

## CISS240: Introduction to Programming Assignment 5

Name: \_\_\_\_\_

### OBJECTIVES

1. Write functions
2. Make function calls
3. Break down a program into functions

The purpose of this assignment is to write functions. You will also see how problems are broken down into subproblems (or subgoals). You have seen this before in the context of breaking down a program into different goals within the code in `main()`. The difference now is that we break down the program into pieces and then create functions for the pieces instead of keeping these pieces inside `main()`.

The first problem (Q1 and Q2) involves writing a simple tool to analyze the representation of primes using polynomials. A basic function is developed in Q1 so that it can be used in Q2. The function checks if an integer is prime.

Questions Q3–Q5 involve the rewrite of our calendar month printing program. Q3 involves writing a function that tells us if a year is a leap year. The second function (in Q4) tells us how many days there are in a given month and year. Q4 obviously must use Q3 since the days in a month for the case of February depends on whether the year is a leap year or not. Finally, Q5 uses the functions in Q3–Q4 to print the calendar itself.

Q6 involves writing several functions to analyze the famous unsolved “ $3x+1$ ” problem.

There are several short functions to be implemented in Q7. The functions are then used to compute an approximation to the value of  $\pi$  (which we know is roughly 3.14159...) using a technique called Monte–Carlo simulation.

## Q1. [NOT GRADED – SEE NOTES]

First write a function `isprime()` such that `isprime(n)` returns `true` when `n` is prime and `false` when `n` is not a prime. Here's the skeleton code which you **MUST** use. Several test cases are included. Note that the integers 0 and 1 are NOT primes. Negative integers are also not primes.

```
#include <iostream>

bool isprime(int n)
{
    ... your code here ...
}

int main()
{
    int i;
    std::cin >> i;

    std::cout << i << (isprime(i) ? " is prime" : " is not prime")
              << std::endl;

    return 0;
}
```

## TEST 1.

```
-3
-3 is not prime
```

## TEST 2.

```
0
0 is not prime
```

## TEST 3.

```
1
1 is not prime
```

## TEST 4.

```
2
2 is prime
```

TEST 5.

3  
3 is prime

TEST 6.

12  
12 is not prime

TEST 7.

23  
23 is prime

Q2. For this question, you should use the `isprime()` function from Q1.

This question involves polynomials that generate primes. Consider the following polynomial:

$$p(x) = x^2 + 2x + 3$$

Note that

$$p(0) = 0 * 0 + 2 * 0 + 3 = 3, \text{ which is a prime}$$

$$p(1) = 1 * 1 + 2 * 1 + 3 = 6, \text{ which is not a prime}$$

$$p(2) = 2 * 2 + 2 * 2 + 3 = 11, \text{ which is a prime}$$

Therefore this  $p(x)$ , when  $x$  ranges from 0 to 2 (inclusive), generates 2 primes, i.e., 3 and 11.

Write a program that accepts  $A, B, C, N$  and then analyzes all the polynomials

$$p(x) = ax^2 + bx + c$$

for  $a$  ranging from 1 to  $A$  (inclusive),  $b$  ranging from 1 to  $B$  (inclusive),  $c$  ranging from 1 to  $C$  (inclusive) and reporting the number of  $x$  ranging from 0 to  $N$  (inclusive) such that  $p(x)$  is prime.

Finally, print the polynomial that represents the most primes. If there's a tie, report the one that is smallest according to "dictionary order", i.e., the one that appears first.

Note that all integers are printed with `std::setw(5)`.

[An experiment for fun: Is there a polynomial  $p(x)$  that represents primes for all  $x = 0, 1, 2, 3, \dots$ ? In general, suppose that the prime length of a polynomial  $p(x)$  is the number  $n$  such that  $p(0), p(1), p(2), \dots, p(n-1)$  are all primes. What is the largest prime length you can find and what is the polynomial?]

TEST 1

1	1	1	5
1x^2 +	1x +	1:	4
largest			
1x^2 +	1x +	1:	4

In this case  $A = 1, B = 1, C = 1, N = 5$ .

Therefore, we're only analyzing a single polynomial, i.e.,  $x^2+x+1$  for  $x = 0, 1, 2, 3, 4, 5$ .

Here is the analysis by hand:

$x$	$x^2 + x + 1$	
0	$0^2 + 0 + 1 = 1$	not prime
1	$1^2 + 1 + 1 = 3$	prime
2	$2^2 + 2 + 1 = 7$	prime
3	$3^2 + 3 + 1 = 13$	prime
4	$4^2 + 4 + 1 = 21$	not prime
5	$5^2 + 5 + 1 = 31$	prime

Hence, there are 4 primes represented by this polynomial for  $x$  running from 0 to 5.

TEST 1.

$2 \ 2 \ 2 \ 10$				
$1x^2 +$	$1x +$	$1:$	$6$	
$1x^2 +$	$1x +$	$2:$	$1$	
$1x^2 +$	$2x +$	$1:$	$0$	
$1x^2 +$	$2x +$	$2:$	$5$	
$2x^2 +$	$1x +$	$1:$	$5$	
$2x^2 +$	$1x +$	$2:$	$5$	
$2x^2 +$	$2x +$	$1:$	$6$	
$2x^2 +$	$2x +$	$2:$	$1$	
largest				
$1x^2 +$	$1x +$	$1:$	$6$	

In this case, note that there is a tie: there are two polynomials that achieve 6 primes for  $x$  running from 0 to 10. The earlier one is reported as the “winner”.

TEST 2.

```

10 10 10 100
  1x^2 +    1x +    1:    32
  1x^2 +    1x +    2:     1
  1x^2 +    1x +    3:    15
  1x^2 +    1x +    4:     0
  1x^2 +    1x +    5:    30
  1x^2 +    1x +    6:     0
  1x^2 +    1x +    7:    32
  1x^2 +    1x +    8:     0
  1x^2 +    1x +    9:    13
  1x^2 +    1x +   10:     0
  1x^2 +    2x +    1:     0
  1x^2 +    2x +    2:    19
  1x^2 +    2x +    3:    11
  1x^2 +    2x +    4:    17
  1x^2 +    2x +    5:    21
  1x^2 +    2x +    6:     7
... output not shown ...
 10x^2 +   10x +    5:     1
 10x^2 +   10x +    6:     0
 10x^2 +   10x +    7:    27
 10x^2 +   10x +    8:     0
 10x^2 +   10x +    9:    11
 10x^2 +   10x +   10:     0
largest
  8x^2 +   10x +    1:    53

```

The output of this test case is too long to be shown in its entirety. Part of it is snipped. Note that, in this case, the maximum density of primes represented is  $53/101$ , about 52.4%. Can we find a better polynomial to represent primes for  $x = 0, \dots, 100$  if we expand our search?

TEST 3.

```

20 20 20 100
... output not shown ...
largest
  7x^2 +    7x +   17:    70

```

Now we have achieved a density of  $70/101$ , almost 70%.

TEST 4.

```

30 30 30 100
... output not shown ...
largest
  1x^2 +   23x +   23:    75

```

The maximum density is about 75%.

TEST 5.

```
50 50 50 100
... output not shown ...
largest
1x^2 + 1x + 41: 87
```

Still improving!

TEST 6.

```
60 60 60 100
... output not shown ...
largest
1x^2 + 1x + 41: 87
```

Oops ... we still have the same polynomial.

TEST 7.

```
70 70 70 100
... output not shown ...
largest
1x^2 + 1x + 41: 87
```

Wait ... what's happening??? Expanding our brute force search doesn't come up with a better polynomial ... hmmm ...

TEST 8.

```
80 80 80 100
... output not shown ...
largest
1x^2 + 1x + 41: 87
```

Is this a coincidence???

TEST 9.

```
100 100 100 100
... output not shown ...
largest
1x^2 + 1x + 41: 87
```

What!!! Is there something special about  $x^2 + x + 41$ !?!?

Q3. Write a function `is_leap_year()` such that `is_leap_year(year)` returns `true` exactly when the `year` (an `int`) is a leap year. You must use the skeleton code given below.

```
#include <iostream>

//-----
// This function returns true if year is a leap year. Otherwise false is
// returned.
// (This function has no output.)
//-----
bool is_leap_year(int year)
{
    ... your code here ...
}

int main()
{
    int y;
    std::cin >> y;
    std::cout << is_leap_year(y) << std::endl;

    return 0;
}
```

For your reference, here's a relevant program from a previous assignment:

```
#include <iostream>

int main()
{
    int year = 0;
    std::cin >> year;

    bool is_leap_year = ((year % 4 == 0)
                        && (year % 100 != 0
                          || (year % 100 == 0 && year % 400 == 0))
                        );

    if (is_leap_year)
    {
        std::cout << "leap year" << std::endl;
    }
}
```



```
    else
    {
        std::cout << "not leap year" << std::endl;
    }

    return 0;
}
```

Make sure you test your program thoroughly.

TEST 1.

```
1991
0
```

TEST 2.

```
2004
1
```

TEST 3.

```
1200
1
```

TEST 4.

```
1900
0
```

Q4. Write a function `days_in_month()` such that `days_in_month(month, year)` returns the number of days in the given month (an `int`) of the given year (an `int`). (See *a06*.)

You must use the skeleton code given below. Note that the test cases are included in the code.

You will need the function from Q3. Therefore you should copy-and-paste the function from Q3 to the code for this question.

```
#include <iostream>

//-----
// This function returns true if year is a leap year. Otherwise false is
// returned.
// (This function has no output.)
//-----
bool is_leap_year(int year)
{
    ... your code here ...
}

//-----
// This function returns the number of days in given month and year.
// (This function has no output.)
//-----
int days_in_month(int month, int year)
{
    ... your code here ...
}

int main()
{
    int m, y;
    std::cin >> m >> y;
    std::cout << days_in_month(m, y) << std::endl;

    return 0;
}
```

For your reference, here's a relevant program from a previous assignment:

```
#include <iostream>

int main()
```

```
{
    int yyyymmdd = 0;
    std::cin >> yyyymmdd;

    int yyyy = yyyymmdd / 10000;    // year
    int mm = yyyymmdd / 100 % 100;  // month
    int dd = yyyymmdd % 100;        // day-of-month

    bool valid = false; // true exactly when yyyymmdd is a valid date

    if (1 <= mm && mm <= 12)
    {
        if (mm == 2)
        {
            //-----
            // CASE: February.
            // There are 29 or 28 days depending on whether the year
            // is a leap year or not.
            //-----
            if (yyyy % 4 == 0
                && ((yyyy % 100 != 0) || (yyyy % 100 == 0)
                    && (yyyy % 400 == 0)))
            {
                valid = (1 <= dd && dd <= 29);
            }
            else
            {
                valid = (1 <= dd && dd <= 28);
            }
        }
        else if ((mm % 2 == 1 && mm <= 7) || (mm % 2 == 0 && mm >= 8))
        {
            //-----
            // CASE: 31 days in months 1,3,5,7,8,10,12
            //-----
            valid = (1 <= dd && dd <= 31);
        }
        else
        {
            //-----
            // CASE: 30 days in months 4,6,9,11
            //-----

```

```
        valid = (1 <= dd && dd <= 30);
    }
}

if (valid)
{
    std::cout << "correct" << std::endl;
}
else
{
    std::cout << "incorrect" << std::endl;
}

return 0;
}
```

Make sure you test your program thoroughly.

TEST 1.

2 2004  
29

TEST 2.

2 2003  
28

TEST 3.

5 2003  
31

TEST 4.

2 1900  
28

TEST 5.

9 2008  
30

TEST 6.

$\frac{2}{29} \cdot 1200$
---------------------------

Q5. Recall that we had a previous problem that involved printing a calendar month. Here's the problem:

Write the following program that prints a calendar month. The program prompts the user for three integers: the month, the year, and the day-of-week for the first day of the month (with 0 representing Sunday, 1 representing Monday, etc.) For instance, if the user entered

```
3 2008 6
```

It means that he/she wants the calendar for March 2008 and the first day of the month is a Saturday. The output is

```
March 2008
-----
Su Mo Tu We Th Fr Sa
      1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

Rewrite the calendar month printing program so that it uses the two functions `is_leap_year()` and `days_in_month()` from the previous questions. You must use the skeleton code given below.

Note that we are not changing how the original program works. The point is to “factor” out useful code from our original program. This makes the program more readable – it will be very obvious once you’re done with this question. Also, the functions factored out can be used for other programs.

```
#include <iostream>

//-----
// This function returns true if year is a leap year. Otherwise false is
// returned.
// (This function has no output.)
//-----
bool is_leap_year(int year)
{
    ... your code here ...
}
```

```
//-----  
// This function returns the number of days in given month and year.  
// (This function has no output.)  
//-----  
int days_in_month(int month, int year)  
{  
    ... your code here ...  
}  
  
//-----  
// This function prints the calendar of the given month and year.  
// The parameter day_of_first is the day of the 1st of the given month  
// and year.  
//-----  
void print_calendar_month(int month, int year, int day_of_first)  
{  
    ... your code here ...  
}  
  
int main()  
{  
    int month, year, day_of_first;  
    std::cin >> month >> year >> day_of_first;  
  
    print_calendar_month(month, year, day_of_first);  
  
    return 0;  
}
```

For your reference, here's the relevant program from a previous assignment:

```
#include <iostream>  
#include <iomanip>  
#include <cmath>  
  
int main()  
{  
    int month, year, day_of_first;  
    std::cin >> month >> year >> day_of_first;
```

```
switch (month)
{
    case 1: std::cout << "January "; break;
    case 2: std::cout << "February "; break;
    case 3: std::cout << "March "; break;
    case 4: std::cout << "April "; break;
    case 5: std::cout << "May "; break;
    case 6: std::cout << "June "; break;
    case 7: std::cout << "July "; break;
    case 8: std::cout << "August "; break;
    case 9: std::cout << "September "; break;
    case 10: std::cout << "October "; break;
    case 11: std::cout << "November "; break;
    case 12: std::cout << "December "; break;
}

std::cout << year << '\n'
          << "-----\n"
          << "Su Mo Tu We Th Fr Sa\n";

// Compute the number of days in the month and store in days_in_month
int days_in_month = 0;
if (1 <= month and month <= 12)
{
    if (month == 2) // 28 or 29 days
    {
        if (year % 4 == 0
            && ((year % 100 != 0) || (year % 100 == 0)
                && (year % 400 == 0)
            )
        {
            days_in_month = 29;
        }
        else
        {
            days_in_month = 28;
        }
    }
    else if ((month % 2 == 1 && month <= 7) ||
             (month % 2 == 0 && month >= 8)) // 31 days
    {
        days_in_month = 31;
    }
}
```



```

    }
    else // 30 days
    {
        days_in_month = 30;
    }
}

// Print spaces before the first day of the month
for (int i = 0; i < day_of_first; i++)
{
    std::cout << "    ";
}

int day_of_week = day_of_first;
for (int day_in_month = 1; day_in_month <= days_in_month; day_in_month++)
{
    std::cout << std::setw(2) << day_in_month << ' ';

    day_of_week++;
    if (day_of_week == 7)
    {
        std::cout << std::endl;
        day_of_week = 0;
    }
}

std::cout << std::endl;

return 0;
}

```

Make sure you test your program thoroughly.

TEST 1.

```

3 2008 6
March 2008
-----
Su Mo Tu We Th Fr Sa
                1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31

```

TEST 2.

```

3 2008 4
March 2008
-----
Su Mo Tu We Th Fr Sa
          1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
    
```

TEST 3.

```

2 2008 5
February 2008
-----
Su Mo Tu We Th Fr Sa
          1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29
    
```

TEST 4.

```

2 2009 0
February 2009
-----
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
    
```

Q6. The following problem is called the “ $3x+1$ ” problem. Consider the following function  $T$ :

$$\begin{aligned} &\text{if } n \text{ is odd, } T(n) = 3n + 1 \\ &\text{if } n \text{ is even, } T(n) = n/2 \end{aligned}$$

Another way to write this (mathematically) is:

$$T(n) = \begin{cases} 3n + 1, & \text{if } n \text{ is odd} \\ n/2, & \text{if } n \text{ is even} \end{cases}$$

Note that there  $T(n)$  is made up of two formulas; you need to choose the right one. Here are some examples:

$$\begin{aligned} T(42) &= 42/2 = 21 && \text{(you choose the } n/2 \text{ formula since } n = 42 \text{ is even)} \\ T(5) &= 3(5) + 1 = 16 && \text{(you choose the } 3n + 1 \text{ formula since } n = 5 \text{ is odd)} \end{aligned}$$

Note that you can use the “output” of  $T(42)$  which is 21 and apply  $T$  to it again:  $T(21)$ .

$$\begin{aligned} T(42) &= 21 \\ T(21) &= 3(21) + 1 = 64 \end{aligned}$$

and yet again:

$$T(64) = 64/2 = 32$$

Of course you can do this as many times as you like, using outputs as inputs:

$T(42) = 42/2 = 21$	because 42 is even
$T(21) = 3(21) + 1 = 64$	because 21 is odd
$T(64) = 64/2 = 32$	because 64 is even
$T(32) = 32/2 = 16$	because 32 is even
$T(16) = 8$	because 16 is even
$T(8) = 4$	because 8 is even
$T(4) = 2$	because 4 is even
$T(2) = 1$	because 2 is even

OK ... that's enough ... let's stop at 1. Altogether we applied  $T$  eight times and in

doing that we got a sequence of numbers:

42, 21, 64, 32, 16, 8, 4, 2, 1

The “ $3x+1$ ” problem/conjecture states that if you apply enough  $T$  to a given strictly positive integer  $n$ , you will always get 1.

Well, this is definitely the case for 42. But is this true in general for all strictly positive  $n$ ?

The “ $3x+1$ ” problem is still an open problem: No one has proven it yet. (By the way, there is a £1000 reward for proving it ... in case you need some money.) This problem is an example of a dynamical system.

Write a program that prompts the user for two integers and verifies the conjecture for all integers in the range described by the two integers by printing the sequences for each integer until it reaches 1. (See explanation in TEST 1.)

The following skeleton must be used.

```
#include <iostream>

//-----
// The function T returns T(n) where:
//   if n is odd, T(n) = 3 * n + 1
//   if n is even, T(n) = n / 2 (integer division)
// (This function has no output.)
//-----
int T(int n)
{
    ... your code here ...
}

//-----
// The function print_3x_plus_1(n) prints the sequence of integers on
// applying T until 1 is reached.
//
// Example: On calling print_3x_plus_1(3),
// 3 10 5 16 8 4 2 1
// is printed.
//-----
```

```
void print_3x_plus_1(int n)
{
    ... your code here ...
}

int main()
{
    int a, b;
    std::cin >> a >> b;

    for (int n = a; n <= b; ++n)
    {
        print_3x_plus_1(n);
    }

    return 0;
}
```

TEST 1.

```
1 10
1
2 1
3 10 5 16 8 4 2 1
4 2 1
5 16 8 4 2 1
6 3 10 5 16 8 4 2 1
7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
8 4 2 1
9 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
10 5 16 8 4 2 1
```

As you can see, the user enters 1 and 10 and the program runs  $n$  from 1 to 10 and for each  $n$ , continually applies  $T$  until 1 is reached. For instance, when  $n$  is 6, continually applying  $T$  gives the sequence 6, 3, 10, 5, 16, 8, 4, 2, 1. In every case, the sequence does ultimately reach 1.

TEST 2.

```
11 20
11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
12 6 3 10 5 16 8 4 2 1
13 40 20 10 5 16 8 4 2 1
14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
15 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
16 8 4 2 1
17 52 26 13 40 20 10 5 16 8 4 2 1
18 9 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
20 10 5 16 8 4 2 1
```