Lab 6: Loops and Functions

1. **[While loop]** In last lab you wrote the program to check whether a number is a palindrome or not. You took the length of the number as input and wrote the program. Now write the program without taking length as an input. Just take the number as an input. Print the reverse and whether it is a palindrome.

   Sample Input :
   5432
   Sample Output :
   2345
   not a palindrome

   Sample Input :
   1223221
   Sample Output :
   1223221
   palindrome

   Use auto-evaluate on BodhiTree to check the correctness of your program.

2. **[Function with return value, pass by value]** Modify the above program so that you now use a function is_palindrome() to determine whether if the given number is a palindrome or not. The function should have an integer parameter and should return a boolean (true if palindrome, false if not). Note that the function should **NOT** print anything out. Thus, your main program should now be:

```
main_program{
  int x;
  cin>>x;
  if (is_palindrome(x))
      cout << "palindrome" << endl;
  else
      cout << "not a palindrome" << endl;
}
```

   Use auto-evaluate on BodhiTree to check the correctness of your program.

3. **[Function with return value, pass by reference]** Now write a program which implements a function reverse_int, which has an integer parameter and a boolean return

value. It returns true if the parameter is a palindrome and false if not. However, it also **changes** the parameter and sets it to the “reverse” of the original number. Your main program should be as follows:

```
main_program() {
        int n;
        cin >> n;
        if (reverse_int(n))
                cout <<  “ number is a palindrome” << endl;
        else
                cout <<  “reverse of the number is: “ << n << endl;
}
```

Sample input: 3242
Sample output:
reverse of the number is: 2423
Sample input: 87322378
number is a palindrome

Use auto-evaluate on BodhiTree to check the correctness of your program.

4. **[for loop, functions]** Write a program which uses a for loop and functions to list out all the prime factors of a number. You should write two functions in this program.

    1. is_prime(x) : to check whether a number is prime or not
    2. prime_factors(x) : to print all the prime factors using is_prime(x) function

    Your main program should just take input and simply call the second function. i.e., the main program should be the following:

```
main_program{
      int x;
      cin>>x;
      prime_factors(x);
}
```

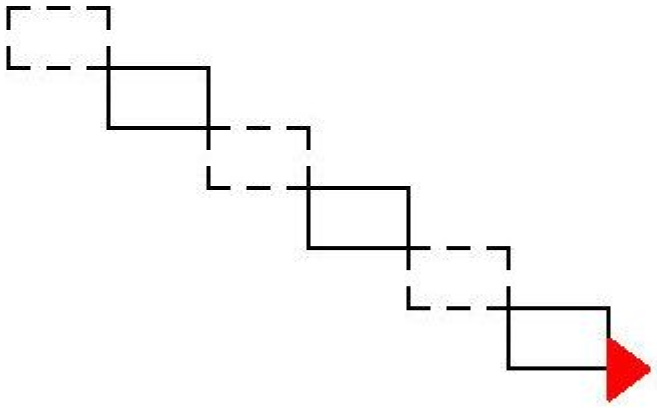
Do not worry about the time taken for large numbers. However, the for loop for checking factors should not go longer than necessary. Output should be all the prime factors in the increasing order.

Sample Input : 30
Sample Output : 2 3 5

Sample Input : 770
Sample Output : 2 5 7 11

Use auto-evaluate on BodhiTree to check the correctness of your program.

5. Write a turtle simulator program to draw the following step pattern, where the number of steps is taken as input from the user.



Do this by writing the following two functions:

- void dashed_line(int length, bool dashed, int dash_length)
    - Function pre-condition: Pen should be down. If "dashed" is true, length should be divisible by dash_length.
    - Function behavior: Line will start in the direction where the turtle is facing before function is called. If "dashed" is true, draw a dashed line, else draw solid line. If dashed, start with dash, then space, then dash...
    - Function post-condition: Turtle is left where the line finishes, in the same direction. Pen is down.
- void turtle_rectangle(int length, int breadth, bool dashed, int dash_length)
    - Function pre-condition: pen is down. If dashed is true, length and breadth should be divisible by dash_length
    - Function behaviour: Starts a rectangle whose top left corner is where the turtle is currently and whose top side is in the direction where the turtle is currently facing
    - Function post-condition: Leaves turtle in the same place and same direction as it was before function was called

Extra questions

1. Write a program to reverse a floating point number. Write a function reverse_float(x) with a single floating point argument, x, reverse this number and store it in x. If x is an

integer, it should be treated as ‘x.0’. Assume that the number has maximum 4 digits before and 4 digits after the decimal point . Your final main_program should be as below.

```
main_program{
      float x;
      cin>>x;
      reverse_float(x);
      cout<<x<<endl;
}
```

Hint :

You may write functions for
finding the floor (greatest integer function) of any number
finding the number of digits in an integer
reversing an integer
and make use of all these to write reverse_float

Input:
123.45
Output:
54.321
Input:
100
Output:
0.001

2. A stream of characters is input one by one. Check when a sequence of consecutive characters ending at the last input character spells the word “program”. Stop the input from the stream at this point and print “found” (Assume all characters are between ‘a’ - ‘z’, lower case). Add an extra condition that the input is stopped whenever the character ‘0’ is entered. If the program is stopped in such a case, print “not found”.

   Sample input
   sasdaprogramadasf
   Sample output
   found
   Sample input
   whatwillthisprint0
   Sample output
   not found

3. Extend the carrom simulation program as follows. Your animation should now simulate a friction-full board. Thus, the striker should look like it is slowing down and then stopping. It should slow down both while it is moving on the board, and when it hits a side. It should slow down a little more when it hits a side vs while it is just moving on the board.