מעבדה 10 - קיפאון (Deadlock)

תרגיל 1:

4 פקידים בכירים עובדים במשרד ממשלתי ב-4 ערים: חיפה, צפת, טבריה וכרמיאל. המזכירות שלהן כל הזמן מתקשרות אחת לשנייה כדי לקבוע פגישות בין הפקידים.
לא נקבעות פגישות בין יותר מ-2 פקידים ואף מזכירה לא יכולה להתקשר בו זמנית ל-2 מזכירות אחרות – רק לאחת.
ז"א, כאשר חיפה מתקשרת לטבריה, רק כרמיאל יכולה להתקשר לצפת או להפך, צפת לכרמיאל.
וכאשר חיפה מתקשרת לכרמיאל, רק צפת יכולה להתקשר לטבריה או להפך, טבריה לצפת.

ויש בעיה נוספת - בצפת ובחיפה הפקידים בכירים יותר מאשר בכרמיאל ובטבריה, לכן, כאשר צפת וחיפה מתקשרות אחת לשנייה, טבריה וכרמיאל מחכות להן ולא קובעות פגישות עד שצפת וחיפה יסיימו.
לעומת זאת, אם טבריה מתקשרת לכרמיאל או להפך, צפת וחיפה לא צריכות לחכות לסיום שיחתן.

נמספר את הערים:
כרמיאל -0 , טבריה -1, חיפה - 2, צפת – 3

ממשו שגרה Calls כך ש:
אף מזכירה לא תצליח להתקשר בו זמנית ליותר ממזכירה אחת.
לא יהיו מזכירות שבכלל לא מצליחות להתקשר.
שיחות המזכירות יהיו מקביליות ככל שניתן.

היעזרו בסמפורים. חשבו: האם יכול להיות קיפאון? אם כן, כיצד? כיצד תפתרו זאת בקוד?

ה- prototype של השגרה:

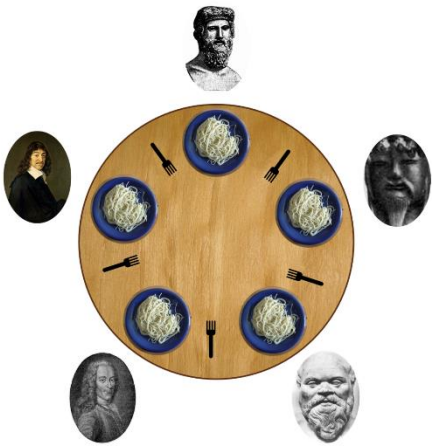
```
void Calls(int City1, int City2)
```

כאשר City1 ו City2 הם מספרים שלמים המייצגים את המזכירות/ערים

כתבו תכנית ראשית המפעילה 4 חוטים (לכל מזכירה חוט אחד) ובמשך 20 שניות נותנת לחוטים לעבוד.
כל חוט מפעיל את פונקציית Func אשר אחרי הגרלת מספר עיר להתקשר אליה, מפעילה את פונקציית Calls.

בפונקציה Calls נדרש שיופיע גם הקוד הבא:

```
printf("%d started to talk with %d\n", City1, City2);
sleep(1);
printf("%d finished to talk with %d\n", City1, City2);
```



תרגיל 2 – פילוסופים סועדים:

פילוסופים יושבים סביב שולחן עגול ומקדישים את חייהם לחשיבה.

במרכז השולחן יש קערה של אורז .

בין כל שני פילוסופים מונח מקל אכילה .

כאשר פילוסוף רעב, הוא נוטל את שני מקלות שמשני צדדיו ואוכל.

לאחר שסיים את ארוחתו הוא מחזיר את המקלות למקומם וממשיך לחשוב,

וכך עד אין סוף.

המשאב המשותף עליו מתחרים הפילוסופים הוא מקל האכילה. לא היינו רוצים לאפשר גישה של שני פילוסופים בו זמנית למקל אכילה כלשהו .

כיצד נסנכרן את הגישה למשאב זה?

נתונה תכנית ב-C תחת UNIX המממשת את פתרון הבעיה. עליכם להבין את הבעיה, לחשוב על פתרון (ניתן להיעזר במצגת התרגול) , להשלים את קוד התכנית, לקמפל ולהריץ את התכנית.

```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <assert.h>
#include <time.h>
#include <stdlib.h>
#include <semaphore.h>
#include <string.h>
#define LEFT (i,N)  (i + N - 1) % N    //מקרו המגדיר את מספר השכן משמאל
#define RIGHT (i,N)  (i + 1) % N    //מקרו המגדיר את מספר השכן מימין
#define THINKING 0    //פילוסוף חושב
#define HUNGRY 1    //פילוסוף רעב (בהמתנה למקלות
#define EATING 2    //פילוסוף אוכל
#define MAX_TIME 5    //זמן מקסימלי שמותר לבזבז על אוכל ו/או חשיבה

int N;    //מספר פילוסופים
sem_t *mutex = NULL; //מצביע לסמפור בינארי
sem_t *eaters = NULL;    //מערך סמפורים של פילוסופים
int* state = NULL;    //מערך שלמים לקביעת מצב פילוסוף (אוכל, רעב, חושב

//תהליך אכילה
void eat(int i) {
	int time = rand() % MAX_TIME;
	printf("Philosopher #%d is eating...\n", i + 1);
	sleep(time);
	printf("Philosopher #%d stopped eating...\n", i + 1);
}
//תהליך חשיבה
void think(int i) {
	int time = rand() % MAX_TIME;
	printf("Philosopher #%d is thinking...\n", i + 1);
	sleep(time);
}
```

//בדיקה האם פילוסוף רעב יכול להתחיל לאכול

```
void test(int i) {
    בודקים האם הפילוסוף רעב והשכנים לא אוכלים (המקלות פנויות//
    ____________________________________________________
    state[i]=EATING;    שינוי סטסוס - הפילוסוף אוכל//
פילוסוף התחיל לאכול, לכן הסמפור שלו עולה כדי שהשכנים שלו לא ינעלו אותו //
    ____________________________________
    }
}
```

//פילוסוף מנסה לקחת מקל

```
void take_sticks(int i) {
    _____ נכנסים לקטע קריטי ונועלים אותו כדי שהמקלות יוכל לקחת רק פילוסוף אחד//

    state[i]=HUNGRY;        הפילוסוף רעב//
    _____ מנסה לקחת 2 מקלות בעזרת פונקציית טסט//
    _____ עוזבים את הקטע הקריטי//
    _____ אם לא הצליח להתחיל לאכול, ננעל. השכנים יוכלו "לפתוח" אותו אחרי שיוכלו //
}
```

//פילוסוף מחזיר מקל

```
void put_sticks(int i) {
    _____ נכנסים לקטע קריטי ונועלים אותו כדי שרק פילוסוף אחד יעבוד עם המקלות//
    state[i]=THINKING;   פילוסוף סיים לאכול, התחיל לחשוב//
    _____ בודקים האם השכן משמאל רעב, ז"א, ממתין למקל, אז נותנים לו סימן שיתחיל לאכול//
    _____ כנ"ל עם השכן מימין//
    _____בשביל מה?//
}
```

//הקוד הזה מתבצע על ידי כל חוט המתאר התנהגות של פילוסוף

```
void* philosopher(void* arg) {
    int i = *((int*)arg);
    while (1 == 1) { //לולאה אין סופית
        _____ קודם כל הפילוסוף חושב//
        _____ אחר כך מנסה לקחת את המקלות//
        _____ אוכל//
        _____ שם את המקלות על השולחן//
    }
}
```

```
int main(int argc, char* argv[]) {
      void*  result;
      if (argc != 2) {
            fprintf(stderr, "Usage: %s <NUMBER_OF_PHILOSOPHERS>\n", argv[0]);
            return 1;
      }
                                                                  //הכנה...
      N = atoi(argv[1]); //המרת כמות הפילוסופים ממחרוזת למספר שלם
      if (N <= 1){fprintf(stderr,"Error by transformation of the argument...\n"); return 2;}//לא פחות מ-2 פילוסופים

      mutex = (sem_t*)malloc(sizeof(sem_t));
      if (sem_init(mutex, 0, 1) != 0) { //יוצרים מיוטקס להפרדה בכניסה לקטע קריטי
            fprintf(stderr, "Error by creating semaphore...\n");        return 3;
      }

      eaters = ____________________//מקצים זיכרון לכמות סמפורים - לכל פילוסוף סמפור
      state = ____________________ //מקצים זיכרון למערך שלמים, כל שלם מייצג מצב פילוסוף מתאים (רעב, אוכל,חושב)

      memset(state, 0, N);

      srand(time(NULL));
      pthread_t *philosophers = (pthread_t*)malloc(N * sizeof(pthread_t)); // מקצים זיכרון לחוטים לפי כמות
הפילוסופים

      int i;
      for (i = 0; i < N; i++) {
            if (sem_init(&eaters[i], 0, 0) != 0) {  //מאתחלים סמפורים של פילוסופים
                  fprintf(stderr, "Error by creating semaphore...\n");          return 3;
            }
      }

      for (i = 0; i < N; i++) {
            if (pthread_create(&philosophers[i], NULL, philosopher, (void*)&i) != 0) {
                  fprintf(stderr, "Error by creating thread\n");          return 2;
            }
            usleep(100000);
      }

   for (i = 0; i < N; i++) {
        if (pthread_join(philosophers[i], &result) != 0) {
                  fprintf(stderr, "Error by joining thread\n");
                  return 3;
        }
   return 0;
}
```