# aPECc

## Second Annual Programming Efficiency mini Conference and Challenge

Code Submission for programming challenge is due (deadline) on **March 27, 2017**

Email your cpp and doc file to Daniel.Creider@tamuc.edu

A prize will be awarded for the Level-1 problem
Certificates (1st, 2nd and 3rd place) for winners will be given for all 3 problem levels
*Winning code will be selected based on correct results, fastest execution time (70%) and least amount of temporary memory allocated in your code.(30%)*

**C/C++ Code to Write** (all code submitted must be in C/C++)

**Level-1 Find modes in a large data set – cannot copy or sort the data set**

Write a void function that will find the modes(s) of a data set and store the actual mode values in an array passed to the function. Submit only the code/functions you wrote; do not submit your complete program including the main module you wrote to test your function. There will be at least one mode in the data set but there could also be many modes which occur with the same frequency. The first line of your function definition must be the following. Do not change the order or type of the parameters or your function will not match the call statement in the main module that has been written to test your code.

void mode(long data[], long size, long &numberOfmodes, long modeNumbers[], long &maxFrequency)

same as

void mode(long *data, long size, long & numberOfmodes, long *modeNumbers, long &maxFrequency)

The data that will be used to validate your code will consist of at least 30 million ***random*** values of type long. For information about creating arrays with millions of elements see below. The main code has been written to test your function and will output the results to determine if your code obtained the correct results. **Do not input or output any values/data in the code that you submit**. The main program will pass the address of data array and the number of values stored in the data array to your function. Your function will compute the number of mode(s) in the data set and assign that value to the parameter, *numberOfmodes* (this represents the number of modes you found which also is the number of mode values stored in the *modeNumbers* array); your program will store the actual value of the mode(s) in successive elements of the *modeNumbers* array; and your program will compute the number of times the mode(s) occurred in the data set and assign that value to the parameter, *maxFrequency*. All modes must occur with the same maximum frequency.
You are permitted to use any algorithm of your choice. ***However, for this problem you are not permitted to copy (duplicate or make a copy of all values in the array) the values stored in the data array and you are not permitted to change the contents of the data array in any manner which includes sorting the data array***. The main program will pass the address of the original data set to your function. If you think you need more than one function to solve the problem you will have to put the prototype of a second/third (etc.) function immediately after the first line of the function definition that the main module will call. The main module will only call 1 function. The code you write would look like the following if you write more than one function.

```
void mode(long data[], long size, long &numberOfmodes, long modeNumbers[], long &maxFrequency)
{
```

```
    void secondFunction(parameter list); //prototype for second function if needed
    void thirdFunction(parameter list); //prototype for third function if needed
    // remainder of code in the mode function
}
void secondFunction(parameter list) // possible second function if needed
{
   // code in the second function
}
```

## Level-2 Find and identify all duplicates values

Write a void function that will store a copy of all duplicate values (first occurrence of a number is not a duplicate) found in a large data set and additional information in a structure defined as follows.

```
struct results  // results becomes a type declaration to define variables or arrays of this type
{
   long ElementNo,        // element number or location/offset of the duplicate number in the data set
        Value,            // actual value of the duplicate number
        FirstOccurrence; // element number or location/offset of the first occurrence of the number
};
```

The structure will be used to create an array of the same size as the data set in the event that all values in the data array are identical. A pointer will be used to create a structure array in the main module using the definition above as show in the following statement.

```
results *DuplicateData
```

You are permitted to use any algorithm of your choice. ***However, for this problem you are not permitted to change the contents of the data array in any manner which includes sorting the original data array***. You must maintain the array in its original order and you must be able to identify the location/offset of the first occurrence of any value determined to be a duplicate. The main program will pass the address of the original data set to your function. If you think you need more than one function to solve the problem you will have to put the prototype of a second/third (etc.) function immediately after the first line of the function that the main module will call as shown in the description of the Level-1 problem.. The main module will only call 1 function. The first line of your function definition must be the following. Do not change the order or type of the parameters or your function will not match the call statement in the main module that has been written to test your code.

```
FindDuplicatesFast(long data[], long size, results DuplicateData[], long &sizeDD)
                                        same as
FindDuplicatesFast(long *data, long size, results *DuplicateData, long &sizeDD)
```

The data that will be used to validate your code will consist of at least 30 million ***random*** values of type long. For information about creating arrays with millions of elements see below. The data set may contain no duplicate values or all values in the data set may be the same, or may contain 1 to many duplicate values (any value could also occur multiple times). The main code has been written to test your function and will output the results to determine if your code obtained the correct results. **Do not input or output any values/data in the code that you submit**. The main program will pass the data array and the number of values stored in the data array to your function. Your function will find all the duplicate values in the data array and store the required information in the structure array passed to the function. You must also assign to the last argument of this function the number of entries you stored in the structure array. The argument ***sizeDD*** must contain a value that corresponds to the number of elements in the structure array that have been assigned a value (number of values stored in the structure array).

## Level-3 Identify and remove all duplicates values

Write a void function to remove all duplicate values from a data set while maintaining the order of the remaining values. This is called a stable algorithm. The first occurrence of a value is not considered to be a duplicate value. Any and all occurrences of any value after the first occurrence must be removed. You are permitted to use any algorithm of your choice. ***However, for this problem you are not permitted to change the order of the data remaining in the array***. . The main program will pass the address of the original data set to your function. If you think you need more than one function to solve the problem you will have to put the prototype of a second/third (etc.) function immediately after the first line of the function that the main module will call as shown in the description of the Level-1 problem.. The main module will only call 1 function. The first line of your function definition must be the following. Do not change the order or type of the parameters or your function will not match the call statement in the main module that has been written to test your code.

RemoveDuplicatesFast(long data[], long &size)
same as
RemoveDuplicatesFast(long *data, long &size)

The data set that will be used to validate your code will consist of at least one million ***random*** values of type long. For information about creating arrays with millions of elements see below. The data set may contain no duplicate values or all values in the data set may be the same, or the data set may contain 1 to many duplicate values (any value could also occur multiple times). The main code has been written to test your function and will output the results to determine if your code obtained the correct results. **Do not input or output any values/data in the code that you submit**. The main program will pass the data array and the number of values stored in the data array to your function. Your function will find all the duplicate values in the data array and remove them from the original data set. You must also assign to the last argument of this function, ***size,*** a value which is the number of values remaining in the array after the duplicates have been removed.

## General Information

If you have any questions about the contest and or about the code you are to write contact Dr. Creider at the email address/phone number listed at the end of this document. Your submission to this programming challenge will be disqualified if your code requires an excessive amount of time to execute. In order not to discourage you in entering this contest contact Dr. Creider after you have written the code with an estimate of the amount of time your code requires.

It is possible that you will not be able to create a compile time array that has millions of elements so the code below describes how to create a pointer and dynamically allocate an array of type long (could be an array of any type including a structure array) of the number of elements corresponding to the number of data values stored in the array.

```
long *dataSet, size;  // pointer needed to dynamically allocate memory
dataSet = new(nothrow)long[size];  // statement to dynamically allocate the data array to have size elements
if(!dataSet)  // this statement will test the pointer to determine if memory was dynamically allocated
  { cout<<"Memory allocation error for data array, program will terminate\n";
    system("pause");  // this is a windows command to pause the program, may need the windows.h header
    exit(0); }    // this statement will terminate the  program if memory was not allocated
```

After memory is allocated and the values are stored in the array you can use the pointer *dataSet* as you would a compile time array called *dataSet*. Subscripts are permitted to be used with the variable *dataSet* but not required. Such as - cout<<dataSet[i]<<endl; You can use either pointers or subscripts in your code.

You will need to time your code and report the results when you submit your code for the programming challenge. You can time your code with a built in function in C/C++ and the code to do this follows. The following variables need to be declared.

```
double duration; // variables used to time the code
clock_t start,stop;  // clock_t is a valid type
```

You might need to include the time header file in your code: #include<time.h>
Additional statements are used to time the function you write for the programming challenge.

```
start=clock(); // Start the benchmark
// call the function that is to be timed, such as following example
RemoveDuplicatesFast(dataSet, size);
stop=clock();  // stop the clock
duration = double(stop-start)/CLK_TCK; // compute/record the time
cout<<"time to execute the mode "<< duration <<endl;  // output the time
```

## Files to submit no later than March 27, 2017

Create a Word document to submit at the same time you submit the cpp file. This file must contain your name, your CWID, your class standing (undergraduate or graduate), your email address and the problem you selected. In this document, briefly describe the algorithm you used to solve the programming challenge. Also, list the time it took your program to execute your code and the number of values in your dataset. Name your Word file with your name as follows – "LastNameFirstName" such as CreiderDaniel. Use the same naming convention for your cpp file and be sure to put your name in comments in the cpp file. Do not forget that there is a deadline for submission of your code.

When you submit, you are certifying that you wrote the code that was submitted and you did not submit code written by someone else and put your name on it. You are permitted to search the Internet for ideas if you need to do so. You can use code found on the Internet to help solve this problem. (It is very unlikely that you will find the exact code needed to solve any of these problems on the Internet.) If you have any questions about the programming challenge, send Dr. Creider an email or see him in person in his office. All questions about this programming challenge should be directed to Dr. Creider.

If you are going to attempt the programming challenge and submit code send an email containing your name, major, class standing and the Level of the problem(s) you will attempt. Send the email to Dr. Dan Creider - Daniel.Creider@tamuc.edu - Office phone - 903-886-5407 - Jour 216.