Data structure : https://medium.com/techie-delight/data-structures-and-algorithms-practice-problems-2810bf3249a4

Bit Manipulation : https://medium.com/techie-delight/bit-manipulation-interview-questions-and-practice-problems-27c0e71412e7

https://medium.com/@codingfreak

Algo : https://www.techiedelight.com/data-structures-and-algorithms-problems/

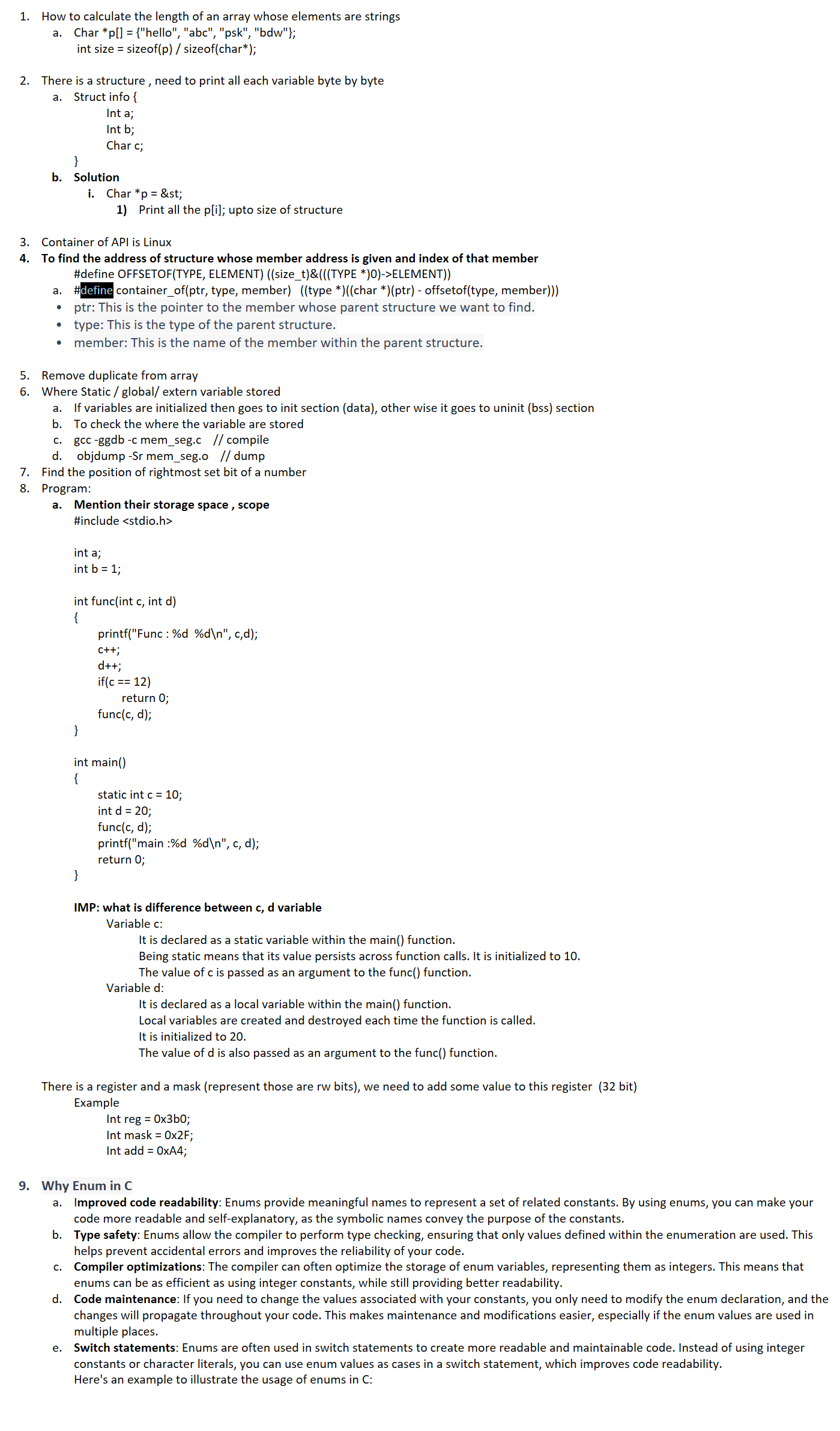
Linked List (leetcode) : https://leetcode.com/tag/linked-list/

Queue (LeetCode) : https://leetcode.com/tag/queue/

String : https://leetcode.com/discuss/study-guide/1333049/Collections-of-string-questions-pattern-for-upcoming-placement-2021

https://leetcode.com/tag/string/

Recursion : https://www.sanfoundry.com/c-programming-examples-recursion/#c-programs-recursion-set-2



1. Round up to the next highest power of 2

a. To find the position of MSB set bit and set the pos bit in another num

2. Round up to the previous highest power of 2

a. To find the position of MSB set bit and set the (pos -1) bit in another num

3. Parity of num // num of set bits in num , even : even parity

4. List Operation

void insert(struct list \*\*head, int data)

{

struct list \*ptr;

ptr = (struct list \*)malloc(sizeof(struct list));

ptr->val = data;

ptr->next = NULL;

if(\*head != NULL)

{

struct list \*temp = \*head;

while(temp->next != NULL) {

temp = temp->next;

}

temp->next = ptr;

}

else

\*head = ptr;

}

5. Implement Stack using linked list

void push(struct list \*\*head, int k)

{

struct list \*ptr;

ptr = (struct list \*)malloc(sizeof(struct list ));

ptr->val = k;

//ptr->next = NULL;

if(\*head == NULL) {

ptr->next = NULL;

\*head = ptr;

}

else {

ptr->next = \*head;

\*head = ptr;

}

}

void pop(struct list \*\*head)

{

struct list \*tmp = \*head;

if(tmp == NULL) {

printf("Stack EMPTY\n");

return;

}

//struct list \*ptr;

printf("Pop: %d\n", tmp->val);

tmp = tmp->next;

free(tmp);

\*head = ptr;

}

6. Implement Queue using linked list

void insert(int k)

{

struct list \*ptr;

ptr = (struct list \*)malloc(sizeof(struct list));

ptr->val = k;

if(front == NULL) {

ptr->next = NULL;

front = ptr;

tail = ptr;

}

else

{

ptr->next= front;

front = ptr;

}

}

void delete()

{

struct list \*ptr = front;

if(front == NULL) {

printf("Queue Underflow\n");

return;

}

else

{

while(ptr->next != tail)

ptr = ptr->next;

free(tail);

ptr->next = NULL;

tail = ptr;

}

}

6. Remove Duplicate from sorted linked list

a. IP : 1, 2,2, 3, 4, 4, 5,5

b. Op : 1, 2, 3, 4, 5

void remove\_duplicate(struct list \*\*head)

{

struct list \*ptr, \*tmp;

ptr = \*head;

tmp = ptr->next;

if(\*head == NULL) {

printf("List Empty\n");

return;

}

while(tmp != NULL)

{

if(tmp->val == ptr->val) {

ptr->next = tmp->next;

free(tmp);

tmp = ptr->next;

}

else {

ptr = ptr->next;

tmp = tmp->next;

}

}

}

7. Detect and Remove the loop from list

void detect\_remove\_loop(struct list \*head)

{

struct list \*slow = head;

struct list \*fast = head;

while(slow && fast) {

slow = slow->next;

fast = fast->next->next;

if(slow == fast)

{

printf("Loop Detected\n");

break;

}

}

// Move slow ptr to head to detect the entry point of loop

slow = head;

// When their is full loop in a list, means head is entry point of loop

if(slow == fast)

{

while(fast->next != slow)

fast = fast->next;

fast->next = NULL;

}

else {

while(fast-> next != slow->next) {

slow = slow->next;

fast = fast->next;

}

fast->next = NULL;

}

}

8. Reverse of list

void reverse\_list(struct list \*\*head)

{

struct list \*prev, \*curr, \*next;

curr = \*head;

prev = NULL;

next = NULL;

while(curr != NULL)

{

// store the next

next = curr->next;

curr->next = prev; // create a link

//move the prev and curr ptr by 1

prev = curr;

curr = next;

}

\*head = prev;

}

9. Delete all even node from the list

void delete\_even\_node(struct list \*\*head)

{

struct list \*tmp, \*ptr;

tmp = \*head;

ptr = tmp->next;

if(tmp == NULL) {

printf("List Empty or Single Node\n");

return;

}

while(ptr && tmp)

{

tmp->next = ptr->next;

free(ptr);

tmp = tmp->next;

if(tmp != NULL)

ptr = tmp->next;

}

}

10. Detect intersect point of list

a. Steps:

i. Calculate the length of both list (len1, len2)

ii. Take a difference of list (d = len1 - len2)

iii. Traverse a longer list to difference, now both list length are equal

iv. Now traverse both the list, list2 from head and list from the current ptr (traverse to difference)

v. Check both ptr are equal, then it’s a intersection point

void detelct\_intersect(struct list \*\*head1, struct list \*\*head2)

{

int d;

int l1 = 0, l2 = 0;

struct list \*ptr, \*ptr2;

ptr = \*head1;

while(ptr != NULL) {

l1++;

ptr = ptr->next;

}

ptr = \*head2;

while(ptr != NULL) {

l2++;

ptr = ptr->next;

}

d = l1 - l2;

// Travesrse the longest loop to till Diff

ptr = \*head1;

while(d)

{

ptr = ptr->next;

d--;

}

ptr2 = \*head2;

while(ptr != NULL)

{

if(ptr == ptr2) {

printf("Intersect Point : %d\n", ptr->val);

break;

}

ptr = ptr->next;

ptr2 = ptr2->next;

}

}

11. Rotate the list by K

1. Calculate the length of list (len)

2. Find the new head

i. Rt = len - k -1 (k : rotate)

3. Traverse the list upto rt location , means list had newhead.

struct ListNode\* rotateRight(struct ListNode\* head, int k){

int cnt = 1;

int rt;

if(head == NULL)

return head;

if(k == 0)

return head;

struct ListNode \*ptr = head;

while(ptr->next != NULL) {

cnt++;

ptr = ptr->next;

}

k = (k % cnt);

if(k == 0)

return head;

rt = cnt - k -1;

ptr->next = head;

struct ListNode \*tmp = head;

struct ListNode \*abc;

for(int i = 0; i < rt; i++)

tmp = tmp->next;

abc = tmp->next;

tmp->next = NULL;

return abc;

12. Delete Nth node from End in List

1. Take 2 ptr (ptr 1 and ptr2)

2. move ptr1 upto k position, means it (len-k+1) position away from end

3. Now move ptr2 upto that position and ptr1 moves to end of list , means ptr2 moves to the kth position from end

struct ListNode\* removeNthFromEnd(struct ListNode\* head, int n){

if(head == NULL || (0 == n))

return head;

struct ListNode \*tmp = head;

struct ListNode \*ptr = head;

for(int i = 0; i < n; i++)

tmp = tmp->next;

if(!tmp) // head node

return head->next;

// traverse the list up to end , tmp : kth position

// ptr --> head, so when tmp reaches to end of list, ptr will pointing to kth position from end

while(tmp->next != NULL) {

tmp = tmp->next;

ptr = ptr->next;

}

ptr->next = ptr->next->next;

return head;

}

13. Swap adjacent pair of node

1. IP : 1->2->3->4-NULL

2. OP : 2->1->4->3->NULL

void swap\_pair\_node(struct list \*\*head)

{

struct list \*tmp = \*head;

struct list \*ptr = tmp->next;

struct list \*abc;

if((\*head == NULL) || (tmp->next == NULL)) // no node or single node

return;

\*head = ptr; // Imp

while(ptr)

{

abc = ptr->next;

ptr->next = tmp;

if((abc == NULL) || (abc->next == NULL)) {

tmp->next = abc;

return;

}

// to maintain the list in a order

tmp->next = abc->next; // trick

tmp = abc;

ptr = tmp->next;

}

}

14. Reverse the node from position n to m

struct list \*reverse\_list\_pos\_n\_m(struct list \*head, int n, int m)

{

struct list \*curr, \*prev;

curr = head;

int i = 0;

for(i = 1; i < n;i++) {

prev = curr;

curr = curr->next;

}

struct list \*new\_head = curr;

struct list \*tail = NULL;

struct list \*next = NULL;

while(i <= m)

{

next = curr->next;

curr->next = tail;

tail = curr;

curr = next;

i++;

}

if(prev != NULL)

prev->next = tail;

else

head = prev;

printf("New\_head: %d\n", new\_head->val);

new\_head->next = curr;

return head;

}

15. Remove all elements of list with value k

Given the head of a linked list and an integer val, remove all the nodes of the linked list that has Node.val == val, and return the new head.

truct ListNode\* removeElements(struct ListNode\* head, int val){

struct ListNode\* tmp = head;

struct ListNode\* ptr = NULL;

if(head == NULL)

return head;

while((head != NULL) && (head->val == val))

head= head->next;

tmp = head;

while(tmp != NULL) {

if(tmp->val == val) {

ptr->next = tmp->next;

tmp = ptr->next;

}

else {

ptr = tmp;

tmp = tmp->next;

}

}

return head;

}

16. Check the Linked list is palindrome

1. Using stack

i. Push all the elements of list in stack (traversing a list to NULL)

ii. Pop all the elements and compare with list (starting) , basically last element comparing with first element

2. Reverse of List

i. Reverse the list

ii. Compare the list element (reverse) with original list

3. Compare 2 half of list

i. Takes 2 pointer to find the mid of the list

ii. In case of even length list , the list is equally divided,

iii. In case of odd , a mid point saves

iv. then reverse of 2nd half and compare the list of elements

struct ListNode\* reverse(struct ListNode \*head)

{

struct ListNode \*curr, \*next, \*prev;

curr = head;

prev = NULL;

while(curr)

{

next = curr->next;

curr->next = prev;

prev = curr;

curr= next;

}

return prev;

}

int compare\_list(struct ListNode\* tmp, struct ListNode\* ptr)

{

while(tmp && ptr) {

if(tmp->val != ptr->val)

return 0;

tmp = tmp->next;

ptr = ptr->next;

}

return 1;

}

bool isPalindrome(struct ListNode\* head){

struct ListNode \*np, \*fp, \*prev, \*sec\_hf\_head, \*mid;

mid = NULL;

if((head == NULL))

return 0;

if(head->next == NULL)

return 1;

np = head;

fp = head;

while(fp && fp->next)

{

prev = np;

np = np->next;

fp = fp->next->next;

}

if(fp != NULL){ // List is odd

mid = np;

np = np->next;

}

sec\_hf\_head = np; // save the head of 2nd half

sec\_hf\_head = reverse(sec\_hf\_head);

prev->next = NULL; // seprate out the list 1 and 2nd half

int k = compare\_list(head, sec\_hf\_head);

if(mid) {

prev->next = mid;

mid->next = sec\_hf\_head;

}

else {

prev->next = sec\_hf\_head;

}

return k;

}

Queue

1. Implement stack using queue

1. Using 1 queue

2. Push()

• The idea behind this approach is to make one queue and push the first element in it.

• After the first element, we push the next element and then push the first element again and finally pop the first element.

• So, according to the FIFO rule of the queue, the second element that was inserted will be at the front and then the first element as it was pushed again later and its first copy was popped out.

• So, this acts as a Stack and we do this at every step i.e. from the initial element to the second last element, and the last element will be the one that we are inserting and since we will be pushing the initial elements after pushing the last element, our last element becomes the first element.

struct queue {

int val;

};

typedef struct {

struct queue \*stack[101];

int front;

int rear;

} MyStack;

MyStack\* myStackCreate() {

MyStack \*st = (MyStack \*)malloc(sizeof(MyStack));

st->front = st->rear = -1;

return st;

}

void enqueue(MyStack \*st, struct queue \*q)

{

if(st->rear == -1)

st->rear = st->front = 0;

else

st->rear++;

st->stack[st->rear] = q;

}

struct queue \*dequeue(MyStack \*st)

{

struct queue \*q;

if(st->front == -1)

return NULL;

q = st->stack[st->front];

if(st->front== st->rear)

st->front = st->rear = -1;

else

st->front++;

return q;

}

void myStackPush(MyStack\* obj, int x) {

struct queue \*q = (struct queue \*)malloc(sizeof(struct queue));

q->val = x;

enqueue(obj, q);

int size = obj->rear - obj->front;

while(size)

{

struct queue \*ptr = dequeue(obj);

enqueue(obj, ptr);

size--;

}

}

int myStackPop(MyStack\* obj) {

struct queue \*ptr = dequeue(obj);

int val = ptr->val;

free(ptr);

return val;

}

int myStackTop(MyStack\* obj) {

struct queue \*ptr = obj->stack[obj->front];

return ptr->val;

}

bool myStackEmpty(MyStack\* obj) {

if(obj->front == -1)

return 1;

return 0;

}

void myStackFree(MyStack\* obj) {

while(!myStackEmpty(obj))

{

struct queue \*ptr = dequeue(obj);

free(ptr);

}

free(obj);

}

/\*\*

\* Your MyStack struct will be instantiated and called as such:

\* MyStack\* obj = myStackCreate();

\* myStackPush(obj, x);

\* int param\_2 = myStackPop(obj);

\* int param\_3 = myStackTop(obj);

\* bool param\_4 = myStackEmpty(obj);

\* myStackFree(obj);

\*/

2. Reverse a list using Recursion

Node \*reverse(Node \*head)

{

If((head== NULL) || (head->next == NULL))

return head;

Node \*rest = reverse(head->next);

head->next->next = head; // to create a link b/w 2->1 , 2 is pointing to 1

Head->next = NULL; // last node 1 ->NULL

return rest;

}

M-2:

LIST \*reverse(LIST \*curr, LIST \*prev)

{

if(curr == NULL)

return prev;

LIST \*next = curr->next;

curr->next = prev;

prev = curr;

return reverse(next, prev);

}

Int main()

{

Struct list \*head;

Insert(10);

Insert(12);

head= reverse(head, NULL);

}