Exercise 4 in OS - Valgrind , Graph data structure and Euler

C/C++ part

1. Create a Graph Data structure (you may use same data structure as ex.3) (10 pts)
2. Implement an Algorithm to find Euler circle (Euler circuit) on the graph or prove one does not exist. (10 pts)
3. Generate Random graph and run the algorithm on it. Receive parameters (number of edges, number of vertices, random seed) using argc,argv with getopt(3) - same as ex.1 (10 pts)
4. Provide code coverage reports, gprof, Valgrind/memcheck report and Valgrind callGraph for your code, (20 pts)
5. The following code has some problems. Run Valgrind/memcheck on it. And report the errors (10 pts)

Origin : https://www.parasoft.com/blog/finding-memory-leaks-in-c-or-c/

```
/*
 * File: hello.c
 */
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[]) {
   char *string, *string_so_far;
   int i, length;     length = 0;
   for(i=0; i<argc; i++) {
      length += strlen(argv[i])+1;
      string = malloc(length+1);

      /*  * Copy the string built so far. */
      if(string_so_far != (char *)0)
         strcpy(string, string_so_far);
      else *string = '\0';
      strcat(string, argv[i]);
      if(i < argc-1) strcat(string, " ");
      string_so_far = string;
   }
   printf("You entered: %s\n", string_so_far);
   return (0);
}
```

6. Demonstrate that you can run Valgrind attached to debugger. (10 pts)
   you may use IDE, graphical debugger or gdb.
7. The following code has a race condition. Detect it using Valgrind/Helgrind (10 pts)

Origin : https://www.classes.cs.uchicago.edu/archive/2014/spring/12300-1/labs/lab4/

```
// File: lab4/race.c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

#define NUM_THREADS 20

long accum = 0;

void *square(void *param) {
   int x = *(int *)param;
   accum += x * x;
   //sleep(1);
   pthread_exit(NULL);
}

int main() {
   pthread_t threads[NUM_THREADS];
   int *params[NUM_THREADS];

   for (long t = 0; t < NUM_THREADS; t++) {
      params[t] = malloc(sizeof(int));
      *params[t] = t + 1;
      pthread_create(&threads[t], NULL, square, (void *)params[t]);
   }

   for (long t = 0; t < NUM_THREADS; t++) {
      pthread_join(threads[t], NULL);
      free(params[t]);
   }

   printf("%ld\n", accum);
   pthread_exit(NULL);
}
```

Using C++ (Preferred) or Java or Python

1. Create a Singleton abstract base class that uses POSIX mutex for locking or corresponding lock from the language you chose
2. Create a Guard (Scope Mutex) class