CSCI 410: Modeling and Simulation

Programming Assignment 1

Due September 30th, 23:59:59PM.

Write the following library functions in C language for list management tasks. All lists should be double-linked lists and all list elements must be stored in a pre-allocated memory block, as we discussed in class on September 8th. You may define your own variables to help the managements. Define a macro of integer value 128 and use it in your program for element data size.

1. Create a list repository

   Function prototype:
   *void * CreateListRepo (unsigned int max_element_num);*
   Parameter:
   max_element_num: Maximum number of elements to be stored in the repository
   Return:
   The pointer as void type to the repository. If memory allocation failed, return NULL.
   Call example:

```
void * repo = CreateListRepo (100);
if (repo == NULL) exit(1);
```

2. Destroy a list repository

   Function prototype:
   *int DestroyListRepo (void * repo);*
   Parameter:
   repo: The pointer of void type to the repository.
   Return:
   Return 1 for success, 0 for failure.
   Call example:

```
if (! DestroyListRepo (repo)) {
  printf ("Failed to delete list repository.\n");
}
```

3. Create a list in the repository

   Function prototype:
   *int CreateList (void * repo, int *tail);*
   Parameter:
   repo: The pointer of void type to the repository.
   tail: Value not used for calling. Upon return, it should point to the tail index of the new list.

Return:

Head index of the new list. If there is no space in the repository, return -1.

Call example:

```
int tail;
int head = CreateList (repo, &tail);
/*
  At this point tail has the index of the list tail element.
 */
if (head < 0) {
  printf ("Failed to create a list.\n");
}
```

4. Delete a list in the repository

Function prototype:

*int DeleteList (void * repo, int head);*

Parameter:

repo: The pointer of void type to the repository.

head: Head index of the list to be deleted.

Return:

Return 1 for success, 0 for failure.

Call example:

```
if (! DeleteList (repo, head)) {
  printf ("Failed to delete list.\n");
}
```

5. Insert an element to the list

Function prototype:

*int InsertElement (void *repo, int * head, int * tail, void * element_ptr, int n);*

Parameter:

repo: The pointer of void type to the repository.

head: Pointer of the head index of the list.

tail: Pointer of the tail index of the list.

element_ptr: Pointer of void type to the element that needs to be inserted into the list

n: If n < 0, insert the element as the (-n)-th last one in the list; if option ⩾ 0, insert the element as the (n + 1)-th element in the list.

Return:

Return 1 for success, 0 for failure.

If necessary, the head and tail index of the list should be updated through the 'head' and 'tail' parameter.

Call example:

```
char element[128];
```

```
if (! InsertElement (repo, &head, &tail, (void *) element, 0))
{
  printf ("Failed to insert element to the beginning.\n");
}
else {
  /* At this point 'head' should have been updated. */
}
if (! InsertElement (repo, &head, &tail, (void *) element, -
1)) {
  printf ("Failed to add element to the end.\n");
}
else {
  /* At this point 'tail' should have been updated. */
}
```

6. Retrieve an element from the list

Function prototype:

*void * RetrieveElement (void * repo, int head, int tail, int n);*

Parameter:

repo: The pointer of void type to the repository.

head: Head index of the list.

tail: Tail index of the list.

n: If n < 0, retrieve the (-n)-th last element in the list; if n ⩾ 0, retrieve the (n+1)-th element.

Return:

Return the pointer of void type to the specified element in the list. If the element does not exist, return NULL.

Call example:

```
char element[128];
/* Retrieve the first element */
element_ptr = RetrieveElement (repo, head, tail, 0);
if (element_ptr != NULL) {
  /* Copy the element data to keep it */
  memcpy ((void *) element, element_ptr, 128);
}
```

7. Delete an element from the list

Function prototype:

*int DeleteElement (void * repo, int * head, int * tail, int n);*

Parameter:

repo: The pointer of void type to the repository.

head: Pointer of the head index of the list.

tail: Pointer of the tail index of the list.

n: If n is < 0, delete the (-n)-th last element in the list; if n ⩾ 0, remove the (n+1)-th element.

Return:

Return 1 for success, 0 for failure.

If necessary, the head and tail index of the list should be updated through the 'head' and 'tail' parameter.

Call example:

```
if (! DeleteElement (repo, &head, &tail, 0)) {
  printf ("Failed to delete the first element.\n");
}
else {
  /* At this point 'head' should have been updated. */
}
if (! DeleteElement (repo, &head, &tail, -1)) {
  printf ("Failed to delete the last element.\n");
}
else {
  /* At this point 'tail' should have been updated. */
}
```

Extra credit work:

Develop test programs and earn no less than 30% of extra credit. If multiple students volunteer, the extra credit will be shared. Please contact the instructor for details.

Files to submit:

- A single file of C code that implements the functions, and
- A header file for other programmers to include in their program in order to use the library functions.

Grading:

- Grading will be done by test programs. The grade will be (Number of passed tests) / (Total number of tests) × 100.