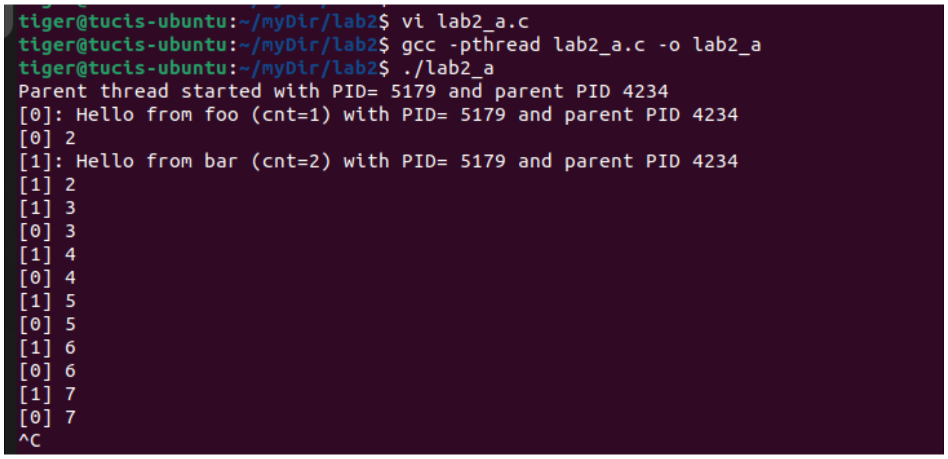
Lab2: Thread

1. Code

```
#include <stdio.h>
#include <unistd.h>     // getpid, getppid, sleep
#include <pthread.h>    // pthread_t, pthread_create, pthread_join
#include <stdint.h>     // intptr_t for portable int<->pointer casts

#define N 2                          // use a compile-time constant

void *thread(void *vargp);
const char **ptr;                    // global pointer to array of strings

int main(void)
{
   pthread_t tid[N];                 // thread IDs

   static const char *msgs[N] = {      // sized by #define N (not a VLA)
      "Hello from foo",
      "Hello from bar"
   };

   printf("Parent thread started with PID= %d and parent PID %d\n",
         getpid(), getppid());

   ptr = msgs;                       // let worker threads see the messages

   for (int i = 0; i < N; i++) {
      pthread_create(&tid[i], NULL, thread, (void *)(intptr_t)i);
   }

   for (int i = 0; i < N; i++) {
      pthread_join(tid[i], NULL);
   }

   return 0;
}

void *thread(void *vargp)
{
   int myid = (int)(intptr_t)vargp;     // recover small integer index portably
   static int cnt = 0;               // shared across threads (demonstration)

   printf("[%d]: %s (cnt=%d) with PID= %d and parent PID %d\n",
         myid, ptr[myid], ++cnt, getpid(), getppid());

   int i = cnt;
   for (;; i++) {                    // infinite loop
```

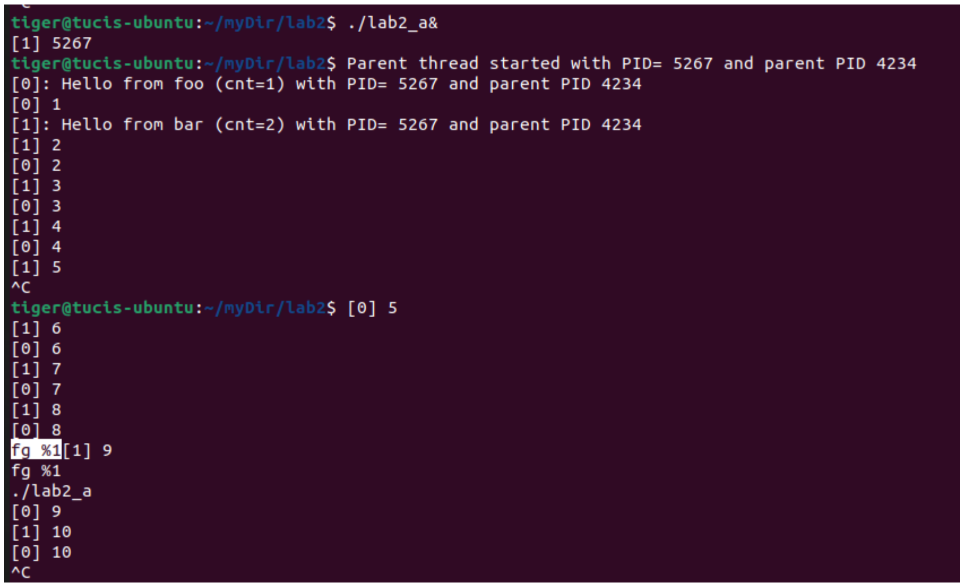
```
        printf("[%d] %d\n", myid, i);
        sleep(cnt);
    }

    return NULL;
}
```

Output:



```
tiger@tucis-ubuntu:~/myDir/lab2$ vi lab2_a.c
tiger@tucis-ubuntu:~/myDir/lab2$ gcc -pthread lab2_a.c -o lab2_a
tiger@tucis-ubuntu:~/myDir/lab2$ ./lab2_a
Parent thread started with PID= 5179 and parent PID 4234
[0]: Hello from foo (cnt=1) with PID= 5179 and parent PID 4234
[0] 2
[1]: Hello from bar (cnt=2) with PID= 5179 and parent PID 4234
[1] 2
[1] 3
[0] 3
[1] 4
[0] 4
[1] 5
[0] 5
[1] 6
[0] 6
[1] 7
[0] 7
^C
```

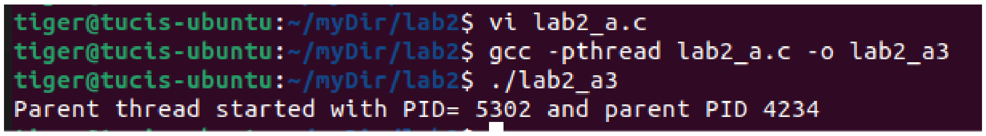
2. Output:



```
tiger@tucis-ubuntu:~/myDir/lab2$ ./lab2_a&
[1] 5267
tiger@tucis-ubuntu:~/myDir/lab2$ Parent thread started with PID= 5267 and parent PID 4234
[0]: Hello from foo (cnt=1) with PID= 5267 and parent PID 4234
[0] 1
[1]: Hello from bar (cnt=2) with PID= 5267 and parent PID 4234
[1] 2
[0] 2
[1] 3
[0] 3
[1] 4
[0] 4
[1] 5
^C
tiger@tucis-ubuntu:~/myDir/lab2$ [0] 5
[1] 6
[0] 6
[1] 7
[0] 7
[1] 8
[0] 8
fg %1[1] 9
fg %1
./lab2_a
[0] 9
[1] 10
[0] 10
^C
```

pressing Ctrl-C. Nothing happens to the background job, because Ctrl-C is delivered only to the **foreground** job. Brought it to foreground using fg %1 then used Ctrl-C to stop it.

3. Output:



```
tiger@tucis-ubuntu:~/myDir/lab2$ vi lab2_a.c
tiger@tucis-ubuntu:~/myDir/lab2$ gcc -pthread lab2_a.c -o lab2_a3
tiger@tucis-ubuntu:~/myDir/lab2$ ./lab2_a3
Parent thread started with PID= 5302 and parent PID 4234
```

Commenting out the two pthread_join lines and running it again. The process exits immediately after creating the threads, because main returns and the process terminate, which kills all threads.

4. Table:

Variable	**Where declared**	**Storage class / duration**
ptr	global	**static storage** (global), external linkage
N, i (in main)	local in main	**automatic** (stack)
tid (in main)	local in main	**automatic** (stack)
msgs (in main)	local in main	**automatic** (stack) array of pointers
string literals "Hello from ..."	in source	**static storage** (read-only)
vargp (param)	param of thread	**automatic** (stack of that thread)
myid (in thread)	local in thread	**automatic** (stack of that thread)
cnt (in thread)	static local	**static storage** (one shared instance)
i (in thread)	local in thread	**automatic** (stack of that thread)

5. Table:

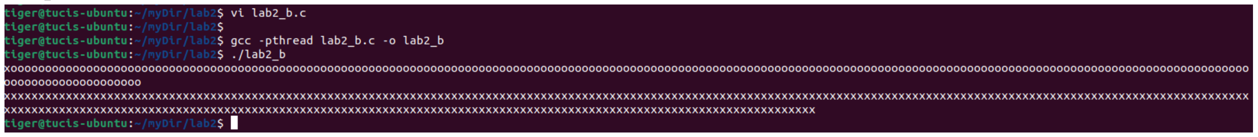
Item	**Segment**
Program instructions	**Text (.text)**
Global ptr (zero-initialized)	**BSS (.bss)**
cnt (static int, init 0)	**BSS (.bss)**
msgs array (local)	**Stack** (in the main thread's stack)
tid, i, N (locals in main)	**Stack** (main thread)
vargp, myid, i in thread	**Stack** (each worker's own thread stack)

6. Code 2:

```
#include <stdio.h>
#include <pthread.h>

/* Structure used to pass multiple parameters to a thread */
struct char_print_parms {
   char character;
   int  count;
};
```

```
/* Thread start routine */
void *char_print(void *parms)
{
    struct char_print_parms *p = (struct char_print_parms *)parms;

    for (int i = 0; i < p->count; i++) {
        printf("%c", p->character);
    }
    printf("\n");

    return NULL;
}

int main(void)
{
    pthread_t thread1_id;
    pthread_t thread2_id;

    struct char_print_parms thread1_args;
    struct char_print_parms thread2_args;

    /* Create a new thread to print 300 'x's. */
    thread1_args.character = 'x';
    thread1_args.count     = 300;
    pthread_create(&thread1_id, NULL, char_print, &thread1_args);

    /* Create a new thread to print 200 'o's. */
    thread2_args.character = 'o';
    thread2_args.count     = 200;
    pthread_create(&thread2_id, NULL, char_print, &thread2_args);

    /* Wait for both threads to finish */
    pthread_join(thread1_id, NULL);
    pthread_join(thread2_id, NULL);

    return 0;
}
```

Output:



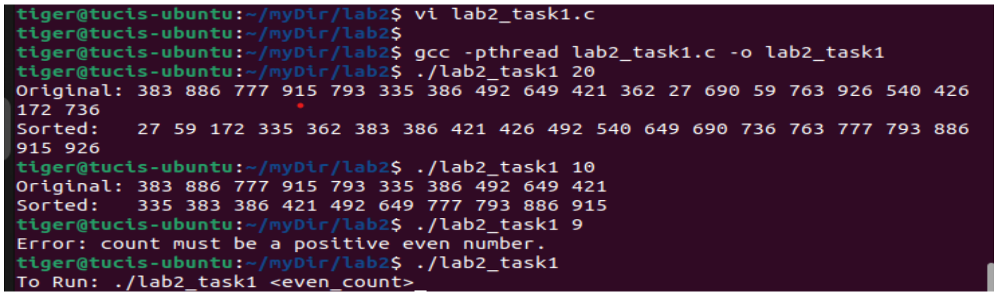
```
tiger@tucis-ubuntu:~/myDir/lab2$ vi lab2_b.c
tiger@tucis-ubuntu:~/myDir/lab2$
tiger@tucis-ubuntu:~/myDir/lab2$ gcc -pthread lab2_b.c -o lab2_b
tiger@tucis-ubuntu:~/myDir/lab2$ ./lab2_b
xoooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooo
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
tiger@tucis-ubuntu:~/myDir/lab2$
```

7. Task1:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <pthread.h>
```

```
/*  Global data shared by all threads  */
static int *g_input  = NULL;   // pointer to the original array
static int *g_output = NULL;   // global pointer to the destination array used by the merge thread at last
static int  g_n      = 0;      // number of elements in the arrays must be even. Stated in Print

/* structure for merging and sorting  */
typedef struct {
    int *arr;     // pointer to the array to be sorted
    int start;    // start index of the subarray this thread will sort
    int end;      // end index of the subarray this thread will sort
} SortArgs;

typedef struct {
    int *src;         // pointer to the source array that holds two sorted halves did in g_input
    int left_lo;      // left half start index
    int left_hi;      // left half end   index
    int right_lo;     // right half start index
    int right_hi;     // right half end   index
    int *dst;         // pointer to destination array to write the merged result
} MergeArgs;

/*
   bubble_sort_range cunction shame as given example in class
   Inputs array start and end index (as half)
   Output : sorted array
   */
static void bubble_sort_range(int *a, int start, int end)
{
    int len = end - start;                 // number of elements in this slice

    for (int pass = 0; pass < len - 1; ++pass) {
        int swapped = 0;
        int upto = (len - 1) - pass;
        for (int i = start; i < start + upto; ++i) {
            if (a[i] > a[i + 1]) {
                int tmp = a[i];
                a[i] = a[i + 1];
                a[i + 1] = tmp;
                swapped = 1;
            }
        }
        if (!swapped) break;
    }
}

/*
   sort_thread calling sort function for threads
   Input  : vp structure describing which subarray to sort
   Output : returns NULL to pthreads and arr[start..end)
         becomes sorted in place.
   */
static void *sort_thread(void *vp)
{
    SortArgs *args = (SortArgs *)vp;              // unpack the struct
    bubble_sort_range(args->arr, args->start, args->end);  // sort that slice
    return NULL;                                   // pthreads expects a void* return; we return nothing
}

/*
   merge_thread that merges two sorted thred
   Input  : vp structure describing two sorted halves in src and a dst
   Output : returns NULL and dst[0..(right_hi-left_lo)) is the
         fully merged globally sorted array.
    */
static void *merge_thread(void *vp)
{
    MergeArgs *m = (MergeArgs *)vp;

    int i = m->left_lo;                 // pointer for left  half
    int j = m->right_lo;                 // pointer for right half
    int k = 0;                         // pointer for destination

    // merge by always copy the smaller head element to dst
    while (i < m->left_hi && j < m->right_hi) {
        if (m->src[i] <= m->src[j]) {    // take from left
            m->dst[k++] = m->src[i++];
        } else {
            m->dst[k++] = m->src[j++];
        }
    }

    // Copy any leftovers from the left half
    while (i < m->left_hi) {
        m->dst[k++] = m->src[i++];
    }

    // Copy any leftovers from the right half
    while (j < m->right_hi) {
        m->dst[k++] = m->src[j++];
    }

    return NULL;
}

/*
```

```
   function for printing
   */
static void print_array(const char *label, const int *a, int n)
{
    printf("%s", label);
    for (int i = 0; i < n; ++i)
        printf("%s%d", (i ? " " : ""), a[i]);
    printf("\n");
}


int main(int argc, char **argv)
{
    // Check for exactly one command-line argument if not give error messgae
    if (argc != 2) {
        fprintf(stderr, "To Run: %s <even_count>\n", argv[0]);
        return 1;
    }

    g_n = atoi(argv[1]);                  // read N from argv[1]
    if (g_n <= 0 || (g_n % 2) != 0) {     // N must be positive and even if not give error message
        fprintf(stderr, "Error: count must be a positive even number.\n");
        return 1;
    }

    // Allocate the two arrays  input and output
    g_input  = (int *)malloc((size_t)g_n * sizeof(int));
    g_output = (int *)malloc((size_t)g_n * sizeof(int));
    if (!g_input || !g_output) {          // check allocation
        fprintf(stderr, "Error: malloc failed.\n");
        free(g_input);
        free(g_output);
        return 1;
    }

    // Fill input with pseudorandom values within 1000
    for (int i = 0; i < g_n; ++i)
        g_input[i] = rand() % 1000;

    // Show the unsorted array using print function
    print_array("Original: ", g_input, g_n);

    // Create two sorting threads, each working on one half of g_input
    pthread_t t_sort_left, t_sort_right;

    int mid = g_n / 2;                              // split input

    SortArgs left_args  = { g_input, 0,   mid   };   // left  half is [0, mid)
    SortArgs right_args = { g_input, mid, g_n   };   // right half is [mid, g_n)

    // thread creation
    if (pthread_create(&t_sort_left,  NULL, sort_thread, &left_args) != 0) {
        fprintf(stderr, "Error: pthread_create (left) failed.\n");
        free(g_input); free(g_output); return 1;
    }
    if (pthread_create(&t_sort_right, NULL, sort_thread, &right_args) != 0) {
        fprintf(stderr, "Error: pthread_create (right) failed.\n");
        free(g_input); free(g_output); return 1;
    }

    // Wait until both halves are sorted
    pthread_join(t_sort_left,  NULL);
    pthread_join(t_sort_right, NULL);

    // Prepare and start the merge thread to combine the two sorted halves into g_output
    pthread_t t_merge;
    MergeArgs margs = {
        .src      = g_input,  // read from g_input as contains two sorted halves now
        .left_lo  = 0,
        .left_hi  = mid,
        .right_lo = mid,
        .right_hi = g_n,
        .dst      = g_output  // write the fully merged, sorted array here
    };
    //creating merge thread
    if (pthread_create(&t_merge, NULL, merge_thread, &margs) != 0) {
        fprintf(stderr, "Error: pthread_create (merge) failed.\n");
        free(g_input); free(g_output); return 1;
    }

    // Wait for the merge to complete
    pthread_join(t_merge, NULL);

    // Print the final sorted array
    print_array("Sorted:   ", g_output, g_n);


    return 0;
}
```
Output:



```
tiger@tucis-ubuntu:~/myDir/lab2$ vi lab2_task1.c
tiger@tucis-ubuntu:~/myDir/lab2$
tiger@tucis-ubuntu:~/myDir/lab2$ gcc -pthread lab2_task1.c -o lab2_task1
tiger@tucis-ubuntu:~/myDir/lab2$ ./lab2_task1 20
Original: 383 886 777 915 793 335 386 492 649 421 362 27 690 59 763 926 540 426
172 736
Sorted:   27 59 172 335 362 383 386 421 426 492 540 649 690 736 763 777 793 886
915 926
tiger@tucis-ubuntu:~/myDir/lab2$ ./lab2_task1 10
Original: 383 886 777 915 793 335 386 492 649 421
Sorted:   335 383 386 421 492 649 777 793 886 915
tiger@tucis-ubuntu:~/myDir/lab2$ ./lab2_task1 9
Error: count must be a positive even number.
tiger@tucis-ubuntu:~/myDir/lab2$ ./lab2_task1
To Run: ./lab2_task1 <even_count>
```

8. Task 2: (This code only works for two or four threads)

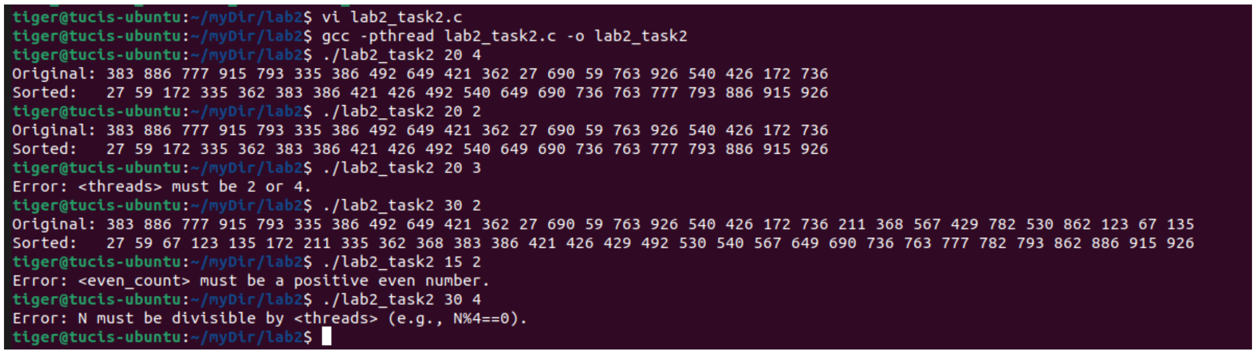
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <pthread.h>
/* This code only works for two or four threads*/

/*  Global data shared by all threads*/
static int *g_input  = NULL;   // pointer to original array
static int *g_output = NULL;   // pointer to output array
static int  g_n      = 0;      // total number of elements must be multiple of 2
static int  g_T      = 0;      // number of sorting threads 2 or 4

/* Structure for sorting thread */
typedef struct {
    int *arr;   // array pointer (g_input)
    int start;  // start index of this thread's slice
    int end;    // end index of this thread's slice
} SortArgs;

/* Structure for the merge thread */
typedef struct {
    const int *src;  // pointer to g_input
    int       *dst;  // pointer to g_output
    int        n;    // total length of the whole array
    int        T;    // number of sorted chunks either 2 or 4
    int       *tmp;  // buffer used only when Thread==4
} MergeArgs;

/*
   bubble_sort_range cunction shame as given example in class
   Inputs array start and end index (as half)
   Output : sorted array
   */

static void bubble_sort_range(int *a, int start, int end)
{
    int len = end - start;
    for (int pass = 0; pass < len - 1; ++pass) {
        int swapped = 0;
        int upto = (len - 1) - pass;
        for (int i = start; i < start + upto; ++i) {
            if (a[i] > a[i + 1]) {
                int tmp = a[i];
                a[i] = a[i + 1];
                a[i + 1] = tmp;
                swapped = 1;
            }
        }
        if (!swapped) break;
    }
}
/*
   sort_thread calling sort function for threads
   Input  : vp structure describing which subarray to sort
   Output : returns NULL to pthreads and arr[start..end)
            becomes sorted in place.
   */

static void *sort_thread(void *vp)
{
    SortArgs *args = (SortArgs *)vp;
    bubble_sort_range(args->arr, args->start, args->end);
    return NULL;
}

/*
   merge_two Threads
   Inputs : src ; source array that holds two sorted ranges
            a_lo : left  range start
            a_hi : left  range end
            b_lo : right range start
            b_hi : right range end
            dst  : destination array to write merged output
            k_lo : starting index in dst
   Output : writes merged (sorted) contents of the two ranges into dst
            beginning at dst[k_lo]
  */
static void merge_two(const int *src,
                      int a_lo, int a_hi,
                      int b_lo, int b_hi,
                      int *dst, int k_lo)
{
    int i = a_lo;                // POINTER for left  range
    int j = b_lo;                // POINTER for right range
    int k = k_lo;                 // POINTER for destination

    // Merge until one of the ranges is done
    while (i < a_hi && j < b_hi)
        dst[k++] = (src[i] <= src[j]) ? src[i++] : src[j++];

    // Copy any leftovers from the left range
    while (i < a_hi) dst[k++] = src[i++];

    // Copy any leftovers from the right range
    while (j < b_hi) dst[k++] = src[j++];
}

/*
   merge_thread 2 or 4
   Inputs : vp structure describing how many chunks exist Thread=2 or Thread==4,
               the source array (src), destination (dst), and scratch tmp
   Output : returns NULL and dst[0..n) becomes globally sorted
   Logic  :
     - T==2 : merge left half [0..n/2) and right half [n/2..n) into dst
     - T==4 : pairwise merge quarters (0,1) and (2,3) into tmp, then
            merge the two halves from tmp into dst
   */
static void *merge_thread(void *vp)
{
    MergeArgs *m = (MergeArgs *)vp;

    if (m->T == 2) {
        int mid = m->n / 2;
        merge_two(m->src, 0, mid, mid, m->n, m->dst, 0); // merge the two halves into dst[0..n)
    } else {//four thread case
        // quarters [0,q), [q,2q), [2q,3q), [3q,n)
        int q = m->n / 4;

        // Stage 1 merge (Q0, Q1) → tmp[0..2q)
        merge_two(m->src, 0,     q,   q,   2*q, m->tmp, 0);

        // Stage 1: merge (Q2, Q3) → tmp[2q..n)
        merge_two(m->src, 2*q, 3*q, 3*q,   m->n, m->tmp, 2*q);

        // Stage 2: merge the two halves from tmp → dst[0..n)
        merge_two(m->tmp, 0,   2*q, 2*q,   m->n, m->dst, 0);
    }

    return NULL;
}

/*
   function for printing
    */
```

```
static void print_array(const char *label, const int *a, int n)
{
    printf("%s", label);
    for (int i = 0; i < n; ++i)
        printf("%s%d", (i ? " " : ""), a[i]);
    printf("\n");
}


int main(int argc, char **argv)
{
    // Check for exactly one command-line argument if not give error messgae
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <even_count> <threads:2|4>\n", argv[0]);
        return 1;
    }

    g_n = atoi(argv[1]);   // number of elements
    g_T = atoi(argv[2]);   // number of sorting threads (2 or 4)

    // N must be positive and even if not give error message threads must be 2 or 4 and N divisible by
threads
    if (g_n <= 0 || (g_n % 2) != 0) {
        fprintf(stderr, "Error: <even_count> must be a positive even number.\n");
        return 1;
    }
    if (!(g_T == 2 || g_T == 4)) {
        fprintf(stderr, "Error: <threads> must be 2 or 4.\n");
        return 1;
    }
    if (g_n % g_T != 0) {
        fprintf(stderr, "Error: N must be divisible by <threads> (e.g., N%%%d==0).\n", g_T);
        return 1;
    }

    // Allocating the two global arrays dynamically
    g_input  = (int *)malloc((size_t)g_n * sizeof(int));
    g_output = (int *)malloc((size_t)g_n * sizeof(int));
    if (!g_input || !g_output) {
        fprintf(stderr, "Error: malloc failed.\n");
        free(g_input);
        free(g_output);
        return 1;
    }

    // input with pseudorandom integers in [0, 1000)
    for (int i = 0; i < g_n; ++i) g_input[i] = rand() % 1000;
```

```
// Show unsorted list
print_array("Original: ", g_input, g_n);

//  g_T sorting threads over equal-size chunks
int chunk = g_n / g_T;  // size of each chunk
pthread_t *tids = (pthread_t *)malloc((size_t)g_T * sizeof(pthread_t));
SortArgs  *sarg = (SortArgs  *)malloc((size_t)g_T * sizeof(SortArgs));
if (!tids || !sarg) {
    fprintf(stderr, "Error: malloc failed.\n");
    free(tids); free(sarg); free(g_input); free(g_output);
    return 1;
}

// Creating each sorting thread with its own slice
for (int t = 0; t < g_T; ++t) {
    sarg[t].arr   = g_input;
    sarg[t].start = t * chunk;
    sarg[t].end   = (t + 1) * chunk;
    if (pthread_create(&tids[t], NULL, sort_thread, &sarg[t]) != 0) {
        fprintf(stderr, "Error: pthread_create (sort %d) failed.\n", t);
        free(tids); free(sarg); free(g_input); free(g_output);
        return 1;
    }
}

// Wait for all sorting threads to finish
for (int t = 0; t < g_T; ++t)
    pthread_join(tids[t], NULL);

// merge thread to combine the sorted chunks
pthread_t t_merge;
int *tmp = NULL;      // temp buffer need only for 4-way merge
if (g_T == 4) {
    tmp = (int *)malloc((size_t)g_n * sizeof(int));
    if (!tmp) {
        fprintf(stderr, "Error: malloc failed (tmp).\n");
        free(tids); free(sarg); free(g_input); free(g_output);
        return 1;
    }
}

MergeArgs margs;
margs.src = g_input;  // read from g_input now contains T sorted chunks
margs.dst = g_output; // output fully merged result
margs.n   = g_n;      // total length
margs.T   = g_T;      // 2 or 4 chunks
margs.tmp = tmp;      // scratch buffer only used when T==4
```

```
    if (pthread_create(&t_merge, NULL, merge_thread, &margs) != 0) {
        fprintf(stderr, "Error: pthread_create (merge) failed.\n");
        free(tmp); free(tids); free(sarg); free(g_input); free(g_output);
        return 1;
    }

    // Wait for merge to complete
    pthread_join(t_merge, NULL);

    // Print sorted array
    print_array("Sorted:   ", g_output, g_n);


    return 0;
}
```

Output:



```
tiger@tucis-ubuntu:~/myDir/lab2$ vi lab2_task2.c
tiger@tucis-ubuntu:~/myDir/lab2$ gcc -pthread lab2_task2.c -o lab2_task2
tiger@tucis-ubuntu:~/myDir/lab2$ ./lab2_task2 20 4
Original: 383 886 777 915 793 335 386 492 649 421 362 27 690 59 763 926 540 426 172 736
Sorted:   27 59 172 335 362 383 386 421 426 492 540 649 690 736 763 777 793 886 915 926
tiger@tucis-ubuntu:~/myDir/lab2$ ./lab2_task2 20 2
Original: 383 886 777 915 793 335 386 492 649 421 362 27 690 59 763 926 540 426 172 736
Sorted:   27 59 172 335 362 383 386 421 426 492 540 649 690 736 763 777 793 886 915 926
tiger@tucis-ubuntu:~/myDir/lab2$ ./lab2_task2 20 3
Error: <threads> must be 2 or 4.
tiger@tucis-ubuntu:~/myDir/lab2$ ./lab2_task2 30 2
Original: 383 886 777 915 793 335 386 492 649 421 362 27 690 59 763 926 540 426 172 736 211 368 567 429 782 530 862 123 67 135
Sorted:   27 59 67 123 135 172 211 335 362 368 383 386 421 426 429 492 530 540 567 649 690 736 763 777 782 793 862 886 915 926
tiger@tucis-ubuntu:~/myDir/lab2$ ./lab2_task2 15 2
Error: <even_count> must be a positive even number.
tiger@tucis-ubuntu:~/myDir/lab2$ ./lab2_task2 30 4
Error: N must be divisible by <threads> (e.g., N%4==0).
tiger@tucis-ubuntu:~/myDir/lab2$ 
```