COMP1511 18s2

Week-05 Tutorial Exercises

1. The tutorial will start with a code review.
   Your tutor has asked a lab pair to present their week 04 work.

   Discuss the good, the bad and the ugly aspects of their code.

   Please be gentle in any criticism - we are all learning!

2. (For the Monday tutorials, your tutor will explain the required concepts for this question.)
   In the following program, what are `argc` and `argv`? The following program prints number of command-line arguments and each command-line argument on a separate line.

   ```
   // print command line argument
   // Andrew Taylor - andrewt@unsw.edu.au
   // 24/4/18

   #include <stdio.h>
   #include <stdlib.h>

   int main(int argc, char *argv[]) {
       int i;
       printf("argc=%d\n", argc);
       i = 0;
       while (i < argc) {
           printf("argv[%d]=%s\n", i, argv[i]);
           i = i + 1;
       }
       return 0;
   }
   ```

   What will be the output of the following commands?

   ```
   % dcc -o print_arguments  print_arguments.c
   %
   % print_arguments  Sydney Olympic 2000
   ```

3. (For the Monday tutorials, your tutor will explain the required concepts for this question.)
   The following program sums up command-line arguments. Why do we need the function `atoi` in the following program? The program assumes that command-line arguments are integers. What if they are not integer values? See `strol` for a more powerful library function which would allow checking.

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

   int main(int argc, char *argv[]) {

       int sum = 0;
       int argument = 1;
       while (argument < argc) {
          sum = sum + atoi(argv[argument]);
          argument= argument + 1;
       }
       printf("sum of command-line arguments = %d\n", sum);

       return 0;
   }
   ```

4. A Caesar cipher shifts each letter a certain number of positions in the alphabet.

   1. Encode and decode a message with a Caesar cipher.
   2. The shift is the key for a Caesar Cipher - how many bits are in it?
   3. How would you crack a Caesar Cipher?

5. Write a program `sum_digits.c` which reads characters from its input and counts digits.
   When the end of input is reached it should print a count of how many digits occurred in its input and their sum.

   The only functions you can use are `getchar` and `printf`.

For example:

```
$ ./sum_digits
1 2 3 o'clock
4 o'clock rock
Input contained 4 digits which summed to 10
$ ./sum_digits
12 twelve 24 twenty four
thirty six 36
Input contained 6 digits which summed to 18
```

Revision questions

The remaining tutorial questions are primarily intended for revision - either this week or later in session. Your tutor may still choose to cover some of the questions time permitting.

6. Write a program `input_statistics.c` that for the characters provided on standard input:
   - outputs the number of white-space characters (spaces, tabs and new lines)
   - outputs the numbers of words word (any contiguous sequence of non-white-space characters), and
   - outputs the length of the shortest word
   - outputs the length of the longest word

   For example:

```
$ ./input_statistics
   "Beauty is truth, truth beauty," -- that is all
  Ye know on earth, and all ye need to know.
Input contains 27 blanks, tabs and new lines
Number of words: 19
Length of shortest word: 2
Length of longest word: 8
$ ./input_statistics
And here is another example with only one line of input!!!!!!!!!
Input contains 11 blanks, tabs and new lines
Number of words: 11
Length of shortest word: 2
Length of longest word: 14
```

7. How many ints can the array `matrix` below hold?

```
#include <stdio.h>

#define N_ROWS 12
#define N_COLUMNS 15

int main(void) {
    int matrix[N_ROWS][N_COLUMNS];
```

   Write nested loops that set every element of `matrix`. Each element should be set to the product of its two indices.

   Write nested loops that print the elements of `matrix` plus sums of each row and sums of each column.

   The output of your code should look like this:

```
$ a.out
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0 |    0
    0    1    2    3    4    5    6    7    8    9   10   11   12   13
   14 |  105
    0    2    4    6    8   10   12   14   16   18   20   22   24   26
   28 |  210
    0    3    6    9   12   15   18   21   24   27   30   33   36   39
   42 |  315
    0    4    8   12   16   20   24   28   32   36   40   44   48   52
   56 |  420
    0    5   10   15   20   25   30   35   40   45   50   55   60   65
   70 |  525
    0    6   12   18   24   30   36   42   48   54   60   66   72   78
   84 |  630
    0    7   14   21   28   35   42   49   56   63   70   77   84   91
   98 |  735
    0    8   16   24   32   40   48   56   64   72   80   88   96  104
  112 |  840
    0    9   18   27   36   45   54   63   72   81   90   99  108  117
  126 |  945
    0   10   20   30   40   50   60   70   80   90  100  110  120  130
  140 | 1050
    0   11   22   33   44   55   66   77   88   99  110  121  132  143
  154 | 1155
----------------------------------------------------------------------
-----
    0   66  132  198  264  330  396  462  528  594  660  726  792  858
  924
```

8. A student has written this program to read ints until the end-of-input. It counts how many numbers it reads categorized by their last digit:

```c
#include <stdio.h>

#define N 10

int main(void) {
    int digit_count[N];
    int x, last_digit;

    while (scanf("%d", &x) == 1) {
        last_digit = x % N;
        digit_count[last_digit] = digit_count[last_digit] + 1;
    }
    last_digit = 0;
    while (last_digit < N) {
        printf("%d numbers with last digit %d read\n", digit_count[last_digit], last_digit);
        last_digit = last_digit + 1;
    }

    return 0;
}
```

It works on the students laptop:

```
$ gcc -Wall -O last_digit.c
$ a.out
42 121 100 11
<cntrl-d>
1 numbers with last digit 0 read
2 numbers with last digit 1 read
1 numbers with last digit 2 read
0 numbers with last digit 3 read
0 numbers with last digit 4 read
0 numbers with last digit 5 read
0 numbers with last digit 6 read
0 numbers with last digit 7 read
0 numbers with last digit 8 read
1 numbers with last digit 9 read
```

But when run at uni, it fails:

```
$ dcc last_digit.c
$ a.out
42 121 100 11
<cntrl-d>
778121076 numbers with last digit 0 read
7632239 numbers with last digit 1 read
-2032569224 numbers with last digit 2 read
32727 numbers with last digit 3 read
0 numbers with last digit 4 read
0 numbers with last digit 5 read
-2032409578 numbers with last digit 6 read
32727 numbers with last digit 7 read
-21600000 numbers with last digit 8 read
32767 numbers with last digit 9 read
```

Why doesn't the code work at uni?

Why doesn't `dcc` detect an error?

Fix the code (make sure you understand how it works - it's a common and useful programming pattern).

9. a. What is the effect of each of the following statements? What are the initial values in the arrays?

```
int nums1[10];
```

```
int nums2[] = {0,1,2,3,4,5,6,7,8,9};
```

```
int nums3[10] = {0,2,4,6,8,-2};
```

```
int nums4[10] = {0};
```

```
int nums5[2][10] = {{0,1,2,3,4,5,6,7,8,9},
                    {10,20,30,40,50,60,70,80,90,100}};
```

b. What would the output of the following fragment of code be - given the array definitions above?

```
printf("%d\n",nums2[3]);
printf("%d\n",nums3[5]);
printf("%d\n",nums5[0][1]);
printf("%d\n",nums5[1][0]);
nums1[0] = nums2[1] + 10 ;
printf("%d\n",nums1[0]);

int i = 0;
printf("%d\n",nums1[i]);
```

c. What is wrong with the following piece of code - given the above array definitions?

```
printf("%d\n",nums2[10]);

printf("%d\n",nums5[2][0]);

printf("%d\n",nums5[1][10]);
```

COMP1511 18s2: Programming Fundamentals is brought to you by
the School of Computer Science and Engineering at the University of New South Wales, Sydney.
For all enquiries, please email the class account at cs1511@cse.unsw.edu.au
CRICOS Provider 00098G