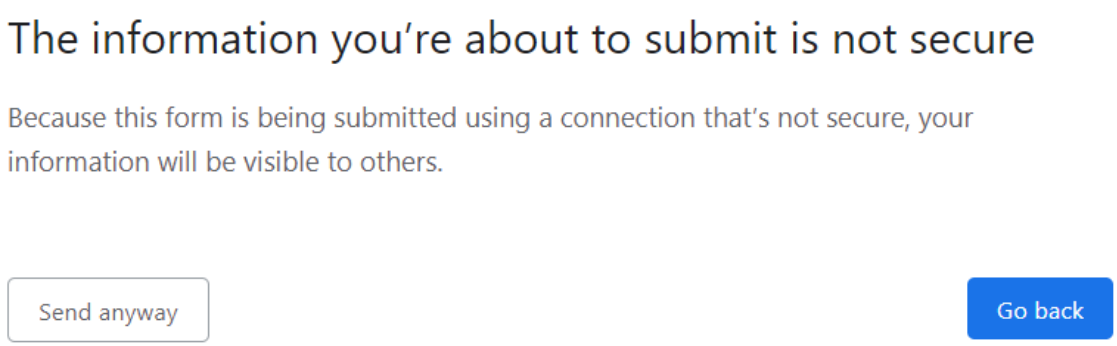
Tutorial 8: Using Arrays to Solve Problems

Topics and References

- Safe Exam Browser (SEB). See Task 1 to set up and test SEB on your machine.
- Arrays. See the section titled "Brief Review of Arrays" for more information about arrays. Chapter 8 and Section 12.3 of the text contain information on arrays.
- Implementing functions with array parameters. See this page for information about functions. Chapter 9 of the text contains information on functions.

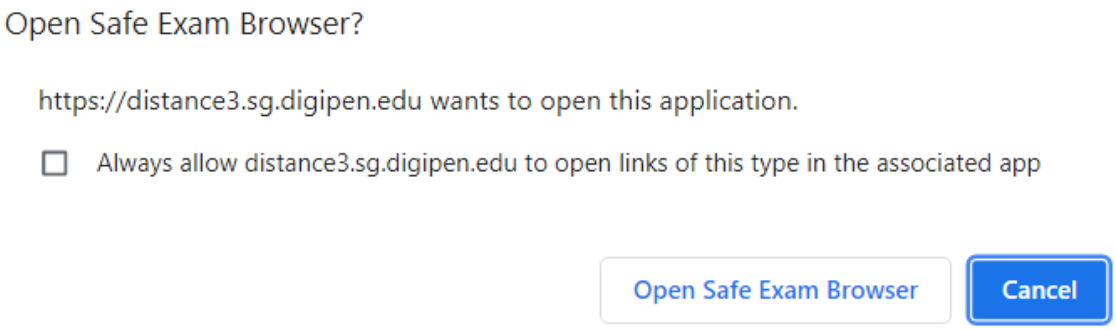
Task 1: SEB Test

1. To use SEB for the midterm test, go to the Moodle section for this week and click on 'Midterm Test - SEB Check'.
2. Select 'Download Safe Exam Browser'.
3. After the page loads go to the section that applies to your machine. For Windows 10, look for 'Download from GitHub' and click on the word 'Download'. For other OS's choose accordingly.
4. Follow the installation process for your machine.
5. Ensure that your system has only one display device in use.
6. In Moodle click on the button 'Launch Safe Exam Browser'.
7. 



If the above message appears click on 'Send anyway'.

8. If the message below appears click on 'Open Safe Exam Browser'.



9. If you are notified of other apps about to be closed, allow it. This includes MS Teams.
10. The Moodle login screen will reappear. Login again and proceed to complete the mock quiz under 'Midterm Test - SEB Check'. Submit your answers to check the mock quiz is working.
11. To exit from SEB, click on the power button on the bottom right corner of SEB.
12. Rejoin the Teams meeting.

Task 2: Computing Summary Statistics

1. We'd like to print the summary statistics (maximum, minimum, and average grades) for a certain course section that will have a *maximum* of 100 students. That is, the number of students enrolled in this section fluctuates but can never be more than 100. Conveniently, grades for the latest offering of the course are stored in a text file. Inconveniently, the number of students enrolled in the course is unknown and can only be determined by the number of grades in the data file. Atypically, this course has minimum and maximum grades of 0 and 1000, respectively.
2. A function `read_ints_from_stdin` is required to read grades from a text file which will be redirected to `stdin` by the user. Functions `maximum`, `minimum`, and `average` must be implemented to print the summary statistics for the course. See the sample input and output files for reference.
3. If you're confident about arrays and can devise algorithms to sum the values in an array, and to determine the maximum and minimum values in an array, you can proceed with the assignment. Otherwise, you may wish to read the review of arrays.

Deliverables and Submission

1. You are given file `qdriver.c` which includes function `main`. You'll not be submitting `qdriver.c`. This means that any changes you make to `qdriver.c` will not be seen by the grader. Repeat: The grader will use their own copy of `qdriver.c` and not yours. In fact, you must not include `qdriver.c` for submission.
2. Notice that `qdriver.c` includes header file `q.h`. This header file will contain the *declarations* of the functions that you're to implement. You must author this file. Use the driver `qdriver.c` to determine the declarations of functions `read_ints_from_stdin`, `minimum`, `maximum`, and `average`. Without this file, you (and the grader) will not be able to successfully compile and link your submission. Start by adding a file header that looks like this:

```
/*!
@file     highbury-lane
@author   Nicolas Pepe  (nicolas.pepe@digipen.edu)
          Rob Holding   (rob.holding@digipen.edu)
          Mezut Ozil    (mezut.ozil@digipen.edu)
@course   CS 999
@section  Z?
@tutorial Tutorial 1
@date     02/02/2020
@brief    This file contains code that puts a salmon on a cedar plank
          and smokes the fish for three hours.
*//*_____________________________________________________________________*/
```

Since this file is the interface that other programmers will use to understand your library, you must provide *function-level* documentation for every function. The documentation should consist of a description of the function, the inputs, and return value. This will provide your clients an opportunity to understand the details necessary for them to use functions from your library. In general, recall that your clients will not have access to the source file where you defined these functions (because they're your intellectual property and you don't want others to steal that property). Instead, you will provide clients the header file and a binary library file containing your implementation in machine language. An example of "function-level" documentation is provided below for a function `reverse32()` that returns the byte order of an unsigned 32-bit integer.

```
/*!
@brief Reverses the byte order of an unsigned 32-bit integral value.

This function takes as input a 32-bit integer and reverses the byte
order of the full word. That is, given the value 0x12345678, the
function will return the value 0x87654321.

@param word - the 32-bit word to be reversed.
@return the reversed 32-bit value of word
*//*_____________________________________________________________________*/
unsigned int reverse32(unsigned int word);
```

3. You must author a file called `q.c` that will contain the definitions of the functions that you're to implement. Obviously, these functions must be implemented exactly as you prototyped them in `q.h`. As is required and necessary, this file must contain appropriate file and function headers.
4. The implementation must cleanly compile for the C11 standard with the full suite of warning options. The command line to build this assignment will look like this:

```
gcc -std=c11 -pedantic-errors -Wall -Wextra -Wstrict-prototypes -Werror
qdriver.c q.c -o q.exe
```

5. You are given a number of sample input files `ip*.txt` and corresponding output files `op*.txt`. For each input file `ip*.txt`, your output must exactly match the contents of the corresponding `op*.txt` file. Use the input and output redirection operators to redirect the contents of input file `ip*.txt` to standard input and redirect your program's output to standard output.

```
./q.exe < ip49.txt > myop49.txt
```

6. Use the `diff` command to compare your implementation's outputs with the correct output provided to you, like this:

```
diff -y --strip-trailing-cr --suppress-common-lines myop49.txt op49.txt
```

   The option `-y`, `--strip-trailing-cr`, and `--suppress-common-lines` are described here. If `diff` completes the comparison without generating any output, then the contents of the two files are an exact match. If `diff` reports differences, you must go back and amend your code to remove these differences.

7. You must submit source file `q.c` and header file `q.h` in Moodle.

Reading unknown number of values from stdin

Suppose that a source file `main.c` contains the definition `int grades[MAX_STUDENT_COUNT];` with the macro declared as `#define MAX_STUDENT_COUNT 100`. Further suppose, that we wish to implement a function `read_ints_from_stdin` in this source file to read an unknown number of `int` values from standard input up to a maximum of `MAX_STUDENT_COUNT` number of values. The source file with a `main` function that exercises function `read_ints_from_stdin` is shown below:

```c
#include <stdio.h>

// declaration of funtion read_ints_from_stdin ...
int read_ints_from_stdin(int array[], int max_num_vales);

#define MAX_STUDENT_COUNT 100

int main(void) {
  // fill grades with values
  int grades[MAX_NUM_GRADES];
  int count = read_ints_from_stdin(grades, MAX_STUDENT_COUNT);
  // print grades to standard output
  for (int i = 0; i < count; ++i) {
    printf("%i: %i\n", i, grades[i]);
  }

  return 0;
}
```

Using the framework provided for writing functions with array parameters, the function `read_ints_from_stdin` can be declared as

```c
/*
Read values of type int from standard input and assign them - in sequence -
to indexed variables in array.
Fill the array with no more than max_array_size number of int values or until
end-of-file is reached, whichever comes first.
Notice that array is NOT declared as a read-only array since we want to
assign or write int values to the array.
*/
int read_ints_from_stdin(int array[], int max_array_size);
```

Using idioms developed throughout the term, the definition of `read_ints_from_stdin` is straightforward:

```c
/*
Here, the parameter array is non-const because the function's intention
is to copy int values from standard input to the array's index variables.
*/

int read_ints_from_stdin(int array[], int max_array_size) {
  int i = 0, num;
  while ( (scanf("%d", &num) != EOF) && i < max_array_size) {
    /*
    Rather than writing two separate statements:
    array[i] = num;
    ++i;
    we use the post-fix increment operator to assign the value read from
    standard input to the indexed variable array[i] and then let the post-fix
    increment operator increment index i's value to i+1.
    */
    array[i++] = num;
  }

  /*
  i specifies the number of int values read from standard input -
  the only thing we care about is that a maximum of max_array_size number
  of elements have been read.
  */
  return i;
}
```