```
Name: Muhammad Asim Ali
Student ID: V00854120
Email: maali@uvic.ca
Course: SENG475
Section: T01

Assignment ID: cpp_basics
Assignment Title: C++ Basics

Submission Source: cpp_basics-MAsimSENG/
Commit ID: ?

Submitted Files
===============

-rw-rw-r--     367 2020-05-21 22:24 ./CMakeLists.txt
-rw-rw-r--     140 2020-05-21 22:24 ./IDENTIFICATION.txt
drwxrwxr-x    4096 2020-05-21 22:24 ./include
drwxrwxr-x    4096 2020-05-21 22:24 ./include/ra
-rw-rw-r--    1253 2020-05-21 22:24 ./include/ra/random.hpp
drwxrwxr-x    4096 2020-05-21 22:24 ./lib
-rw-rw-r--    2079 2020-05-21 22:24 ./lib/random.cpp

Results
=======

Package          Operation Target           Status
nonprog          generate  ---              FAIL (1 0.0s 1L)
random_orig      generate  ---              FAIL (1 0.1s 1L)
random_sane      generate  ---              FAIL (1 0.1s 1L)
rational_orig    generate  ---              FAIL (1 0.1s 1L)
rational_sane    generate  ---              FAIL (1 0.1s 1L)

Normally, an operation is indicated as having a status of either "OK" or
"FAIL".  A status of "?" indicates that the operation could not be performed
for some reason (e.g., due to an earlier error or being a manual step).
The time (in seconds) required for an operation is denoted by an expression
consisting of a number followed by the letter "s" (e.g., "5.0s").
In the case of a test that consists of multiple test cases, the number of
failed test cases and total number of test cases is expressed as a fraction
(e.g., "10/50" means 10 test cases failed out of 50 test cases in total).
The length (in lines) of the log file generated by an operation is denoted by
an expression consisting of a number followed by the letter "L" (e.g., "10L").
To ascertain the reason for the failure of an operation, check the contents
of the log file provided.

Legend
======

Package: nonprog
    Nonprogramming exercises

Package: random_orig
    The code as originally submitted by the student.

Package: random_sane
    Code with modifications to perform API sanity checking.

Package: rational_orig
```

```
    The code as originally submitted by the student.

Package: rational_sane
    Code with modifications to perform API sanity checking.
```

```
1  ERROR: missing file/directory README.pdf
```

```
1  ERROR: missing file/directory app/test_random.cpp
```

```
1  ERROR: missing file/directory app/test_random.cpp
```

```
1  ERROR: missing file/directory include/ra/rational.hpp
```

```
1  ERROR: missing file/directory include/ra/rational.hpp
```

```
1
2
3  cmake_minimum_required(VERSION 3.1 FATAL_ERROR )
4
5  project(LCG LANGUAGES CXX)
6
7  set(random_headers include/ra/random.hpp)
8  set(random_sources lib/random.cpp)
9
10 add_library(random ${random_sources} ${random_headers})
11
12 target_include_directories(random PUBLIC include/ra/)
13
14 install(TARGETS random DESTINATION lib)
15
16 install(FILES ${random_headers} DESTINATION include/ra)
```

```
1  class linear_congruential_generator {
2
3      // part a : unsigned integer type atleast 64bits
4
5  public:
6      typedef unsigned long long int int_type;
7
8
9      linear_congruential_generator(int_type a, int_type c, int_type m, int_type s
   );
10     // no need to provide destructor, it will be provided by default by the comp
   iler.
11
12     linear_congruential_generator(linear_congruential_generator && L); // move c
   onstructor T(T&&t)
13     linear_congruential_generator(const linear_congruential_generator &L); // co
   py constructor : T(const T &t)
14
15     linear_congruential_generator& operator=(const linear_congruential_generator
    & t); // copy assignment operator definition
16
17     linear_congruential_generator& operator=(linear_congruential_generator&& t);
   // move assignment operator definiton
18
19     bool operator==(const linear_congruential_generator& L1);
20
21     bool operator!=(const linear_congruential_generator& L1);
22
23     int_type operator()(); // overloading function call
24
25
26     // member function declarations
27
28     int_type multiplier();
29
30     int_type increment();
31
32     int_type modulus();
33
34     static int_type default_seed();
35
36     void seed(int_type s);
37
38     void discard(int_type n);
39
40     int_type min();
41     int_type max();
42
43
44     private:
45         int_type a;
46         int_type c;
47         int_type m;
48         int_type xo;
49         int_type current_state;
50
51
52
53 };
```

```
1
2   #include "random.hpp"
3   #include <iostream>
4
5   typedef unsigned long long int int_type;
6
7   linear_congruential_generator:: linear_congruential_generator(int_type a, int_ty
    pe c, int_type m, int_type s = default_seed()){
8       s=default_seed();
9       this->a =a;
10      this->c = c;
11      this-> m = m;
12      if(c % m ==0 && s % m ==0) {
13
14          xo= 1;
15      }
16      else {
17          xo=s;
18
19      }
20
21  this->current_state = xo;
22
23  }
24
25
26  int_type linear_congruential_generator::multiplier() {
27      return a;
28  }
29
30
31
32  int_type linear_congruential_generator::increment() {
33      return c;
34  }
35
36
37  int_type linear_congruential_generator::modulus() {
38      return m;
39  }
40
41   int_type linear_congruential_generator::default_seed(){
42      return 1;
43  }
44
45  void linear_congruential_generator::seed(int_type s) {
46
47      if(c%m ==0 and s%m ==0) {
48
49          xo=1;
50
51      }
52      else {
53          xo = s;
54      }
55
56  }
57
58  bool linear_congruential_generator:: operator==(const linear_congruential_genera
    tor& L1) {
59      return (L1.a == this->a && L1.c == this->c && L1.m == this->m && L1.current_
    state == this->current_state);
```

```
60
61
62
63  }
64
65  bool linear_congruential_generator:: operator!=(const linear_congruential_genera
    tor& L1) {
66
67    return !(L1.a == this->a && L1.c == this->c && L1.m == this->m && L1.current_s
    tate == this->current_state);
68
69
70
71  }
72
73  int_type linear_congruential_generator::operator()(){
74
75      // return the value corresponding to the next position of generator
76
77      current_state = (a*xo +c) % m;
78
79      return current_state;
80
81
82  }
83
84
85  void linear_congruential_generator::discard(int_type n ) {
86
87      // discards the next n numbers in the generated seq
88
89      // logic just compute the seed over the next n times
90      // those n seeds will not be used and we will advance
91
92      for(int i=0; i<n; i++){
93
94          xo = (a*xo +c) % m;
95
96      }
97
98
99  }
100
101 int_type linear_congruential_generator:: min() {
102     // min is 1 if c is 0 otherwise it is 0.
103     return c == 0  ? 1 :0;
104 }
105
106
107 int_type linear_congruential_generator:: max() {
108   return m-1;
109 }
110
111 linear_congruential_generator&linear_congruential_generator::operator=(linear_co
    ngruential_generator&& L )=default;
112
113 linear_congruential_generator&linear_congruential_generator::operator=(const lin
    ear_congruential_generator& L )=default;
```