Homework #1
Due: Monday, January 30 at 4pm

INSTRUCTIONS: Fill your answers into the space provided below each question. To submit, upload a zip file containing this file and your code from question 5 to Gradescope.

1. Locate the course syllabus from the class web page. Read the Policy on Academic Honesty, Collaboration and cheating.

   After reading the policy, provide the following statement and the answer to this question:

   I have read **the Policy on Academic Honesty, Collaboration and cheating** & agree to abide by it Zijian Gao [Insert your name].

2. Run the cpu.c code with input. What happens when you run 2 different instances of the program simultaneously. What about 5 different instances?

   When 2 are run simultaneously, the input string is output concurrently and appears to alternate due to the 1 second sleep in the code.

   When 5 are run simultaneously, it becomes clear that the order of the output is not guaranteed.

3. Run the threads.c code with input. Is the program more or less likely to compute the correct value as the size of the input increases? Briefly, explain why?

   Less likely, as the size of the input increases, there is some probability that one thread will read the value of the global variable before it is written by the other thread, and with a larger input size, there is a higher probability that this happens at least once, if not more times.

4. What do the terms Uniprogramming and Multiprogramming mean?

Uniprogramming is a model for an operating system to manage memory where only one program may execute at a time, concurrency is impossible in this model.

Multiprogramming is the model that allows for multiple programs to execute at the same time, which requires specific ways of managing the physical memory access of each executing programs to ensure safety of the operating system/running applications.

5. Write C++ code to implement insert and remove methods for a linked list. Use the provided code template (template.cpp) as a starting point. Make sure you code can compile with g++ and is working correctly.

```
#include <iostream>
#include <cstdio>
#include <stdlib.h>

using namespace std;

// defining the struct element for the linked list
struct LLNode {
   int data;
   LLNode *next;
};

// defining the class for linked list
class linkedlist{
   // the head and tail pointer to the linked list
   LLNode *list;
   LLNode *tail;

public:
   // constructor
   linkedlist(){
      // the first element is a dummy head
```

```
    LLNode *head = (LLNode *)malloc(sizeof(LLNode));
    head->data = 0;
    head->next = NULL;
    list = head;
    tail = head;
}

// insert element x into the list
void insert(int x) {
    //TODO: write code to insert element x to the end of the linkedlist
    LLNode *insert_node = (LLNode *)malloc(sizeof(LLNode));
    insert_node->data = x;
    insert_node->next = NULL;
    tail->next = insert_node;
    tail = insert_node;
}

// remove the first occurrence of element x from the list
void remove(int x) {
    // TODO : write code to remove element x from the list. Also print
    // printf("%d is NOT in the list\n",x); if x is not in the list
    if(!list->next)
        printf("%d is NOT in the list\n",x);

    LLNode *head = list->next;

    if (head->data == x){
        list->next = head->next;
        return;
    }

    LLNode *last = head;
    for (LLNode *iter = head->next; iter; iter = iter->next){
        if (iter->data == x)
        {
            last->next = iter->next;
            return;
        }
        last = iter;
    }

    printf("%d is NOT in the list\n",x);
}
```

```
    void print() {
        LLNode *p = list->next;

        while(p != NULL){
            cout << p->data << ", ";
            p = p->next;
        }

        cout << endl;
    }
};

int main(){
    linkedlist myList;

    cout << "sample tests for code:" << endl;
    myList.insert(2);
    myList.insert(1);
    myList.insert(3);
    myList.insert(4);
    myList.print();

    printf("Removing existing 3:\n");
    myList.remove(3);
    myList.print();

    printf("Removing nonexisting 3\n");
    myList.remove(3);
    myList.print();

    printf("Adding/removing existing 3\n");
    myList.insert(3);
    myList.insert(3);
    myList.remove(3);
    myList.print();

    printf("Removing nonexistant 0:\n");
    myList.remove(0);
    myList.print();

    printf("Inserting, removing 0:\n");

    myList.insert(0);
    myList.remove(0);
```

```
    myList.print();

    //TODO: write your own tests for the code

    return 0;
}
```