University of California San Diego

UCSD

ECE15: Lab #1

This lab relates to the material covered in Lecture Units 2, 3, and 4 in class, and in Chapters 1, 2, and 3 of the Kernighan & Ritchie textbook. The same material is also covered in Chapters 3, 4, 5, and 6 of the Kochan book. Here are several instructions specific to this lab:

- Throughout this lab, you *can assume* that the input provided by the user has the expected ***type*** and is in the appropriate range for this type. For example, if your program prompts the user to enter an integer, you can assume that the user will, indeed, enter an integer (rather than, say, a character or a real number) in the range that fits in a memory cell of type **`int`**.
- However, you *should not assume* that the user enters a valid ***value***. For example, if your program prompts the user to enter an integer between 0 and 10, you can only assume that the input is an integer, but you have to ***verify*** that this integer is in the correct range. If it is not, your program should terminate with an appropriate error message.
- For each of the four programs below, you should use the file `template.c` provided on the course website at `ece15.ucsd.edu/Main/Homeworks.html`. Try to make your programs easy to follow and understand: choose meaningful names for identifiers, use proper indentation, and provide comments wherever they might be helpful.

This lab will be graded out of 100 points, with the first problem worth 16 points (not counting the extra-credit part) and each of the three remaining problems worth 28 points. You should submit the lab by following the instructions posted at `ece15.ucsd.edu/Labs/TurnIn1.html`.

Problem 1.

Write a C program, called `logic_abc.c`, that prompts the user to enter three integers *a*, *b*, *c*, and then computes the results of the following *logical operations*, in sequence:

```
!a || !b++ && c
(a-1 || b/2) && (c*=2)
(a-- || --b) && (c+=2)
a || !(b && --c)
```

Here are three sample runs of this a program, assuming that the executable file is called `logic_abc`.

```
/home/userXYZ/ECE15/Lab1> logic_abc
Enter integers a,b,c: 1 1 1
      !a || !b++ && c: False
(a-1 || b/2) && (c*=2): True
(a-- || --b) && (c+=2): True
     a || !(b && --c): False
```

```
/home/userXYZ/ECE15/Lab1> logic_abc
Enter integers a,b,c: -7 42 0
       !a || !b++ && c: False
(a-1 || b/2) && (c*=2): False
(a-- || --b) && (c+=2): True
      a || !(b && --c): True
```

```
/home/userXYZ/ECE15/Lab1> logic_abc
Enter integers a,b,c: 10 0 1000
       !a || !b++ && c: True
(a-1 || b/2) && (c*=2): True
(a-- || --b) && (c+=2): True
      a || !(b && --c): True
```

Extra Credit (15 points)

Are there values of a, b, and c that would make all the results **`False`**? Submit your answer to this question as a comment in your program. Please explain clearly how you have arrived at this answer.

Hint: It may be useful to start with the last expression and work backwards.

Problem 2.

Write a C program, called `digits.c`, that prompts the user to enter an integer, and then prints out its decimal digits, each one on a separate line, starting with the least significant (rightmost) digit. Each printed line should indicate which of the digits is being printed. Note that the user *can* input either a positive or a negative integer: the inputs n and $-n$, where n is an integer, should produce exactly the same output. Here are three sample runs of this program, assuming that the executable file is called `digits`.

```
/home/userXYZ/ECE15/Lab1> digits
Enter an integer: 512
Digit (1): 2
Digit (2): 1
Digit (3): 5
```

```
/home/userXYZ/ECE15/Lab1> digits
Enter an integer: 0
Digit (1): 0
```

```
/home/userXYZ/ECE15/Lab1> digits
Enter an integer: -16
Digit (1): 6
Digit (2): 1
```

Hint: You can use the integer division and modulo operators to parse a positive integer into its digits. For example, **`n%10`** produces the least significant digit of **`n`**, while **`n/10`** deletes this digit.

Problem 3.

Write a C program, called `power_of_three.c`, that prompts the user to input a *positive integer*, and then checks whether this integer is a power of 3. If the input is, indeed, equal to 3^n for some non-negative integer n (for example $1 = 3^0$, $81 = 3^4$, or $129140163 = 3^{17}$), the program should tell the user that the input is a power of 3 and print the exponent n. If the input is a positive integer which is not a power of 3, the program should print a corresponding message. Finally, if the input is an integer less than 1, the program should terminate with an appropriate error message. Here are four sample runs of this program, assuming that the executable file is called `power_of_three`.

```
/home/userXYZ/ECE15/Lab1> power_of_three
Enter a positive integer: 81
81 is a power of 3, exponent = 4
```

```
/home/userXYZ/ECE15/Lab1> power_of_three
Enter a positive integer: 1
1 is a power of 3, exponent = 0
```

```
/home/userXYZ/ECE15/Lab1> power_of_three
Enter a positive integer: 12
12 is not a power of 3!
```

```
/home/userXYZ/ECE15/Lab1> power_of_three
Enter a positive integer: -1
Error: invalid integer entered!
```

Note:

- ▶ The number 3 should be introduced into the program as a symbolic constant **`BASE`**, using the **`#define`** compiler directive. It should not appear *anywhere else* in the program. Moreover, the program should continue working correctly when **`BASE`** is re-defined to stand for another integer greater than 1, including all the messages that the program prints out.

Hints:

- In order to check whether a given positive integer is a power of 3, you could use, for example, the **`while`** loop. Inside the loop, you could use the modulo (`%`) and integer division (`/`) operators.
- You can use the relational expression **`x<=y`** in order to test whether **`x`** is less than or equal to **`y`**. In order to test whether **`x`** is not equal to **`y`**, you can use the expression **`x!=y`**.
- If you'd like, you can use the following outline for your program:

```
#define BASE 3

int main()
{
   int num, ... ;

   printf("Enter a positive integer: ");
   scanf("%d", &num);

   if (num <= 0)
   {
      printf("Error: invalid integer entered!\n");
      return 1;
   }

   ⋮

   return 0;
}
```

Problem 4.

Write a C program, called `integer_sum.c`, that computes the sum of integers interactively provided by the user. At each iteration, the program should print the number of integers that the user has entered so far, along with a menu describing three possible actions from which the user could choose. The three possible actions are as follows:

1. Entering a new integer
2. Printing the sum of all the integers entered so far
3. Exiting the program

If the user enters a number other than 1, 2, or 3 when presented with this menu, the program should simply print the same menu again, prompting the user to enter a new choice of action. You are *not* required to introduce the constants 1, 2, and 3 into the program using **`#define`**, although you can do so. You *can* assume that when the user is prompted to enter an integer, the user indeed enters an integer.

Here is one sample run of such a program, assuming that the executable file is called `integer_sum`. Note that the input from the user is marked as underlined text below.

```
/home/userXYZ/ECE15/Lab1> integer_sum
Welcome to the integer sum program!
So far, you have entered 0 numbers. You may:
1. Enter a new integer
2. Display the current sum
3. Exit
Please enter your choice: 2
The current sum is: 0
So far, you have entered 0 numbers. You may:
1. Enter a new integer
2. Display the current sum
3. Exit
Please enter your choice: 1
Enter the new integer: 30
So far, you have entered 1 numbers. You may:
1. Enter a new integer
2. Display the current sum
3. Exit
Please enter your choice: 1
Enter the new integer: -17
So far, you have entered 2 numbers. You may:
1. Enter a new integer
2. Display the current sum
3. Exit
Please enter your choice: 2
The current sum is: 13
So far, you have entered 2 numbers. You may:
1. Enter a new integer
2. Display the current sum
3. Exit
Please enter your choice: 6
So far, you have entered 2 numbers. You may:
1. Enter a new integer
2. Display the current sum
3. Exit
Please enter your choice: 2
The current sum is: 13
So far, you have entered 2 numbers. You may:
1. Enter a new integer
2. Display the current sum
3. Exit
Please enter your choice: 1
Enter the new integer: 42
So far, you have entered 3 numbers. You may:
1. Enter a new integer
2. Display the current sum
3. Exit
Please enter your choice: 3
/home/userXYZ/ECE15/Lab1>
```

***Hint*:**

► If you'd like, you can use the following outline for your program:

```
int main()
{
   int number, sum, choice, total_numbers;

   printf("Welcome to the integer sum program!\n");

   choice = 0; total_numbers = 0; sum = 0;
   while (choice != 3)
   {
       ⋮
   }
   return 0;
}
```