will implement an array-based simple list. When you have to organize and search through a larger number of things, more sophisticated data structures may be a better choice (covered later in the course).

For this exercise you are given a program that reads a file containing a series of floating point values, each on one line in the file, and stores those values in an array of doubles. You will modify this program to sort the values in the array and to print the sorted array.

ACTIVITIES

Perform each of the following activities. If you have questions, issues, or doubts, please ask for help and do not just guess.

1. Create a function that sorts an array of doubles, in ascending order, using an insertion sort algorithm. The function should take the array to be sorted as a parameter and sort the data in place within the array. It should *not* copy the array.
2. Modify main() to call your sort function and to print the sorted array to the terminal.
3. Create a makefile to build your program.
4. Compile your program using the makefile that you created.
5. Run your program. Copy and paste the output of your program below:
6. Document your code carefully.
7. When you are ready to submit your work, first remove all intermediate files from your src directory.
8. Save and archive your finished C program, your makefile, and this document, and upload them to the LMS.



Code: (Radix sort)

1. #include <stdio.h>  
   #include <stdlib.h>  
   #include <stdbool.h>  
     
   #define DATA\_FILE "DataFile.txt"  
   #define INPUT\_SIZE 512 // reasonably large for a line containing one double  
     
   **int** \*read\_file(**const char** \*file\_name, size\_t \*size);  
     
   // just a print of an array  
   **void** print(**int** arr[], **int** n)  
   {  
    **for** (**int** i = 0; i < n; i++)  
    printf("%d ", arr[i]);  
    printf("\n");  
   }  
     
   // Max element of array  
   **int** Max(**int** arr[], **int** n)  
   {  
    **int** maxD = arr[0];  
    **for** (**int** i = 1; i < n; i++)  
    **if** (arr[i] > maxD)  
    maxD = arr[i];  
    **return** maxD;  
   }  
     
     
   **void** Sort(**int** arr[], **int** n, **int** pw) {  
    **int** out[1000]; // output array  
    **int** i, position[1000] = {0};  
     
     
    **for** (i = 0; i < n; i++)  
    {  
    position[(arr[i] / pw) % 10]++;  
    // Store count of occurrences in count[]  
    //print(position, n);  
    }  
     
    **for** (i = 1; i < 10; i++)  
    {  
    position[i] += position[i - 1];  
    // see the position of each number in digit  
    //print(position, n);  
    }  
     
    **for** (i = n - 1; i >= 0; i--)  
    {  
    out[position[(arr[i] / pw) % 10] - 1] = arr[i];  
    position[(arr[i] / pw) % 10]--;  
    // step by step proses of radix sort  
    print(out, n);  
    }  
     
    // Saving "out" to our current list  
    **for** (i = 0; i < n; i++)  
    {  
    arr[i] = out[i];  
    }  
   }  
   // The main function to that sorts arr[] of size n using  
   // Radix Sort  
   **void** radix(**int** arr[], **int** n)  
   {  
    **int** maxD = Max(arr, n); // Max value of list, needed for radix algorithm  
     
    //print(arr,n);  
    **for** (**int** pw = 1; maxD/pw > 0; pw \*= 10)  
    Sort(arr, n, pw);  
   }  
     
     
     
   // Driver program to test above functions  
   **int** main()  
   {  
    size\_t num; // the number of elements in the array created by read\_file  
     
    // Read in the values to sort from the data file  
    **int** \*arr= read\_file(DATA\_FILE, &num);  
     
    radix(arr, num);  
    print(arr, num);  
    **return** 0;  
   }  
     
   **int** \*read\_file(**const char** \*file\_name, size\_t \*size)  
   {  
    // open the file for reading  
    FILE \*fp = fopen(file\_name, "r");  
    **if** (fp == NULL) {  
    fprintf(stderr, "Unable to open %s for reading\n", file\_name);  
    **return** NULL;  
    }  
     
    // Determine the number of lines in the file (and therefore the number of double  
    // values, since the file is expected to have one value per line) by counting  
    // the lines. We need to know this so we can allocated an appropriate sized  
    // array for the values.  
    **char** input[INPUT\_SIZE]; // space to store the line read from the file  
    **int** num = 0; // number of items read  
    // fgets returns NULL at end of file or if there is an error  
    **while**(fgets(input, INPUT\_SIZE, fp) != NULL) {  
    num ++;  
    }  
    rewind(fp); // move the file pointer back to the start of the file  
     
    // returning the number of values in the file  
    \*size = num;  
     
    // Allocate space for the contents of the file. The size to allocate is the number  
    // of values in the file multiply by the size, in bytes, of each item (in this case,  
    // we are storing doubles in the array)  
    **int** \*array = (**int** \*)malloc(num \* **sizeof**(**double**));  
    **if**(!array) {  
    fprintf(stderr, "Unable to allocate %ld bytes for file buffer\n", (**long**) num);  
    **return** NULL;  
    }  
     
    // Read the file one line at a time and parse the value from the line using sscanf.  
    **int** i;  
    bool have\_error = false;  
    **for** (i=0; i<num && !have\_error; i++) {  
    // Get the line and put it into a string  
    **if**(fgets(input, INPUT\_SIZE, fp) == NULL) {  
    fprintf(stderr, "Unable to read values from %s\n", DATA\_FILE);  
    have\_error = true;  
    }  
    // Parse the value from the input line and store it in the array.  
    // sscanf returns the number of values in the string that match the pattern it is given,  
    // in this case "%lf" for one double value.  
    **if**(sscanf(input, "%d", &array[i]) != 1) {  
    fprintf(stderr, "Line %d of %s does not contain a valid floating point number\n", i+1, DATA\_FILE);  
    have\_error = true;  
    }  
    }  
     
    // close the input file  
    fclose(fp);  
     
    // check for an error - if there was an error parsing the file, free the space and return NULL  
    **if** (have\_error) {  
    free(array);  
    array = NULL;  
    \*size = 0;  
    }  
     
    **return** array;  
   }