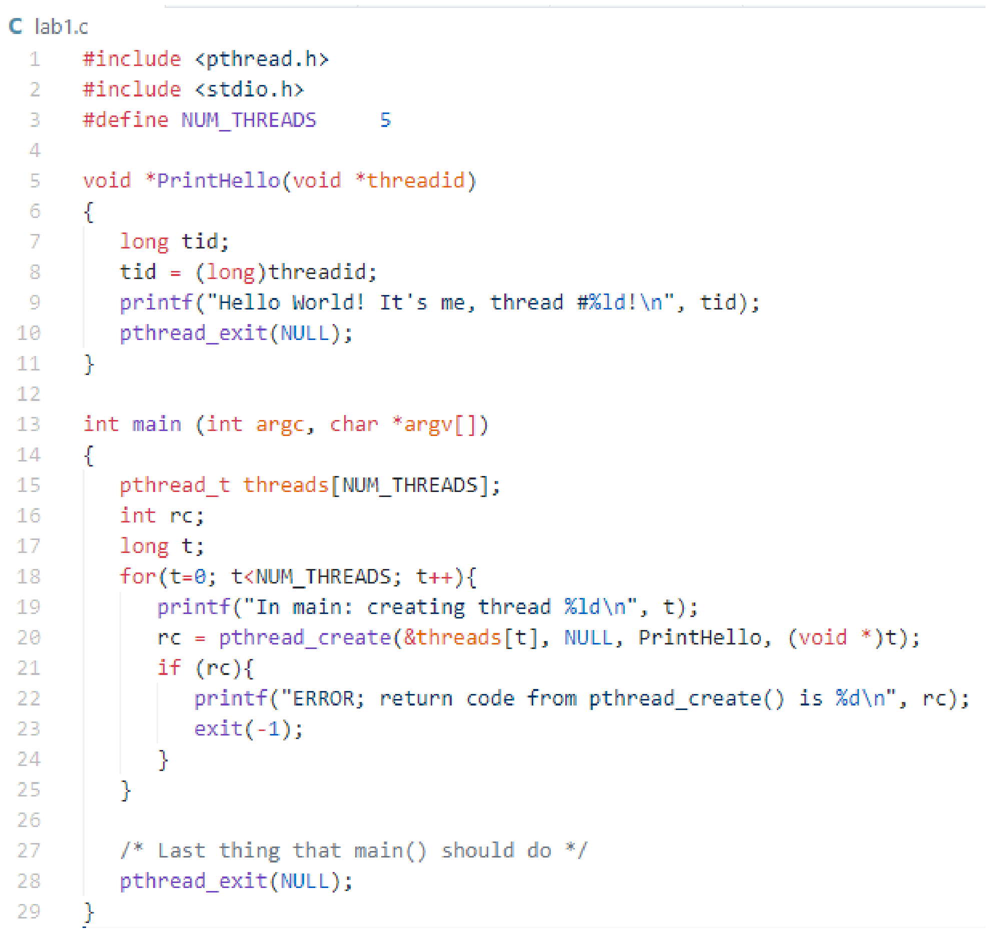
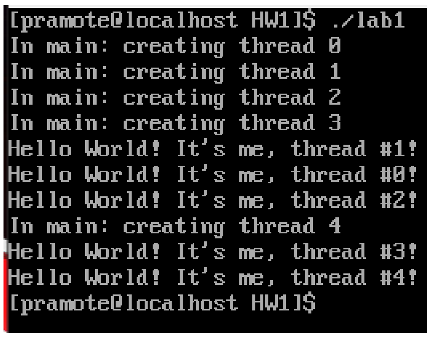
นายปราโมทย์ ศิริภาพ เลขทะเบียน 6210530017

Lab 1



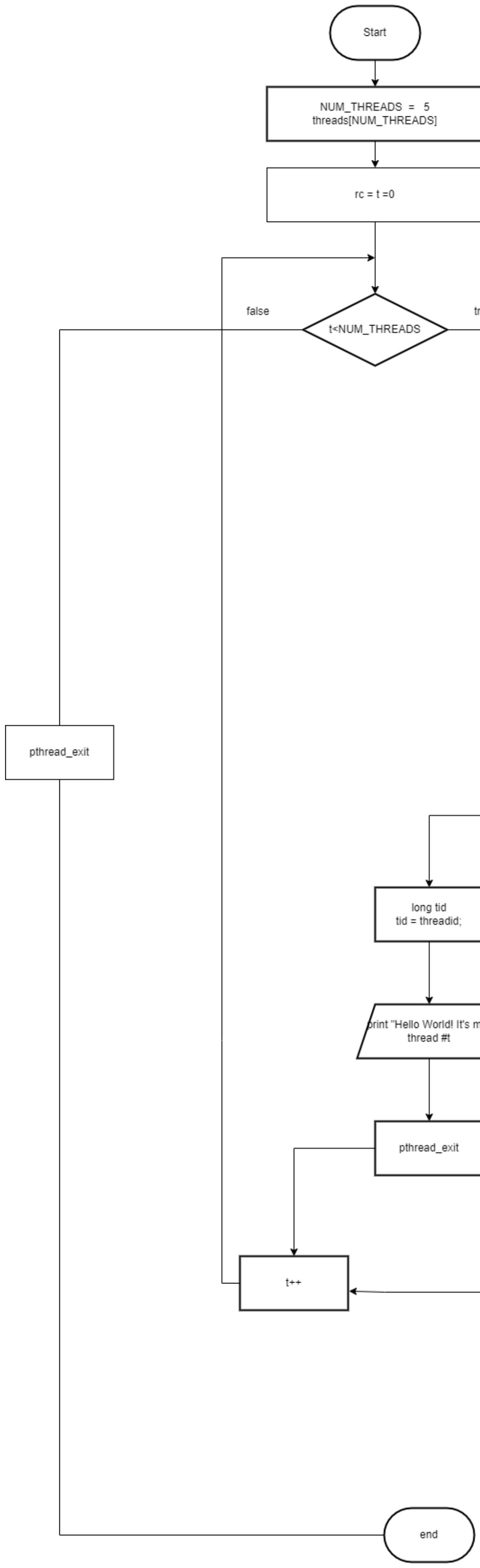
```c
lab1.c
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS     5

void *PrintHello(void *threadid)
{
   long tid;
   tid = (long)threadid;
   printf("Hello World! It's me, thread #%ld!\n", tid);
   pthread_exit(NULL);
}

int main (int argc, char *argv[])
{
   pthread_t threads[NUM_THREADS];
   int rc;
   long t;
   for(t=0; t<NUM_THREADS; t++){
      printf("In main: creating thread %ld\n", t);
      rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
      if (rc){
         printf("ERROR; return code from pthread_create() is %d\n", rc);
         exit(-1);
      }
   }

   /* Last thing that main() should do */
   pthread_exit(NULL);
}
```



```
[pramote@localhost HW1]$ ./lab1
In main: creating thread 0
In main: creating thread 1
In main: creating thread 2
In main: creating thread 3
Hello World! It's me, thread #1!
Hello World! It's me, thread #0!
Hello World! It's me, thread #2!
In main: creating thread 4
Hello World! It's me, thread #3!
Hello World! It's me, thread #4!
[pramote@localhost HW1]$
```

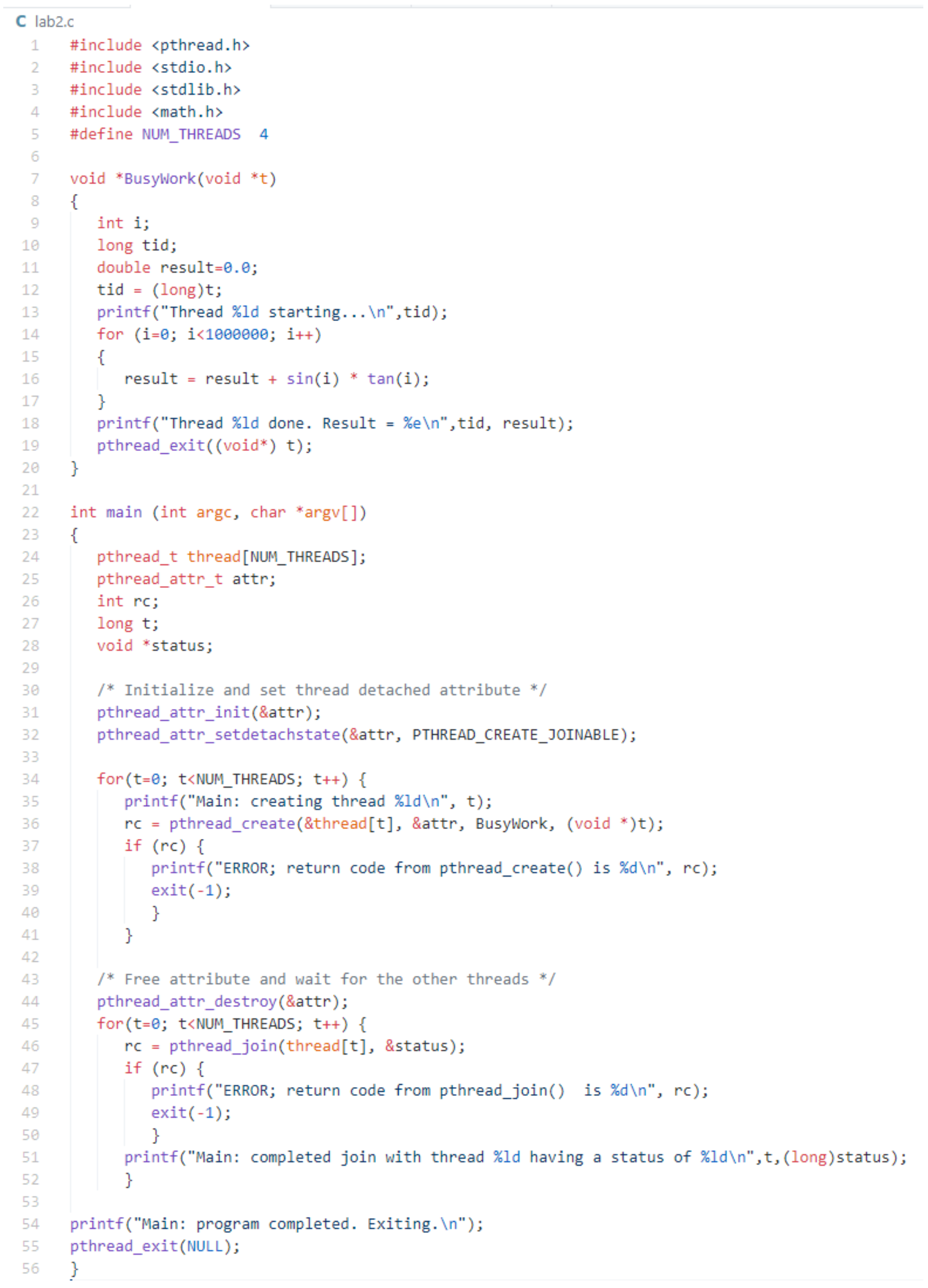
นายปราโมทย์ ศิริภาพ เลขทะเบียน 6210530017

Flow chart lab 1



นายปราโมทย์ ศิริภาพ เลขทะเบียน 6210530017

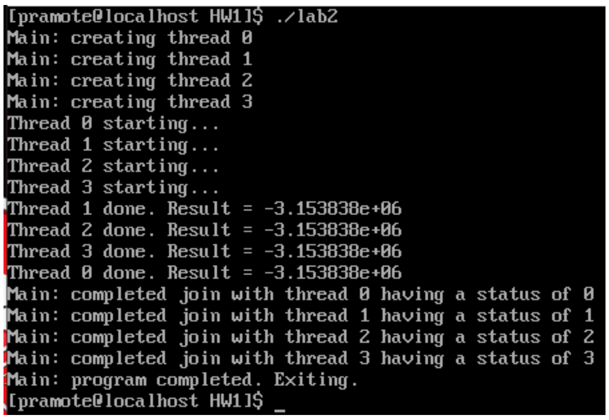
Lab 2



lab2.c

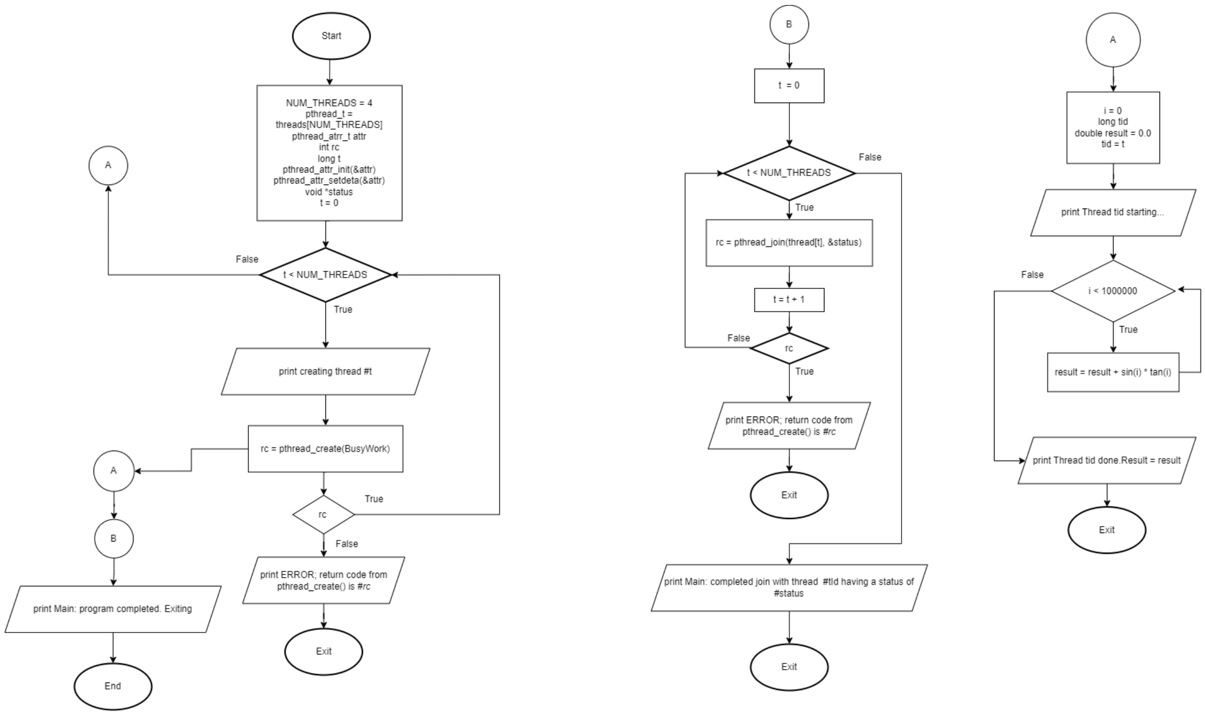
```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define NUM_THREADS  4

void *BusyWork(void *t)
{
    int i;
    long tid;
    double result=0.0;
    tid = (long)t;
    printf("Thread %ld starting...\n",tid);
    for (i=0; i<1000000; i++)
    {
        result = result + sin(i) * tan(i);
    }
    printf("Thread %ld done. Result = %e\n",tid, result);
    pthread_exit((void*) t);
}

int main (int argc, char *argv[])
{
    pthread_t thread[NUM_THREADS];
    pthread_attr_t attr;
    int rc;
    long t;
    void *status;

    /* Initialize and set thread detached attribute */
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

    for(t=0; t<NUM_THREADS; t++) {
        printf("Main: creating thread %ld\n", t);
        rc = pthread_create(&thread[t], &attr, BusyWork, (void *)t);
        if (rc) {
            printf("ERROR; return code from pthread_create() is %d\n", rc);
            exit(-1);
            }
        }

    /* Free attribute and wait for the other threads */
    pthread_attr_destroy(&attr);
    for(t=0; t<NUM_THREADS; t++) {
        rc = pthread_join(thread[t], &status);
        if (rc) {
            printf("ERROR; return code from pthread_join()  is %d\n", rc);
            exit(-1);
            }
        printf("Main: completed join with thread %ld having a status of %ld\n",t,(long)status);
        }

printf("Main: program completed. Exiting.\n");
pthread_exit(NULL);
}
```

นายปราโมทย์ ศิริภาพ เลขทะเบียน 6210530017



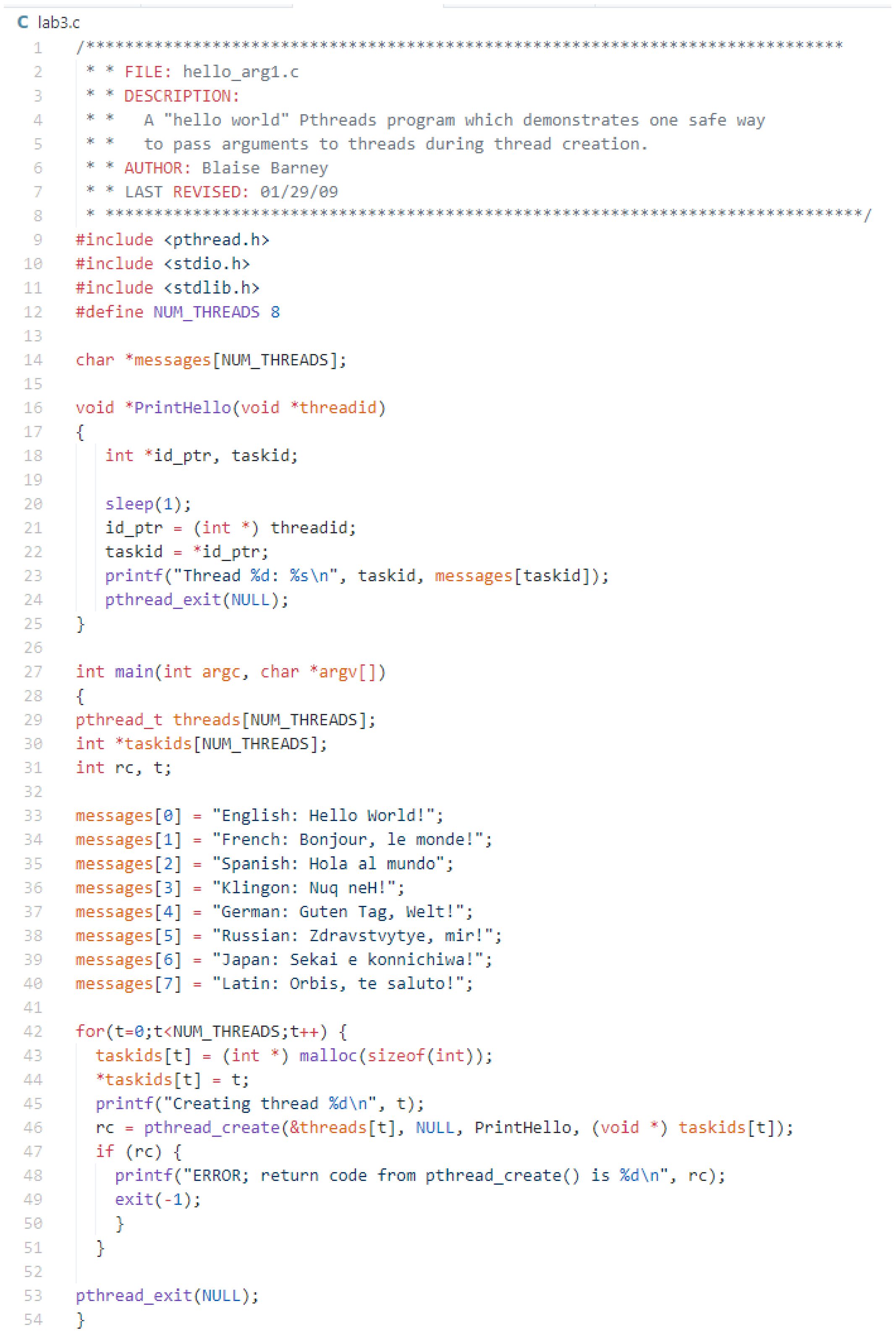
```
[pramote@localhost HW1]$ ./lab2
Main: creating thread 0
Main: creating thread 1
Main: creating thread 2
Main: creating thread 3
Thread 0 starting...
Thread 1 starting...
Thread 2 starting...
Thread 3 starting...
Thread 1 done. Result = -3.153838e+06
Thread 2 done. Result = -3.153838e+06
Thread 3 done. Result = -3.153838e+06
Thread 0 done. Result = -3.153838e+06
Main: completed join with thread 0 having a status of 0
Main: completed join with thread 1 having a status of 1
Main: completed join with thread 2 having a status of 2
Main: completed join with thread 3 having a status of 3
Main: program completed. Exiting.
[pramote@localhost HW1]$ _
```

Flow chart lab 2

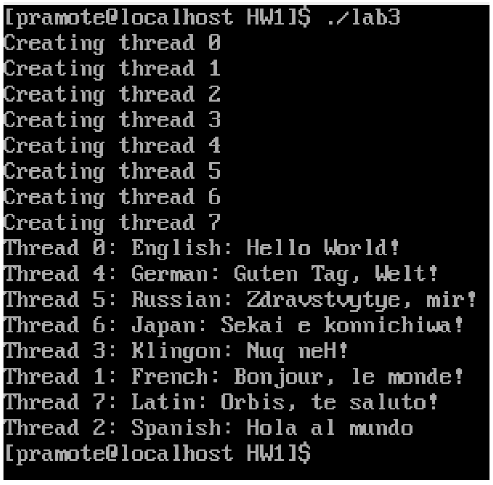

Start
NUM_THREADS = 4
pthread_t =
threads[NUM_THREADS]
pthread_atrr_t attr
int rc
long t
pthread_attr_init(&attr)
pthread_attr_setdeta(&attr)
void *status
t = 0
A
False
t < NUM_THREADS
True
print creating thread #t
rc = pthread_create(BusyWork)
A
B
rc
True
False
print ERROR; return code from pthread_create() is #rc
Exit
print Main: program completed. Exiting
End
B
t = 0
t < NUM_THREADS
False
True
rc = pthread_join(thread[t], &status)
t = t + 1
False
rc
True
print ERROR; return code from pthread_create() is #rc
Exit
print Main: completed join with thread #tId having a status of #status
Exit
A
i = 0
long tid
double result = 0.0
tid = t
print Thread tid starting...
False
i < 1000000
True
result = result + sin(i) * tan(i)
print Thread tid done.Result = result
Exit

นายปราโมทย์ ศิริภาพ เลขทะเบียน 6210530017

Lab 3

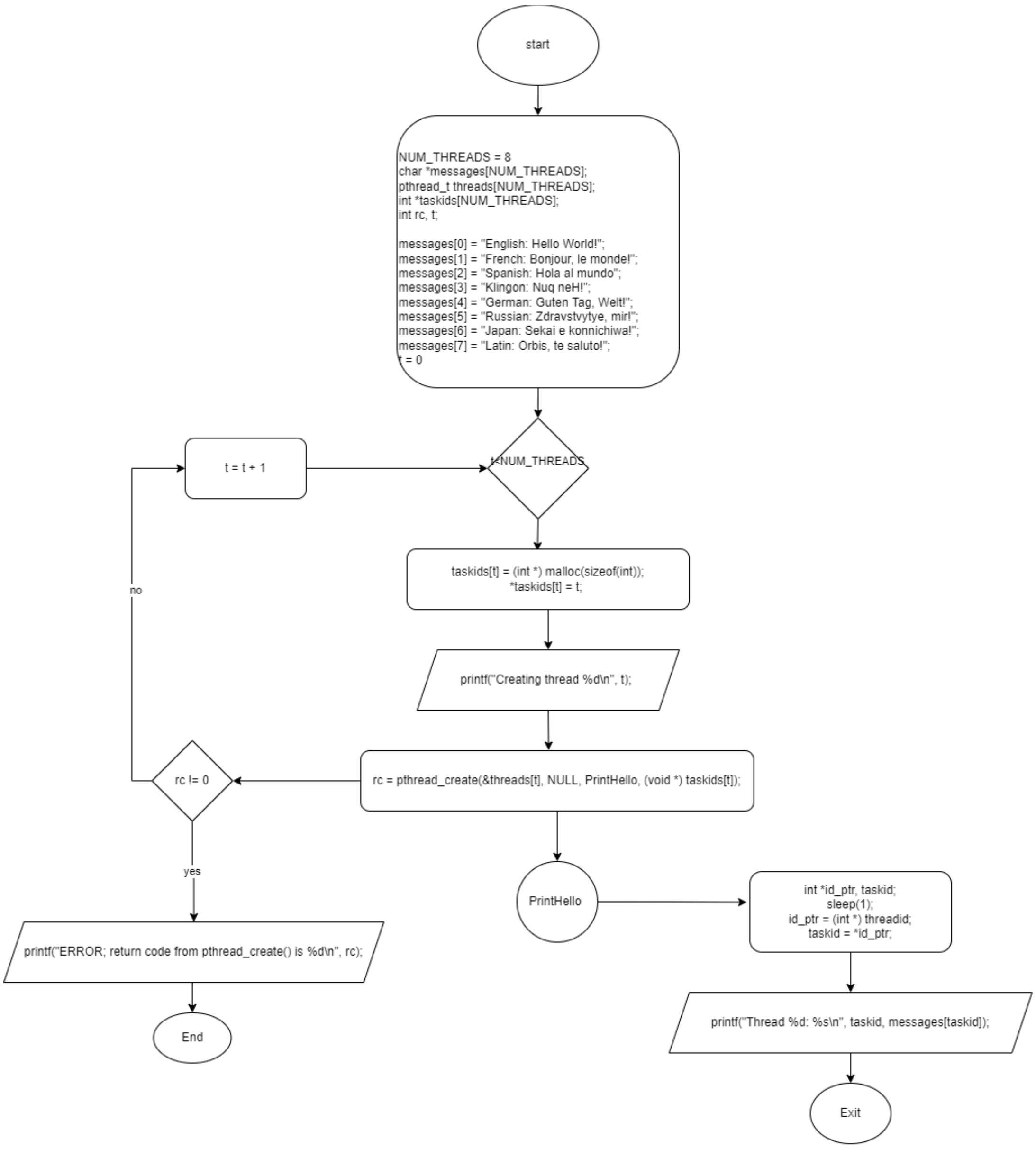

C lab3.c

```c
/******************************************************************************
 * * FILE: hello_arg1.c
 * * DESCRIPTION:
 * *   A "hello world" Pthreads program which demonstrates one safe way
 * *   to pass arguments to threads during thread creation.
 * * AUTHOR: Blaise Barney
 * * LAST REVISED: 01/29/09
 * ******************************************************************************/
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS 8

char *messages[NUM_THREADS];

void *PrintHello(void *threadid)
{
   int *id_ptr, taskid;

   sleep(1);
   id_ptr = (int *) threadid;
   taskid = *id_ptr;
   printf("Thread %d: %s\n", taskid, messages[taskid]);
   pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
pthread_t threads[NUM_THREADS];
int *taskids[NUM_THREADS];
int rc, t;

messages[0] = "English: Hello World!";
messages[1] = "French: Bonjour, le monde!";
messages[2] = "Spanish: Hola al mundo";
messages[3] = "Klingon: Nuq neH!";
messages[4] = "German: Guten Tag, Welt!";
messages[5] = "Russian: Zdravstvytye, mir!";
messages[6] = "Japan: Sekai e konnichiwa!";
messages[7] = "Latin: Orbis, te saluto!";

for(t=0;t<NUM_THREADS;t++) {
  taskids[t] = (int *) malloc(sizeof(int));
  *taskids[t] = t;
  printf("Creating thread %d\n", t);
  rc = pthread_create(&threads[t], NULL, PrintHello, (void *) taskids[t]);
  if (rc) {
    printf("ERROR; return code from pthread_create() is %d\n", rc);
    exit(-1);
    }
  }

pthread_exit(NULL);
}
```



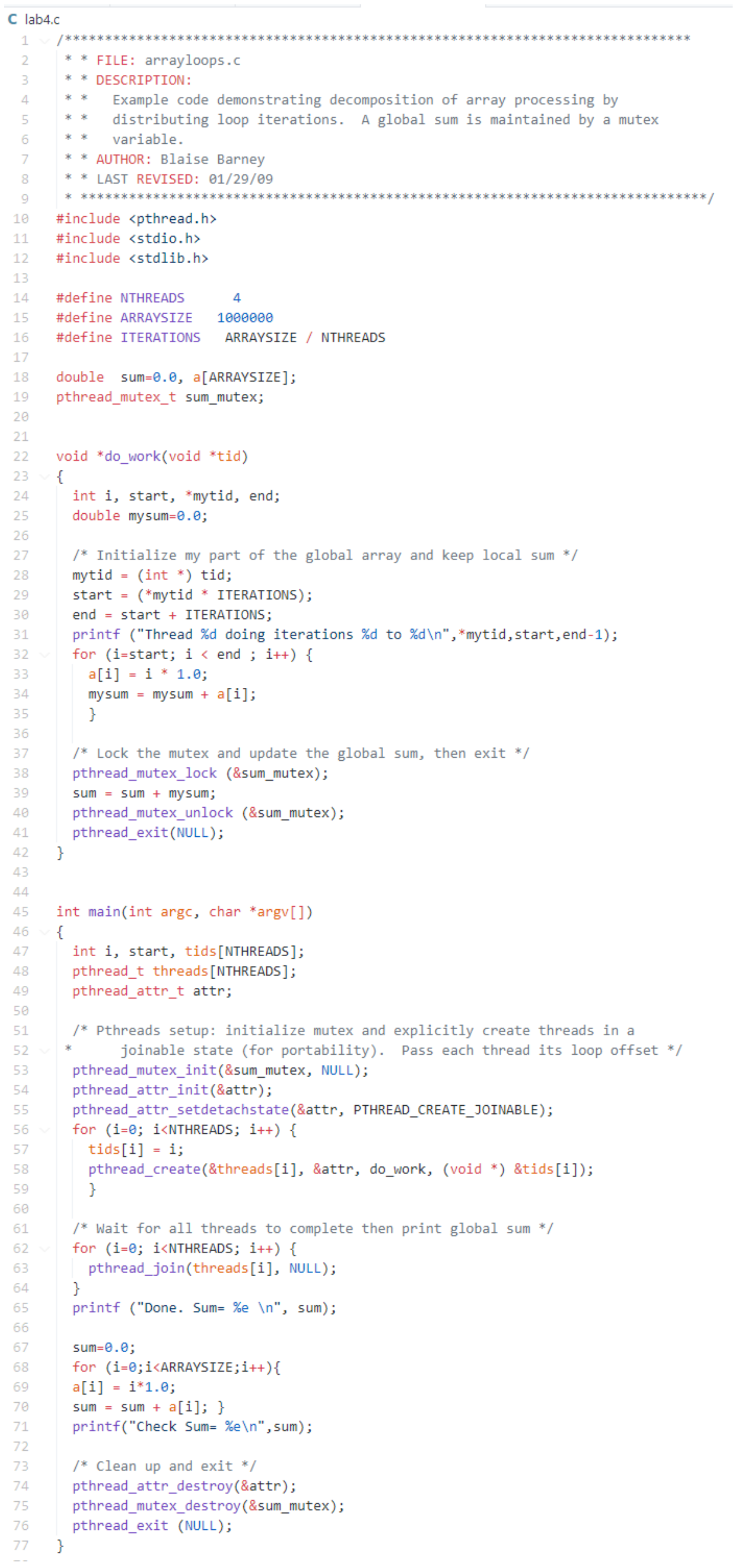
```
[pramote@localhost HW1]$ ./lab3
Creating thread 0
Creating thread 1
Creating thread 2
Creating thread 3
Creating thread 4
Creating thread 5
Creating thread 6
Creating thread 7
Thread 0: English: Hello World!
Thread 4: German: Guten Tag, Welt!
Thread 5: Russian: Zdravstvytye, mir!
Thread 6: Japan: Sekai e konnichiwa!
Thread 3: Klingon: Nuq neH!
Thread 1: French: Bonjour, le monde!
Thread 7: Latin: Orbis, te saluto!
Thread 2: Spanish: Hola al mundo
[pramote@localhost HW1]$
```

Flow chart lab 3


start
NUM_THREADS = 8
char *messages[NUM_THREADS];
pthread_t threads[NUM_THREADS];
int *taskids[NUM_THREADS];
int rc, t;
messages[0] = "English: Hello World!";
messages[1] = "French: Bonjour, le monde!";
messages[2] = "Spanish: Hola al mundo";
messages[3] = "Klingon: Nuq neH!";
messages[4] = "German: Guten Tag, Welt!";
messages[5] = "Russian: Zdravstvytye, mir!";
messages[6] = "Japan: Sekai e konnichiwa!";
messages[7] = "Latin: Orbis, te saluto!";
t = 0
t<NUM_THREADS
t = t + 1
taskids[t] = (int *) malloc(sizeof(int));
*taskids[t] = t;
printf("Creating thread %d\n", t);
rc = pthread_create(&threads[t], NULL, PrintHello, (void *) taskids[t]);
rc != 0
no
yes
printf("ERROR; return code from pthread_create() is %d\n", rc);
End
PrintHello
int *id_ptr, taskid;
sleep(1);
id_ptr = (int *) threadid;
taskid = *id_ptr;
printf("Thread %d: %s\n", taskid, messages[taskid]);
Exit

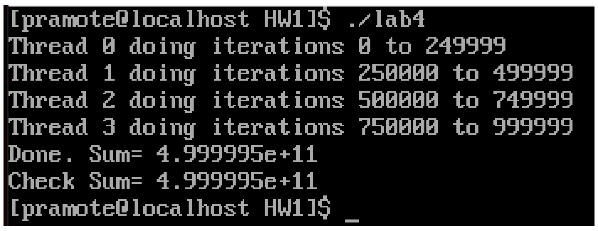
นายปราโมทย์ ศิริภาพ เลขทะเบียน 6210530017

Lab 4



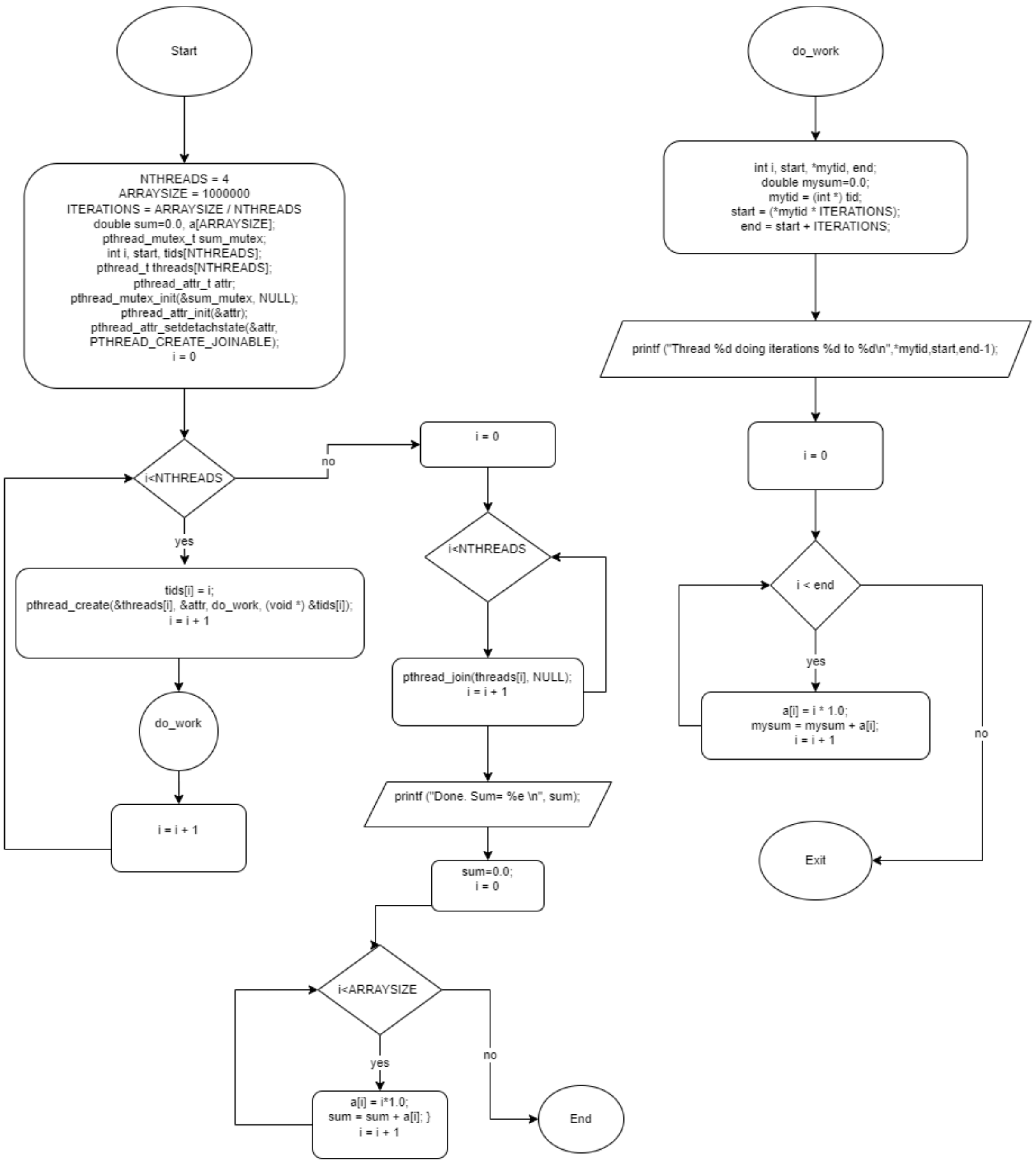
C lab4.c

```c
/******************************************************************************
 * * FILE: arrayloops.c
 * * DESCRIPTION:
 * *   Example code demonstrating decomposition of array processing by
 * *   distributing loop iterations.  A global sum is maintained by a mutex
 * *   variable.
 * * AUTHOR: Blaise Barney
 * * LAST REVISED: 01/29/09
 * ****************************************************************************/
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define NTHREADS      4
#define ARRAYSIZE   1000000
#define ITERATIONS   ARRAYSIZE / NTHREADS

double  sum=0.0, a[ARRAYSIZE];
pthread_mutex_t sum_mutex;


void *do_work(void *tid)
{
  int i, start, *mytid, end;
  double mysum=0.0;

  /* Initialize my part of the global array and keep local sum */
  mytid = (int *) tid;
  start = (*mytid * ITERATIONS);
  end = start + ITERATIONS;
  printf ("Thread %d doing iterations %d to %d\n",*mytid,start,end-1);
  for (i=start; i < end ; i++) {
    a[i] = i * 1.0;
    mysum = mysum + a[i];
    }

  /* Lock the mutex and update the global sum, then exit */
  pthread_mutex_lock (&sum_mutex);
  sum = sum + mysum;
  pthread_mutex_unlock (&sum_mutex);
  pthread_exit(NULL);
}


int main(int argc, char *argv[])
{
  int i, start, tids[NTHREADS];
  pthread_t threads[NTHREADS];
  pthread_attr_t attr;

  /* Pthreads setup: initialize mutex and explicitly create threads in a
 *      joinable state (for portability).  Pass each thread its loop offset */
  pthread_mutex_init(&sum_mutex, NULL);
  pthread_attr_init(&attr);
  pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
  for (i=0; i<NTHREADS; i++) {
    tids[i] = i;
    pthread_create(&threads[i], &attr, do_work, (void *) &tids[i]);
    }

  /* Wait for all threads to complete then print global sum */
  for (i=0; i<NTHREADS; i++) {
    pthread_join(threads[i], NULL);
  }
  printf ("Done. Sum= %e \n", sum);

  sum=0.0;
  for (i=0;i<ARRAYSIZE;i++){
  a[i] = i*1.0;
  sum = sum + a[i]; }
  printf("Check Sum= %e\n",sum);

  /* Clean up and exit */
  pthread_attr_destroy(&attr);
  pthread_mutex_destroy(&sum_mutex);
  pthread_exit (NULL);
}
```



```
[pramote@localhost HW1]$ ./lab4
Thread 0 doing iterations 0 to 249999
Thread 1 doing iterations 250000 to 499999
Thread 2 doing iterations 500000 to 749999
Thread 3 doing iterations 750000 to 999999
Done. Sum= 4.999995e+11
Check Sum= 4.999995e+11
[pramote@localhost HW1]$ _
```

Flow chart lab 4



Start
NTHREADS = 4
ARRAYSIZE = 1000000
ITERATIONS = ARRAYSIZE / NTHREADS
double sum=0.0, a[ARRAYSIZE];
pthread_mutex_t sum_mutex;
int i, start, tids[NTHREADS];
pthread_t threads[NTHREADS];
pthread_attr_t attr;
pthread_mutex_init(&sum_mutex, NULL);
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
i = 0
i<NTHREADS
yes
tids[i] = i; pthread_create(&threads[i], &attr, do_work, (void *) &tids[i]); i = i + 1
do_work
i = i + 1
no
i = 0
i<NTHREADS
pthread_join(threads[i], NULL); i = i + 1
printf ("Done. Sum= %e \n", sum);
sum=0.0; i = 0
i<ARRAYSIZE
yes
a[i] = i*1.0; sum = sum + a[i]; } i = i + 1
no
End
do_work
int i, start, *mytid, end; double mysum=0.0; mytid = (int *) tid; start = (*mytid * ITERATIONS); end = start + ITERATIONS;
printf ("Thread %d doing iterations %d to %d\n",*mytid,start,end-1);
i = 0
i < end
yes
a[i] = i * 1.0; mysum = mysum + a[i]; i = i + 1
no
Exit