Deadline
Mon May 17 17:00:00 GMT-8 2004

Learning Keywords
structure, user-defined types, modules, static functions, creating library, void *, type casting, memory management, function pointer, macros, variable argument list.

Your Task
In this assignment, you are required to implement a C programming library that provides a vector data structure similar to the Vector class in Java.

As you may recall, the Vector class in Java implements a growable array of items. Items inside a Vector can be accessed directly using an integer index, just like an array in C. However, the size of a Vector can grow or shrink when necessary, to accommodate insertion and deletion of items. Furthermore, the Vector class in Java stores items of type Object. Thus, any objects of non-primitive type can be stored in a Vector.

Your task is to implement a similar data structure in C called vector through a static library libvector.a. Your library should provide a set of functions to manipulate the vector type and a header file called vector.h that supply the declarations of the types and functions. By providing both libvector.a and vector.h, you allow others to use your library by #include "vector.h" in their C code and by -lvector when compiling with gcc.

Besides type vector, your library should also provide a new type called vector_item, which is the type of the items stored in vector. To allow for arbitrary type to be contained in your vector, vector_item should be equivalent to void *. Since void * is 4 bytes on 32-bit architecture, it can be used to store any data less than 4 bytes directly (with proper casting). For data that are larger than 4 bytes (say, a structure or long long) the vector can store a pointer to the data instead. You can assume that the elements stored in vectors are not-NULL (non-zero).

The type vector should internally maintain two numbers, size, which is the largest index of any existing element in the vector, plus 1, and capacity, which is the maximum number of elements your vector can hold. At any one time, capacity must be at least as large as size. Your vector should expand when it ran out of space. For instance, if capacity is 10, and the user requests an element to be inserted at position 30, the vector should grow so that capacity is larger than 30. A good way to expand is to keep doubling its capacity until there is enough space.

The set of functions that your library should provide are given in the next section.
API to vector type

vector *vector_new(int capacity)
Create and return a new vector with the given capacity. Return NULL if errors.

void vector_add_element_at(vector *v, int index, vector_item element)
Add element into vector v at position index. Shifts the element currently at position index (if any) and any subsequent elements to the right (adds one to their indices).

void vector_add_element(vector *v, vector_item element)
Append element to the end of vector v.

void vector_add_elements(vector *v, vector_item element, ...)
This is the variable argument list version of vector_add_element(). All elements passed in will be appended to the end of vector v in the order they are passed in. The argument list should be terminated with NULL.

int vector_capacity(vector *v)
Return the current capacity of the vector v.

void vector_clear(vector *v)
Remove all elements from vector v. The vector will become empty after this function call returns.

vector *vector_clone(vector *v)
Return a clone of vector v, which contains a shallow copy of all elements in v.

int vector_contains(vector *v, vector_item element)
Return 1 if vector v contains element, return 0 otherwise.

vector_item vector_element_at(vector *v, int index)
Return the element stored at position index in vector v.

void vector_free(vector **v)
Deallocate the vector v. v should points to NULL after this function returns.

int vector_index_of(vector *v, vector_item element)
Return the index of element in v. If more than one occurrences of element appears in v, return the smallest index of such occurrences. If element does not exist in v, return -1.

int vector_is_empty(vector *v)
Return 1 if the vector v is empty, return 0 otherwise.

vector *vector_map(vector *v, vector_item (*f)(vector_item))
Return a new vector w where each element in w is a result of applying function f on the corresponding element in v. For example, if vector v contains strings "luke","leia","han" and f takes in a string and returns its length, then vector w contains the values 4, 4, 3. For those who are familiar with scheme, this is similar to the map function in scheme.
void vector_remove_element(vector *v, vector_item element)
    Remove the first occurrence of element from vector v. If the element is found, shift all
    component in v that has a larger index than element to the left (reduce their indices
    by 1).

vector_item vector_remove_element_at(vector *v, int index)
    Remove the element at position index from vector v and return the element removed.
    Each component in v with an index larger than index is shifted to the left (subtract
    one to their indices). The size of v is decreased by one. If index is not valid, return
    NULL.

int vector_size(vector *v)
    Return the current size of the vector v.

void vector_trim_to_size(vector *v)
    Shrink the vector capacity to be equal to the size of the vector.

Example Usage of vector type

#include "vector.h"
int main()
{
    vector *v = vector_new();
    vector_add_element(v, (vector_item)"one");
    vector_add_element_at(v, 3, (vector_item)"four");
    if (vector_contains(v, (vector_item)"4"))
        vector_remove_element(v, (vector_item)"4");
}

You can download a simple test program from CS2281 website to test your vector pro-
gram. Feel free to change it to test your library.

Submission Requirement

You are required to submit the encrypted versions of four files, (i) vector.c, which imple-
ments the functions, (ii) vector.h, which provides functions declarations and type defini-
tions, and (iii) a Makefile, which compiles vector.c into libvector.a and links with a
test program main.c, (iv) main.c, which contains your test program.

Make sure you have read the submission instruction document posted on CS2281 website.
For this assignment, create a subdirectory under $HOME/CS2281_LABs/ called a3 and put your
encrypted files under the subdirectory. You must include your name as a comment in the
first line of your files. I will access your submission through the following pathname:

- $HOME/CS2281_LABs/a3/vector.c.pgp,
- $HOME/CS2281_LABs/a3/vector.h.pgp,
• $HOME/CS2281_LABs/a3/main.c.pgp, and
• $HOME/CS2281_LABs/a3/Makefile.pgp.

It is your responsibility to make sure that the filenames are correct and permissions are set properly according to the instructions given.

**Additional Tips**

• You should make sure that your library is modularly designed. A user of your library should not be able to call a function in `vector.c` that is not part of the public API.

• When implementing a function from the API, you should try and reuse other vector functions that you have written as much as possible.

• You should try and make use of the `assert` macros where appropriate to check for invariants in your programs.

• You may find the following memory manipulation functions useful: `memcpy`, `memset`, `calloc` and `realloc`. Check their man page to learn their usage.