1 Objectives

The objectives of this lab are:

1. getting more practice with C in general;
2. getting more practice with writing and calling your own functions;
3. getting more practice with memory allocation and deallocation;
4. gaining some experience with detecting integer overflow.

2 Requirements

Big picture: You will be filling up a dynamically allocated array with potentially large random numbers. You will then use different approaches to add up the numbers in the array.

2.1 Program Description/Requirements

This program generates an array of random numbers. **After the array is allocated and filled with numbers**, a function shall be called to display each element of the array, as well as the subtotal up to that point.

- The first column of the output will display each random number.
- The second column shall display the running total **with** overflow detection in an `unsigned int`.
- The third column will display the subtotal **without** rollover detection in an `unsigned int`.
- The fourth column will display the subtotal without rollover detection in an `unsigned long long`.

For example, assume a 3-element array was populated as shown below:

49066129
87851954
7065299

In this case, the output would look like the following:

```
                      unsigned       unsigned       unsigned
                           int            int      long long
                      Subtotal       Subtotal       Subtotal
       Random      w/ Rollover   w/o Rollover   w/o Rollover
       Number        Detection      Detection      Detection
-------------    -------------  -------------  -------------
     49066129         49066129       49066129       49066129
     87851954        136918083      136918083      136918083
      7065299        143983382      143983382      143983382
```

Each column shall be right-justified as shown in the example to allow for a 13-digit number.

When calculating the subtotal for the second column, if adding a number to the current total would cause a rollover, then the output shall signify that. When printing the output, it shall look like the following (including the red text):

[prior numbers not shown]
```
     3163711    3959034731    3959034731    3959034731
    76146772    4035181503    4035181503    4035181503
    29027890    4064209393    4064209393    4064209393
   115020362    4179229755    4179229755    4179229755
   100109211    4279338966    4279338966    4279338966
   123456789      REJECTED     107828459    4402795755
   100109211      REJECTED     207937670    4502904966
           3    4279338969     207937673    4502904969
   109822771      REJECTED     317760444    4612727740
```
[continue the display of all numbers]

2.2 Additional Requirements and Details

1. The name of the code file shall be "rollover.c".
2. Use `srandom/random` for the random number generation.
3. The number of elements of the array is chosen randomly, from a minimum size of 1 element up to a maximum size of 100 elements. Do not use a linked-list approach – allocate an array using malloc, based on the number of elements needed.
4. The array of random numbers must be allocated dynamically using `malloc`.
5. Each element of the array shall be an "unsigned int".
6. The variables holding the running totals for the second and third columns shall be declared as "unsigned int", while the variable holding the running total for the fourth column shall be declared as "unsigned long long".
7. Limit the maximum size of each random number stored in the array to a number **less than** 1,234,567,891.
8. You shall not use global variables.
9. You shall call a function called `display` that will display the results on the screen as described. Only one call to this function is allowed. The function prototype shall be:
   ```
     void display(unsigned int values[], unsigned int num_values);
   ```
   where "values" is the array of random numbers, and where "num_values" communicates the number of elements in the array.
10. Free the allocated memory properly before the program terminates.

2.3 Help

1. If you "#include <limits.h>", you get access to UINT_MAX, which #defines the largest number that an unsigned integer can hold in our virtual environment.
2. The proper way to test for a rollover is given in Unit 7, slide 7, under "proactive approach". THE_LARGEST_REPRESENTABLE_TYPE shown in the slide is UNIT_MAX.

3. Once again, you will need to use the following to change font colors:
   ```
   #define CHAR_RED "\033[31m"
   #define CHAR_RESET "\033[0m"
   ```
4. When displaying information, `printf` allows you to specify how much space the information should take, and whether it should be right-justified or left-justified. Here are a couple of examples:
   a. To print a number such that it will be right-justified within 15 spots on the screen:
      ```
      printf("%15d\n", x);
      ```
   b. To display right-justified text within 15 spots on the screen:
      ```
      printf("%15s\n", "Hello World”);
      ```
   c. An example of a left-justification request is `"%-15d"`.

2.4 Makefile

Submit a Makefile with the same kind functionality as described in earlier projects, with the following targets: all, rollover, clean, dist.

3 Submission

Submit proj4.tar to Sakai by the deadline.

4 Grading

Your grade shall be based on the following **guidelines**:

1	Makefile works properly	5	
2	Compiles with no warnings:	5	(An error = **significant** deductions)
3	No program crashes	10	
4	Proper formatting of output:	5	
5	Proper memory & pointer management	30	
6	Valid output	30	
7	Style guide followed properly	5	
8	All other code requirements met	10	

I reserve the right to deduct points when it appears appropriate.