```c
//=======================================================================
// THREADS - examples
// t01.c
// A program that launches 2 threads and waits for them to end
// Illustrating thread execution interleaving
//-----------------------------------------------------------------------

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

#define NUM_CHARS 10000

void *thr_func(void *arg)
{
  int i;

  fprintf(stderr, "Starting thread %s\n", (char *) arg);
  for (i = 0; i < NUM_CHARS; i++)
    write(STDOUT_FILENO,(char *) arg,1);
  return NULL;
}

int main(void)
{
  pthread_t tid1, tid2;

  printf("Hello from main thread\n");
  pthread_create(&tid1, NULL, thr_func, "A");
  pthread_create(&tid2, NULL, thr_func, "B");
  pthread_join(tid1,NULL);
  pthread_join(tid2,NULL);
  return 0;
}

//=======================================================================
// THREADS - examples
// t02.c
// What may happen if the main thread is the first one to end ... :-(
//-----------------------------------------------------------------------

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

void *thr_func(void *arg)
{
  sleep(3);
  printf("Hello from auxiliar thread\n");
  return NULL;
}

int main(void)
{
  pthread_t tid;

  printf("Hello from main thread\n");
  pthread_create(&tid, NULL, thr_func, NULL);
  return 0;
}
```

```
//======================================================================
// THREADS - examples
// t03.c
// - A child thread can continue running after the main thread end !!!
// - Passing info between threads using global variables
//----------------------------------------------------------------------

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

int global;

void *thr_func(void *arg)
{
  printf("Aux thread: %d\n", global);
  return NULL;
}

int main(void)
{
  pthread_t tid;
  global = 20;

  printf("Main thread: %d\n", global);
  pthread_create(&tid, NULL, thr_func, NULL);
  pthread_exit(NULL);
}

//======================================================================
// THREADS - examples
// t04.c
// - Passing info bidirectionally, using global variables
// - Waiting for the end of a thread (alternative: use sync. mechan.)
//----------------------------------------------------------------------

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

int global;

void *thr_func(void *arg)
{
  global = 20;
  printf("Aux thread: %d\n", global);
  return NULL;
}

int main(void)
{
  pthread_t tid;

  global = 10;
  printf("Main thread: %d\n", global);
  pthread_create(&tid, NULL, thr_func, NULL);
  pthread_join(tid, NULL);
  printf("Main thread: %d\n", global);
  return 0;
}
```

```c
//=======================================================================
// THREADS - examples
// t05.c
// Passing info through thread arguments and return values
//-----------------------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void *thr_func(void *arg)
{
  void *ret;
  int value;

  value = *(int *) arg;
  printf("Aux thread: %d\n",value);
  value++;
  ret = malloc(sizeof(int));
  *(int *)ret = value;
  return ret;
}

int main(void)
{
  pthread_t tid;
  int k = 10;
  void *r;

  pthread_create(&tid, NULL, thr_func, &k);
  pthread_join(tid, &r);
  printf("Main thread: %d\n", *(int *)r);
  free(r);
  return 0;
}
//=======================================================================
// THREADS - examples
// t06.c
// Passing arguments to threads - BE CAREFUL !!!
//-----------------------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define NUM_THREADS 10

void *printHello(void *threadId)
{
  printf("Thread %2d: Hello World!\n", *(int*)threadId);
  pthread_exit(NULL);
}

int main()
{
  pthread_t tid[NUM_THREADS];
  int rc, t;
  for(t=1; t<= NUM_THREADS; t++){
    printf("Creating thread %d\n", t);
    rc = pthread_create(&tid[t-1], NULL, printHello, &t);
    if (rc)
    {
      printf("ERROR; return code from pthread_create() is %d\n", rc);
      exit(1);
    }
  }
  pthread_exit(NULL);
}
```

```c
//========================================================================
// THREADS - examples
// t07.c
// Passing arguments to threads
// One solution the the "passing arguments to the threads" problem
// (only possible in some situations ... when?)
//------------------------------------------------------------------------

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

#define NUM_THREADS 10

void *printHello(void *threadId)
{
  printf("Thread %2d: Hello World!\n", (int)threadId);
  pthread_exit(NULL);
}

int main()
{
  pthread_t tid[NUM_THREADS];
  int rc, t;
  for(t=1; t<= NUM_THREADS; t++){
    printf("Creating thread %d\n", t);
    rc = pthread_create(&tid[t-1], NULL, printHello, (void *)t);
    if (rc)
    {
      printf("ERROR; return code from pthread_create() is %d\n", rc);
      exit(1);
    }
  }
  pthread_exit(NULL);
}
```

```c
//========================================================================
// THREADS - examples
// t08.c
// Passing arguments to threads
// Another solution (?) - see execution example after the code
// to the "passing arguments to the threads" problem
//------------------------------------------------------------------------

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

#define NUM_THREADS 10

void *printHello(void *threadId)
{
  printf("Thread %2d: Hello World!\n", *(int *) threadId);
  pthread_exit(NULL);
}

int main()
{
  pthread_t tid[NUM_THREADS];
  int rc, t;
  int thrArg[NUM_THREADS];

  for(t=1; t<= NUM_THREADS; t++){
    printf("Creating thread %d\n", t);
    thrArg[t-1] = t;
    rc = pthread_create(&tid[t-1], NULL, printHello, &thrArg[t-1]);
    if (rc)
    {
      printf("ERROR; return code from pthread_create() is %d\n", rc);
      exit(1);
    }
  }
  pthread_exit(NULL);
}
```

```c
//========================================================================
// THREADS - examples
// t09.c
// Passing arguments to threads
// The solution to the "passing arguments to the threads" problem:
// allocate space for the arguments in the heap
//------------------------------------------------------------------------

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

#define NUM_THREADS 10

void *printHello(void *threadId)
{
  printf("Thread %2d: Hello World!\n", *(int *) threadId);
  free(threadId);
  pthread_exit(NULL);
}

int main()
{
  pthread_t tid[NUM_THREADS];
  int rc, t;
  int *thrArg;

  for(t=1; t<= NUM_THREADS; t++){
    printf("Creating thread %d\n", t);
    thrArg = (int *) malloc(sizeof(t));
    *thrArg = t;
    rc = pthread_create(&tid[t-1], NULL, printHello, thrArg);
    if (rc)
    {
      printf("ERROR; return code from pthread_create() is %d\n", rc);
      exit(1);
    }
  }
  pthread_exit(NULL);
}
//TO DO: modify in order to free the memory allocated in the heap
```

```c
//========================================================================
// THREADS - examples
// t10.c
// What is the danger of using the 'global' variable?
//------------------------------------------------------------------------

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

#define NUM_ITER 20

int global = 0;

void *thrFunc(void *arg)
{
  while (global++ < NUM_ITER)
  {
    printf("t%d - %d\n",*(int *)arg,global);
    sleep(1); //  <----- COMMENT AND RE-EXECUTE
  }
  return NULL;
}

int main(void)
{
  pthread_t tid1, tid2;
  int t1=1, t2=2; //thread number

  printf("Hello from main thread\n");
  pthread_create(&tid1, NULL, thrFunc, (void *)&t1);
  pthread_create(&tid2, NULL, thrFunc, (void *)&t2);
  pthread_join(tid1,NULL);
  pthread_join(tid2,NULL);
  return 0;
}
```

```c
//=========================================================================
// THREADS - examples
// t10a.c
// What is the danger of using the 'global' variable?
//-------------------------------------------------------------------------

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

#define NUM_ITER 20

int global=0;

void *thrFunc(void *arg)
{
  while (global++ < NUM_ITER)
  {
    printf("t%d - %d\n",*(int *)arg,global);
    //sleep(1); //  <----- COMMENT AND RE-EXECUTE
  }
  return NULL;
}

int main(void)
{
  pthread_t tid1, tid2;
  int t1=1, t2=2; //thread number

  printf("Hello from main thread\n");
  pthread_create(&tid1, NULL, thrFunc, (void *)&t1);
  pthread_create(&tid2, NULL, thrFunc, (void *)&t2);
  pthread_join(tid1,NULL);
  pthread_join(tid2,NULL);
  return 0;
}
```