



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 125 - Introduction to Computing Science and Programming II

Arrays & Linked-Lists

Arrays

- Collection of items stored at contiguous memory locations
- Idea to store multiple items of the same type together
- Each element can be uniquely identified by its index in the array
- Advantages:
 - Allows random access to elements
 - Ability to represent multiple data items of the same type using a single name

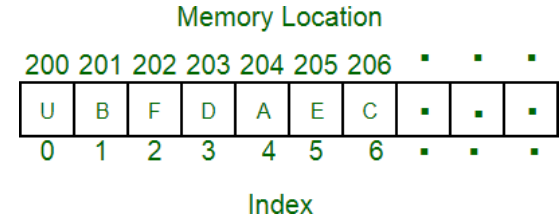


Fig1: Array Structure
Source: [geeksforgeeks](https://www.geeksforgeeks.org/)

Array Operations

- **Get(int index):** Get a value stored at a particular index
 - Time Complexity: $O(1)$
- **Set(int index, int value):** Set a value at a particular index
 - Time Complexity: $O(1)$
- **Append(int value):** Appends value to the end of the array.
 - **Best Time** Complexity: $O(1)$
 - **Worst Time** Complexity: $O(N)$
- **Print():** Print all values of the array
 - Time Complexity: $O(N)$

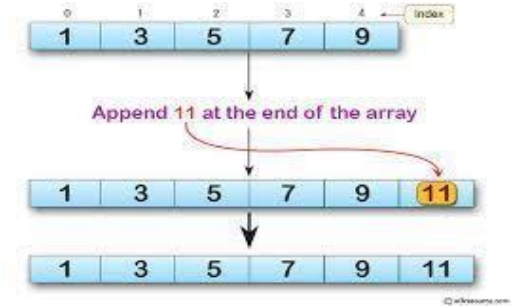


Fig2: Array Append

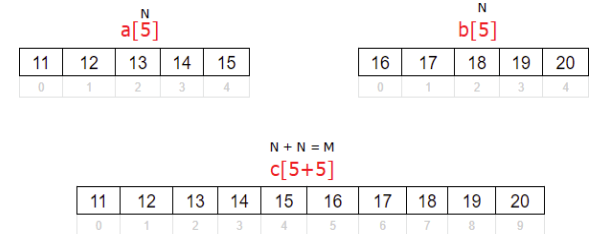


Fig3: Array Concatenation

Array Operations

- 3 files in the zip folder:
 - my_array/my_array.c
 - my_array/my_array.h
 - my_array/test_my_array.c
- "my_array" implemented to help us array like in Python without worrying about resizing
- Functions implemented for operations to get, set, append and print values in an array

Exercise

- Read and understand the functions defined in `my_array.c`
- Implement the function **extend()** in `my_array.c`, which concatenates the values of one array to another array
- Modify the **append()** function, to double the capacity of the array instead of increasing it by 1.

Linked List - Recap

- Chain of separate elements
- Head points to the first element
- Tail points to the last element
- Each element has a data part which contains value and a pointer which points to the next node in the list

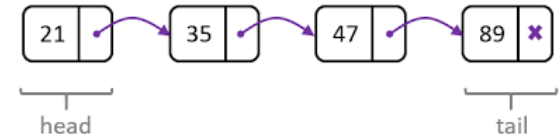


Fig1: Linked List
Source: [101computing](https://www.101computing.com/linked-list/)

Linked List - Operations

- **LL_add_to_head(LL_t *list, int value):** Add element to the head of the list
- **LL_add_to_tail(LL_t *list, int value):** Add element to the tail of the list
- **LL_remove_from_head(LL_t *list):** Remove element from the head of the list
- **LL_size(const LL_t *list):** Return the size of the list
- **LL_print(const LL_t *list):** Prints all elements of the list from head to tail
- **LLnode_free(node_t *node):** Frees memory used by the node
- **LL_free(LL_t *list):** Frees memory used by the list

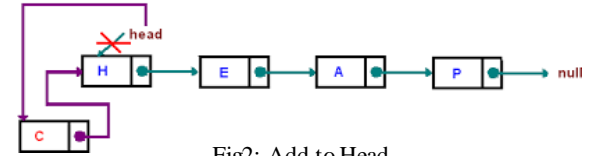


Fig2: Add to Head

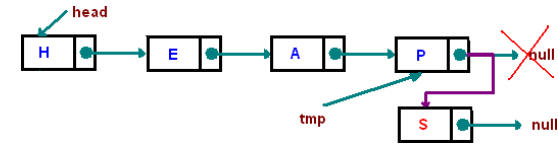


Fig3: Add to Tail

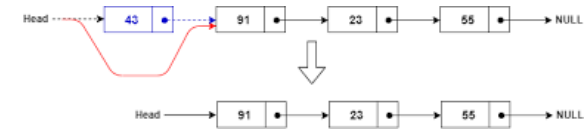


Fig4: Remove from Head

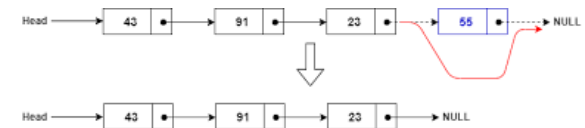


Fig5: Remove from Tail

Steps to compile code

- Unzip and open the directory in VSCode

In the terminal, run:

- `> cd LL`
- `> make`
- `> ./driver_LL`

Exercise

- Read and understand the functions defined in LL.c
- Implement the functions:
 - **LL_remove_from_tail()**: removes element from tail of the list
 - **LL_print_reverse()**: prints list elements in reverse order. Try doing it in $O(N)$ time.
 - **to_array()**: gets a linked list and creates an array with same values
 - **array_to_list()**: gets an array and creates a linked list
 - **are_equal()**: check if two linked lists are equal (equal length and same values in order)
- Add more test cases to test the functions you implement