REPORT

```
void take_forks(int i)                  /* i: philosopher number, from 0 to N–1 */
{
     down(&mutex);                      /* enter critical region */
     state[i] = HUNGRY;                 /* record fact that philosopher i is hungry */
     test(i);                           /* try to acquire 2 forks */
     up(&mutex);                        /* exit critical region */
     down(&s[i]);                       /* block if forks were not acquired */
}
void put_forks(i)                       /* i: philosopher number, from 0 to N–1 */
{
     down(&mutex);                      /* enter critical region */
     state[i] = THINKING;               /* philosopher has finished eating */
     test(LEFT);                        /* see if left neighbor can now eat */
     test(RIGHT);                       /* see if right neighbor can now eat */
     up(&mutex);                        /* exit critical region */
}

void test(i)                            /* i: philosopher number, from 0 to N–1 */
{
     if (state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING) {
          state[i] = EATING;
          up(&s[i]);
     }
}
```

Functions before the run():

take_forks(), put_forks(), test() are all from recitation 5. All functions are as explained in recitations. Down -> acquire
Up -> release
What I added in the functions are GUI methods to visualize.
*** Basically take_forks() goes to a hungry state then tests if it can eat.
put_forks() goes thinking state and checks if any neighbor can eat that if so gives the fork.
test() is for checking if no neighbor is eating and if it is hungry then it is okay to eat.

Inside the run():
Firstly we created a random number between 1-10 seconds.
Then we put thread down to sleep with respect to its walking time.
Then we release (up) their barrier with respect to the “ID” except the current thread and tell that the thread is ready to eat.
Visualize the table with PutPlate_GUI.
After that we make sure no one has started eating without any absence at the table. In order to do that we acquire(down) all barriers of the threads that are currently in the table.
We started dining at the table.
For better visualization I put them down to sleep before eating for 2 sec.
Each sleeps for a random time between 0-10 for the thinking stage.
At last every philosopher takes a fork, eats (with GUI method Eating_GUI), then puts down the fork.
That is basically how my program works.