

THE UNIVERSITY OF WESTERN ONTARIO

DEPARTMENT OF COMPUTER SCIENCE
LONDON CANADA

Software Tools and Systems Programming (Computer Science 2211a)

ASSIGNMENT 2 (amended - September 29, 2021 in red text)

Due date: Wednesday, October 06, 2021

11:55 pm Eastern Daylight Time – 3:55 am Greenwich Mean Time)
allow up to one day late ONLY – assignment closed Oct. 07, 2021 11:55pm EDT : 3:55am GMT

Assignment overview

The purpose of this assignment is to provide the student with experience with arrays. This assignment will further provide experience using the basic code control structures covered in class.

PREPERATION:

For this assignment, create a new directory under the [assignments](#) directory created in the first assignment. Label this new directory: [asn2](#)

All work should be performed in this directory. Use a UNIX editor like vi to create and compile the C code.

The code MUST compile in the UNIX environment to be considered correct.

note: this is not a trivial assignment. Start early. Break this down into discrete doable steps. You will also have the benefit of much easier access to the TAs since you will be beating the typical last minute rush.

Again, this assignment will be contained within a single main.c program.

Synopsis:

This program will generate an array of a randomly determined length.

It will populate that array with random numbers within a given range.

The program will then cycle through the array and remove duplicate values.

When a duplicate value is removed, all the remaining elements in the array will be shifted over to the left one place and the final shifted value location will be filled with a zero (0) value.

Once all the duplicated values are removed, the remaining values will be sorted in ascending order. The elements of the array will then be outputted to the screen.

The above process will loop and repeat all but the first step for a specified number of iterations.

EXAMPLE:

If the original array consisted of the values:

| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 3 | 6 | 4 | 7 | 4 | 5 | 8 | 3 | 5 | 2 |

The program will start at index [0] and traverse the remainder of the array searching for duplicate values (in the case above the value 3).

If a duplicate is found (in this case at index [7], then all the remaining values in the array will be shifted to the left and the last value (index [9] in this case) will be replaced with a zero.

| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 3 | 6 | 4 | 7 | 4 | 5 | 8 | 5 | 2 | 0 |

The next duplicate value is 4 at index [4]. all the remaining values in the array will be shifted to the left and the last value (index 8 in this case) will be replaced with a zero.

| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 3 | 6 | 4 | 7 | 5 | 8 | 5 | 2 | 0 | 0 |

This process will continue until all the elements are checked and no duplicates remain.

| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 3 | 6 | 4 | 7 | 5 | 8 | 2 | 0 | 0 | 0 |

Once all the duplicates are removed only unique values remain. The total number of unique elements in the example above are 7 (the array size is still 10).

This process will then sort only the unique, non-zero values in ascending order.

| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 2 | 3 | 4 | 5 | 6 | 7 | 8 | 0 | 0 | 0 |

Size of the array:

The initial size of the array will be based on a random number between 50 and 50 times the number of iterations. This one array is used for the remainder of the program.

Example: if the number of iterations is set at 4 then the range would be from 50 to 200.

if the number of iterations was 10 then the range would be from 50 to 500.

Populating the elements of the array:

The elements of the array will be populated with random numbers based on the value of the current iteration. The range will be decreased by a factor of 10 for each iteration.

For the last iteration, the values will be in the range of 1 to 10 inclusive.

For the second last iteration, the values will be in the range of 1 to 100 inclusive.

For the third last iteration, the values will be in the range of 1 to 1,000 inclusive.

For example, if the number of iterations is 6 then the first time through the range will be from 1 to 1,000,000 (six zeros), the second time through this will be reduced by a factor of 10, so the fifth iteration will be in the range of 1 to 100,000 (five zeros). The fourth iteration will be in the range of 1 to 10,000 (four zeros) and the last iteration, regardless of the total number of iterations, will always be in the range of 1 to 10.

The number of iterations of the program section outlined above will be denoted by a preprocessor definition with the label of: [ITERATIONS](#).

Debug Statements:

The program is to contain at least two debug statements.

There is to be one debug statement displaying the upper range value of the random number generator for the values being inserted into the array:

```
-----  
THIS IS ITERATION NUMBER: 2 of 3  
Upper Range Value: 100  
This is the original array populat
```

This is to appear immediately after the line showing the current iteration of the program.

The other debug is to display each time when matching values are found along with the indexes of the matching values.

```
77 14 16 11 72 25 53 17 74 44  
The value of 62 at array index [0] and the value of 62 at array index [33] are the same.  
The value of 98 at array index [1] and the value of 98 at array index [95] are the same.  
The value of 29 at array index [2] and the value of 29 at array index [9] are the same.  
The value of 29 at array index [2] and the value of 29 at array index [11] are the same.  
The value of 29 at array index [2] and the value of 29 at array index [34] are the same.  
The value of 27 at array index [3] and the value of 27 at array index [42] are the same.  
The value of 27 at array index [3] and the value of 27 at array index [78] are the same.
```

All debug statements are to be included in your final copy of the program you submit but must not be compiled or executed. [They cannot be commented out.](#)

Random Number Generator:

```
#include <time.h>  
#include <stdlib.h>
```

```
srand(time(NULL)); // Initialization, should only be called once.
```

```
int r = rand() % 10; // Returns a pseudo-random integer between 0 and 9 inclusive.
```

The code above will generate a random number between 0 and 9.

Use the rand() function to generate the value for the number of array elements in your array. The one array will be used throughout the program in all the iterations and no other arrays are to be generated or used. The <time.h> library must be included in your code for this to compile.

The rand() function will also be used to populate the values of the elements in the array.

Output:

The following are screen shots displaying the exact output and words expected:

```
Assignment Two

Value of random size of array: 187
Size of array: 748 bytes

THIS IS ITERATION NUMBER: 1 of 3

This is the original array populated with values in the range of 1 and 1000
Number of elements in the original array is: 187

721 310 849 326 685 42 953 962 922 803 348 199 552 314 125 260
854 287 389 591 869 294 195 411 393 496 212 67 284 383 754 174
975 9 435 905 630 213 56 10 688 589 60 969 566 328 597 787
407 577 772 644 922 841 59 631 700 988 172 340 2 988 690 547
598 279 104 155 725 624 848 81 666 673 918 189 340 502 508 156
521 926 578 316 211 77 952 399 83 404 433 182 448 340 381 831
388 368 648 853 885 486 384 884 853 888 588 584 388 588 856 884
```

The program will display the assignment number.

It will then print out the randomly determined size of the array to be used throughout the rest of the program.

Next it will determine how many bytes this array will use based on the architecture of the O/S the program is currently being executed.

Then, for each iteration, the program will display the iteration, the range of the random numbers populating the array and the number of elements of the array (it will always match the size of the array – this is a good way to show the program is functioning correctly).

The program will then print out the elements of the array evenly spaced apart.

After the program has removed all duplicate values, the program will display the total number of unique, non-zero values of the array, followed by a printout of all these values, again evenly spaced out.

```
This is the current state of the array with all duplicate values removed
Number of unique (non-zero, non-duplicate) elements in the array is: 166

546 607 481 751 703 291 829 655 382 174 650 346 466 135 474 192
796 808 990 97 122 559 215 897 862 529 445 878 866 342 206 493
158 322 925 80 672 268 548 821 96 130 639 38 415 369 908 722
448 83 792 220 201 647 745 979 297 417 412 280 512 143 835 774
623 392 200 711 207 213 439 504 849 667 340 106 348 625 765 589
25 433 846 676 994 6 822 735 246 889 787 561 760 64 384 304
534 827 73 524 744 732 648 77 886 285 194 476 776 783 996 551

These are the unique non-zero elements in the array sorted in ascending order
```

Finally, the program will sort the array in ascending order and then display the unique, non-zero elements of the array in sorted order, evenly spaced.

```

529 599 256 891 203 981 487 636 323 764 52 697 68 553 917 944 838
66 982

These are the unique, non-zero elements in the array sorted in ascending order :
2 9 10 42 49 52 56 59 60 66 67 68 72 77 78 81 83
126 133 155 156 172 174 182 189 195 199 203 204 205 211 212 213 220
282 284 286 287 294 306 310 314 316 323 326 328 332 337 340 348 368
399 404 406 407 411 433 435 448 450 482 483 484 487 496 498 502 508
550 552 553 566 568 577 578 583 589 591 597 598 599 604 624 630 631
673 681 685 688 690 697 700 701 703 721 725 754 764 772 777 787 790
869 878 891 902 905 917 918 921 922 926 927 940 944 952 953 954 962
988 989

-----
THIS IS ITERATION NUMBER: 2 of 3

```

The program will then proceed to loop and repeat the process for the designated number of iterations, populating the array with a new set of numbers within the range that is based on the value of the current iteration.

Number of iterations:

The program is to be written to handle any number of iterations.

The program is to be submitted with the number of iterations set to three (3) but the code will be tested with different iterations (~~up to 20~~). (~~up to 5~~ – amended Sept. 29, 2021).

Example: Iteration 2 out of a set of 3 iterations of the program (without the debug statements):

```

640 650 661 662 670 670 670 710 717 727 737 738 742 752 754 761 763 763 766 1000

-----
THIS IS ITERATION NUMBER: 2 of 3

This is the original array populated with values in the range of 1 and 100
Number of elements in the original array is: 175
48 88 55 11 17 93 99 58 91 14 56 68 48 48 38 64 15 84 94 70 65 2 55 7
28 47 88 7 29 24 87 96 93 32 70 91 98 32 60 36 91 46 86 79 46 85 6 68
79 55 28 6 21 26 55 7 54 67 78 98 82 80 8 15 40 16 48 64 22 13 17 96
98 73 10 60 29 85 93 87 83 38 85 78 67 16 28 84 40 57 10 40 92 50 43 89
16 40 51 69 61 46 28 29 57 57 87 48 77 40 83 94 41 25 54 46 80 31 50 79
58 25 23 72 91 24 77 55 82 84 93 55 29 77 15 88 71 15 3 90 8 42 96 16
56 52 79 48 47 99 28 24 10 92 43 79 25 92 3 100 25 12 43 41 88 48 7 16
37 76 56 3 28 43 91

This is the current state of the array with all duplicate values removed
Number of unique (non-zero, non-duplicate) elements in the array is: 74
48 88 55 11 17 93 99 58 91 14 56 68 38 64 15 84 94 70 65 2 7 28 47 29
24 87 96 32 98 60 36 46 86 79 85 6 21 26 54 67 78 82 80 8 40 16 22 13
73 10 83 57 92 50 43 89 51 69 61 77 41 25 31 23 72 71 3 90 42 52 100 12
37 76

These are the unique, non-zero elements in the array sorted in ascending order :
2 3 6 7 8 10 11 12 13 14 15 16 17 21 22 23 24 25 26 28 29 31 32 36
37 38 40 41 42 43 46 47 48 50 51 52 54 55 56 57 58 60 61 64 65 67 68 69
70 71 72 73 76 77 78 79 80 82 83 84 85 86 87 88 89 90 91 92 93 94 96 98
99 100

-----
THIS IS ITERATION NUMBER: 2 of 3

```

Required Coding Standards

All code is to be indented correctly.

Comments at the very beginning (top – first lines) of each of the C code files must be:

```
/* CS2211a 2021 */
/* Assignment 02 */
/* your name */
/* your student number */
/* your UWO User Name */
/* Date Completed */
```

Your program is to be submitted as C code file.

Your script will be a script file created in UNIX.

All variables **MUST** have a comment describing their intended use(s).

A comment describing the code for each section / operation must be included.

The comment(s) can be brief but must convey what that section of code performs.

ALSO – very important standard!

You are **NOT** to use the code

break

-or-

continue

anywhere in your code (except if you have a `switch` statement in your code).

This is bad coding practice and leads to ‘spaghetti code’ (with apologies to my Italian heritage...) where program execution control can go any which where.

We do not want to get into the habit of using these constructs. All the code can be written in a fluid manner without terminating the processes with a `break` or `continue`.

Working in UNIX.

Save this file and name it: **yourUserName_asn2.c**

(see the end of this document for a detailed explanation of **yourUserName**).

NEXT: Follow the steps below to complete Part 2.

1. Type the following to begin recording your session in a file called **yourUserName_asn2.output**

script YourUserName_asn2.output

note - (using your actual user name).

2. Display the current date and time using the appropriate command

3. Display your username using the appropriate command

4. Display the contents of the current working directory using the ‘l’ switch (lower case L).

5. Display the contents of the file `yourUserName_asn2.c` (i.e. show the C program)
6. Compile the program again ensuring the executable is labeled: `asn2`
7. Run the program.
8. Type **exit** to stop your screen capture session.
9. Compress the listed required submission files into a single .tar.gz file.
(Ensure have a complete of the Asn2_SubmissionForm in your asn2 directory.)
`YourUserName_asn2.c`
`YourUserName_asn2.output`
`Asn2_SubmissionForm.txt` (or .pdf)
10. Copy (i.e. using an FTP or any method of your choice) the .tar.gz file to your computer so you can upload it through OWL for submission.

Submission Instructions:

Complete the *CS2211b Assignment Submission Form* Name (or rename) that form to **Asn2_SubmissionForm.txt (or Asn2_SubmissionForm.pdf)**

Save this form in your asn2 directory as a text file or as **PDF** (most word processors have this option).

Submit via the CS2211 OWL Web Site the files in your asn2 directory using the instructions on how to compress and submit the single compressed .tar.gz file.

Submit via the CS2211 OWL Web Site the following three files inside your compressed submission file:

`YourUserName_asn2.c`
`YourUserName_asn2.output`
`Asn2_SubmissionForm.txt` (or .pdf)

note: do NOT include your executable file: `asn2`

note: you are submitting one tar file (and only one file).

this is a compressed file that contains the files listed above.

do NOT send any of the three above files separately.

note: If you have elected to use an SRA – it must be denoted in the Asn2_SubmissionForm or the SRA will not be applied.

note: marks will deducted if the Asn2_SubmissionForm is omitted.

It is the student's responsibility to ensure the work was submitted and posted in OWL.
OWL replies with a summation verification email (every time).

Submission date and time is based on the last file submitted. So if you re-submit after the due date, the entire assignment will be graded as late based on that timestamp.

The teaching assistant grading your assignment will compile and run your program.

If the program does not compile under UNIX, the TA will NOT attempt to correct or fix your program so it will run.

(*yourUserName* - example: assume my UWO email is kdoit373@uwo.ca
i.e. if my email is – **kdoit373@uwo.ca** then my user name will be – **kdoit373**
So, my UWO User Name is: **kdoit373** and this assignment is **asn1**
therefore, one of the file names that is to be used for submission is:
kdoit373_asn1.c

It is the student's responsibility to ensure the work was submitted and posted in OWL.
OWL replies with a summation verification email (every time).

Any assignment **not** submitted correctly will **not** be graded.

PS: remember: do your own work – you will need to know all this for the exam !!!!

Please check CS2211 Assignment Submission Guidelines.

note: Guidelines mention the example for assignment 1 – please substitute a 2 for the 1.

note: Guidelines mention a top directory name **Asn** – ignore that and use **assignments** as specified in this (and the last assignment).