Imperative Programming (BSc, 18) homework programming task

In this task, we implement a “parity array” type: an array of integers where the even numbers are at the beginning of the array and the odd numbers are at the end of the array. For example, let the size of the array be 5. Initially, the array is empty: _ _ _ _ _ After adding 2: 2 _ _ _ _ After adding 7: 2 _ _ _ 7 After adding 1: 2 _ _ 1 7 After adding 9: 2 _ 9 1 7 After the addition of 4: 2 4 9 1 7

Basic task (9 points)

Define a preprocessor symbol called MAX_ARRAY_SIZE, which is used to reference the maximum size of the array (instead of burned-in values). Write a function called InsertToParityArray () that takes an array of signed integers as a parameter and the element to be inserted (a signed integer) as a parameter. Although the first solution accepts signed integers, it is only suitable for storing non-negative integers, because we will represent the empty spaces (_) in the array with a value of -1 (this constraint is also solved at the end of the problem sequence). The function stores in an even_idx and odd_idx variables what is the next free index in the array for an even or odd element to be inserted. For example, if the array is 2 _ _ 1 7, then even_idx = 1 and odd_idx = 2. Even_idx and odd_idx should be local variables of the InsertToParityArray () function that retain their value between function calls (if this cannot be resolved, be global variables that are not initialized in the body of the function). The return value of the function is 0 for successful insertion and 1 for unsuccessful insertion. Structure of the InsertToParityArray (array, item) function: [structogram] Create a function called PrintParityArray () that prints the elements of the array to the screen. Empty spaces are represented by a value of -1, array elements of -1 should not be displayed (in the case of an empty space, nothing should appear on the screen). Create a main program in which you define an array, each element of which is initially initialized to -1. Add even and odd numbers to the array with the InsertToParityArray () function, and then use the PrintParityArray () function to print the contents of the array to the screen at least twice: once when it is full and once when there are empty spaces in the array. It is not necessary to read about standard input here, it is enough to call functions with "" burned-in examples.

Modularization (3 points)

Expand your program into translation units. Function implementations should be placed in a separate translation unit for which a header file should be created. Protect the header file with include guard.

ParityArray type (9 points)

Introduce your own ParityArray type and have the functions continue to work by receiving these types of parameters. ParityArray is a structure that has an array of integers MAX_ARRAY_SIZE and two fields, even_idx and odd_idx for the even or odd number, the next free index in the array. ParityArray should also be a type name, so ParityArray pa; count as a valid variable declaration. Modify the InsertToParityArray () function: It receives data of type ParityArray, and the local variables even_idx and odd_idx of the InsertToParityArray () function are no longer needed because this data is stored in fields with the same name in the structure instance. Create a function called InitParityArray () that initializes the ParityArray instance obtained as a parameter: sets all elements of the array to an empty space (-1) and initializes the even_idx and odd_idx fields with the appropriate initial value. Similarly, make the necessary changes to the PrintParityArray () function, but take the ParityArray instance as a parameter to avoid copying it, because potentially a ParityArray can be very large. Also modify the main program to match the changed function parameterization.

Dynamic memory management (9 points)

Drop the MAX_ARRAY_SIZE constraint from the program, instead storing it in the ParityArray type in an unsigned whole field to determine the maximum number of items that the instance can store. Set this field to a new size obtained by the InitParityArray () function as a function parameter. Instead of an array-type structure field, functions store data on the heap; in the type, a pointer is stored instead of an array. Perform a dynamic memory allocation with the InitParityArray () function, which indicates a return value of 1 if the memory reservation was unsuccessful and a return value of 0 if it was successful. In this solution, we already need to support the storage of any signed integer. To do this, delete the part of the InsertToParityArray () function that caused the function to terminate in the case of a negative element. From the InitParityArray () function, deselect all items in the memory area to -1. Therefore, PrintParityArray () can no longer rely on a blank value of -1 to detect blanks; change the PrintParityArray () function accordingly so that you see the same result as before (does not print anything on the screen for blank spaces). Create a function called DisposeParityArray () that resets the ParityArray instance received as a parameter: sets the fields to 0, freeing up the dynamic memory space it allocates.