```
      }

      return 0;

};




/*-------------------------------------------------*/
/* Node 2 side of use case                         */
/*-------------------------------------------------*/


/* Helper functions for Node 2 - these are not part of MRAPI, and could be
   implemented using various appropriate mechanisms */

/* Function to receive an integer message from Node 1, via some appropriate
mechanism (e.g. MCAPI) */
int receive_from_node1();

/* Function to determine whether an mrapi_rmem_get call succeeded */
int get_successful(mrapi_status_t*);

/* Function to tell Node 1 that processing has completed */
void notify_node1();

/* Function for doing processing on an 'Entity' */
float process(Entity *);


#define BUFFER_SIZE 1024

/* Buffers local to Node 2 used to store results, for double-buffered write-
back */
float result_buffers[2][BUFFER_SIZE];

int node2_remote_memory_use_case_1()
{
      mrapi_status_t status; /* For error checking */

      /* Handles for software cache- and DMA-accessed remote memory */
      mrapi_rmem_hndl_t sw_cache_hndl;
      mrapi_rmem_hndl_t dma_hndl;

      unsigned int start_of_node1_heap;
      unsigned int address_of_next_entity_to_process;

      start_of_node1_heap = receive_from_node1();
      address_of_next_entity_to_process = receive_from_node1();
```