2) ~~Swapping Integer~~ Week-1 - Practice Program:

o/p Current Balance: 1100
Current Balance: 1000
Current Balance: 1000
Sorted Array: Abhinav Prana Prathan pravar
Enter a key to be searched: 2
Element found at index [1][0]
Subset from (0,7)
Enter element to be searched: 8
last occurrence of number is at index 9
key to be searched: 1
key at index 1

Element to be searched: 1
Element found at index 1
Max in list = 10001; Min in list = -5.

Week-2

1) Swapping Integer with pointers (21/12/23)

```
#include <stdio.h>
void swap (int *a, int *b) {
    int temp = *a; *a = *b; *b = temp;
}
int main() {
    int a = 3, b = 4;
    printf("a = %d; b = %d", a, b);
    swap (&a, &b);
    printf("a = %d; b = %d", a, b);
    return 0;
}
```

o/p a = 3; b = 4
a = 4; b = 3

2) Stack implementation (21/12/23)

~~Dynamic Memory Allocation~~:

```
#include <stdio.h>
#include <stdlib.h>
#define size 10
int pos = -1;
int stack[size];

void push(int a);
int pop();
void display();

int main(){
    printf("1. Push\n 2. Pop\n 3. Display stack\n 4. Exit\n
                                    Enter Choice: ");

    int choice;
    int a;
    scanf("%d", &choice);
    while(choice != 4){
        switch(choice){
            case 1:
                printf("Enter integer to be pushed: ");
                scanf("%d", &a);
                push(a);
                break;
            case 2:
                a = pop();
                printf("Integer popped = %d\n", a);
                break;
            case 3:
                display();
                break;
}
```

```
        printf("Enter choice: ");
        scanf("%d", &choice);
    }
}

void push(int a){
    if(pos == a){
        printf("Stack Overflow Condition");
        return;
    }
    stack[++pos] = a;
}

int pop(){
    if(pos == -1){
        printf("Stack Underflow Condition");
        return (int)NULL;
    }
    return stack[pos--];
}

void display(){
    for(int i=0; i<size; i++){
        printf("%d ", stack[i]);
    }
    printf("\n");
}
```

1. Push
2. Pop
3. Display Stack

o/p
1. Push
2. Pop
3. Display Stack
4. Exit
Enter choice: 1
Enter integer to be pushed: 3
Enter choice: 1
Enter integer to be pushed 4
Enter choice: 2
Integer popped = 4
Enter choice: 1
Enter integer to be pushed: 5
Enter choice: 3
3 5 0 0 0 0 0 0 0 0
Enter choice: 4

3) Dynamic Memory Allocation: (21/12/23)

```
#include <stdio.h>
#include <stdlib.h>

int siz = 3;

int main(){
    int* arr = malloc(siz * sizeof(int));
    int* arr2 = calloc(siz, sizeof(int));
    printf("arr and arr2 can store %d integers\n", siz);
    realloc(arr, 2*siz*sizeof(int));
    for(int i=0; i<siz; i++){
        arr2[i] = 1;
    }
    for(int i=0; i<2*siz; i++){
        arr[i] = 0
    }
}
```

```
printf("\n arr: ");
for (int i=0; i<2*size; i++){
    arr[i] = 0 printf("%d ", arr[i]);
}
printf("\n arr2: ");
for (int i=0; i<size; i++){
    printf("%d ", arr2[i]);
}
free(arr); free(arr2);
return 0;
}
```

o/p arr and arr2 can store 3 integers

Can now store 6 integers in arr and 3 integers in arr2

arr: 0 0 0 0 0 0

arr2: 1 1 1

Sol:

$2^1 | 2^2 | 2^3$

Week-3

1) Infix to postfix (28/12/23)

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

#define siz 20

void push(char a);
char pop();
void display();
char* postfix(char* exp);
bool character(char c);
bool lower_precedence(char op1, char op2);
bool isEmpty();

int pos = -1;
char stack[siz];
int n;
int main(){
    printf("Enter size of expression: ");
    scanf("%d", &n);
    fflush(stdin);
    char* exp = (char*) malloc(n(n+1)* sizeof(char));
    printf("Enter infix expression: \n");
    scanf("%[^\n]s", exp);
    char* postfixexp = postfix(exp);
    printf("Postfix expression: %s\n", postfixexp);
}

bool isEmpty(){
    return pos == -1;
}
```

```
void push (char a){
    if (pos == size-1){
        printf("Stack Overflow Condition");
        return;
    }
    stack[++pos] = a;
}
char pop(){
    if (pos == -1){
        printf("Stack Underflow condition");
        return (char) NULL;
    }
    char return_val = stack[pos];
    stack[pos] = (char) NULL;
    pos--;
    return return_val;
}
void display(){
    printf("Stack: ");
    for (int i=0; i<siz; i++){
        printf("%c ", stack[i]);
    }
    printf("\n");
}
bool character(char c){
    return (c>='a' && c<='z') || (c>='A' && c<='Z') ||
           (c>='0' && c<='9');
}
bool lower_precedence(char op1, char op2){
    if (op1 == op2 && op2 == '^') return false;
    char op_order[] = {'^', '/', '*', '+', '-'};
    int o1, o2;
```

```
    for (int i=0; i<(; i++){
        if (op_orda[i] == op1) o1 = i;
        if (op_orda[i] == op2) o2 = i;
    }
    return o1 <= o2
}
char* postfix(char* exp){
    char* return_exp = (char*) malloc ((size+3)* sizeof(char));
    int current = 0;
    for (int i=0; i<n; i++){
        if (character(exp[i])) return_exp[current++] = exp[i];
        else if (exp[i] == '+' || exp[i] == '-' || exp[i] == '*' ||
                 exp[i] == '/' || exp[i] == '^') {
            if (isEmpty) push(exp[i]);
            else if (lower_precedence(stack[pos], exp[i])){
                while (lower_precedence(stack[pos], exp[i]) &&
                       !isEmpty()) {
                    if (stack[pos] != '(') return_exp[current++] = pop();
                    else {
                        pos--;
                        break;
                    }
                }
                push(exp[i]);
            } else
                else push(exp[i]);
        }
        else if (exp[i] == '(') push('(');
        else if (exp[i] == ')') {
            while (stack[pos] != '(' && !isEmpty()){
                return_exp[current++] = pop();
            }
        }
    }
```

```
while (!isEmpty()){
    if (stack[pos] != '(') return exp[current++] = pop();
    else pos--;
}
return return_exp;
}
```

o/p Enter size of expression: 9
Enter infix expression: 1*2+3*4-5
Postfix expression: 12*34*+5-

2) Evaluating postfix expression [28/12/23]

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#define size 20

void push(int a);
int pop();
void display();
bool isEmpty();
bool isOperator(char op);

int stack[size];
int pos = -1;
int n;
int main(){
    printf("Enter size of expression: ");
    scanf("%d", &n);
    char* exp = (char*) malloc(size * sizeof(char));
```

```
Printf("Enter postfix expression: ");
Scanf("%s", exp);
int n1, n2;
for (int i=0; i<n; i++){
    if (exp[i] >='0' && exp[i] <='9') push(exp[i]);
    else if (isOperator(exp[i])){
        n1 = pop(); n2 = pop();
        switch (exp[i]){
            case '+':
                push(n2+n1);
                break;
            case '-':
                push(n2-n1);
                break;
            case '*':
                push(n2*n1)
                break;
            case '/':
                push(n2/n1);
                break;
            case '^':
                push(n1^n2);
                break;
            default:
                printf("Invalid");
                break;
        }
    }
}
printf("Final value: %d", pop());
}
```

```
bool isOperator(char op){
    char operators[7] = {'^', '/', '*', '+', '-'};
    for(int i=0; i<5; i++){
        if(op == operators[i]) return true;
    }
    return false;
}
bool isEmpty(){
    return pos == -1;
}
void push(int a){
    if(pos == size-1){
        printf("Stack Overflow");
        return;
    }
    stack[++pos] = a;
}
int pop(){
    if(pos == -1){
        printf("Stack Underflow");
        return (int) NULL;
    }
    int return_value = stack[pos];
    stack[pos] = (int) NULL;
    pos--;
    return return_value;
}
void display(){
    printf("Stack: ");
    for(int i=0; i<size; i++){
        printf("%d ", stack[i]);
    }
    printf("\n");
}
```

o/p Enter Size of expression: 9
Enter postfix expression: 12+34*+5-
Final value: 9

5) Queue # (28/12/23)

```
#include <stdio.h>
#include <stdlib.h>

#define size 5

void push(int a);
int pop();
void display();
int fpos=-1, rpos=-1; int queue[size];
int main(){
    int choice;
    printf("1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n
    Enter choice:");
    scanf("%d", &choice);
    int a;
    while (choice != 4) {
        switch (choice){
            case 1:
                printf("Enter integer: ");
                scanf("%d", &a);
                push(a);
                break;
            case 2:
                a = pop();
                printf("Popped integer = %d\n", a);
                break;
            case 3:
                display();
                break;
```

```
        default:
            printf("Idk");
            break;
        }
        printf("Enter choice: ");
        scanf("%d", &choice);
    }
}
void push(int a){
    if (fpos == -1 && apos == -1) {
        queue[++apos] = a;
        fpos++;
        return;
    }
    else if (apos == size-1) {
        printf("Queue overflow condition");
        return;
    }
    else {
        queue[++apos] = a;
        return;
    }
}
int pop(){
    if (fpos == -1) printf("Queue Underflow Condition");
    int n = queue[fpos];
    queue[fpos] = (int) NULL;
    fpos++; return n;
}
void display(){
    printf("Queue: ");
    for (int i=0; i<size; i++)
        printf("%d ", queue[i]);
    printf("\n");
}
```

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter choice: 1
Enter integer: 2
Enter choice: 1
Enter integer: 5
Enter choice: 1
Enter integer: 4
Enter choice: 1
Enter integer: 5
Enter choice: 1
Enter integer: 6
Enter choice: 3
Queue: 2 3 4 5 6
Enter choice: 2
Popped integer = 2
Enter choice: 3
Queue: 0 3 4 5 6
Enter choice: 4

4) Circular Queue (28/12/23)

```
#include <stdio.h>
#include <stdlib.h>

#define size 5

void push (int a);
int pop();
void display();
int fpos=-1, rpos=-1; int queue[size];
```

papergrid

```
int main(){
    int choice;
    printf("1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");
    scanf("%d", &choice);
    int a;
    while (choice != 4){
        switch (choice){
            case 1:
                printf("Enter integer: ");
                scanf("%d", &a);
                push(a);
                break;
            case 2:
                a = pop();
                printf("Popped integer: %d", a); break;
            case 3:
                display(); break;
            default:
                printf("Idle"); break;
        }
        printf("Enter choice: ");
        scanf("%d", &choice);
    }
}
void push(int a){
    if (fpos == -1 && rpos == -1){
        queue[++rpos] = a;
        fpos++;
        return;
    }
    else if ((rpos+1) % size == fpos % size){
        printf("Queue Overflow condition");
        return;
    }
```

```
        else {
            rpos++;
            queue[(rpos % size)] = a;
            return;
        }
}
int pop(){
    if(fpos == rpos){
        printf("Queue Underflow condition");
    }
    int n = queue[fpos % size];
    queue[fpos % size] = (int) NULL;
    fpos++;
    return n;
}
void display(){
    printf("Queue: ");
    for(int i=0; i<n; i++){
        printf("%d ", &queue[i]);
    printf("\n");
}
```

o/p

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter choice: 1
Enter integer: 2
Enter choice: 1
Enter integer: 3
Enter choice: 1
Enter integer: 4
Enter choice: 1
Enter integer: 5
```

Enter choice: 1
Enter integer: 6
Enter choice: 3
Queue: 2 3 4 5 6
Enter choice: 2
Popped integer = 2
Enter choice: 1
Enter integer: 1
Enter choice: 3
Queue: 1 3 4 5 6
Enter choice: 4

Week-4

1) Singly linked list: (11/1/24)

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
typedef struct Node{
    int data;
    struct Node* next;
} node;
node* head = NULL;
int count = 0;

void insert(int data, int position);
void delete(int position);
void display();

int main(){
    int data, choice, pos;
    printf("1. Insert\n2. Delete\n3. Exit\n Choice: ");
    scanf("%d", &choice);
```

```
while (choice != 3){
    if (choice == 1) {
        printf("Enter data and position: ");
        scanf("%d%d", &data, &pos);
        insert(data, pos);
        printf("Count: %d\n", count);
    }
    display();
    printf("Enter choice: ");
    scanf("%d", &choice);
}
return 0;
}

void insert(int data, int pos){
    if (pos==0){
        node* new_node = (node*)malloc(sizeof(node));
        new_node->data = data;
        new_node->next = NULL;
        head = new_node;
        count++; return;
    } else if (position == count){
        node* new_node = malloc(sizeof(node));
        new_node->data = data;
        new_node->next = NULL;
        node* temp = head;
        while (temp->next != NULL) temp = temp->next;
        temp->next = new_node;
        count++; return;
    } else if (position > count || position < 0){
        printf("Unable to insert\n"); return;
    } else {
```

```
node *temp = head;
for (int i=0; i<pos-1; i++)
    temp = temp->next;
node * new_node = malloc (sizeof (node));
new_node -> data = data;
new_node -> next = temp->next;
temp-> next = new_node
count++; return
}
}
void display(){
    node *temp = head;
    printf(printf(" Linked List: ");
    while (temp-> next != NULL){
        printf("%d ", temp->data);
        temp= temp -> next;
    }
    printf("%d ", temp-> data);
    printf("\n");
}
```

Output:

1. Insert
2. Delete
3. Exit

Choice: 1

Enter data and position: 10

Count: 1

Linked List: 1

Choice: 1

Enter data and position: 2 1

Count: 2

Linked List: 1 2

Enter Choice: 1
Enter data and position: 32
Count: 3
Linked List: 1 2 3
Enter Choice: 1
Enter data and position: 41
Count: 4
Linked List: 1 4 3 2
Enter Choice: 1
Enter data and position: 52
Count: 5
Linked List: 1 4 5 3 2
Enter Choice: 3

S.T.
10/24

Week-5 (18/1/24)

1) Single linked List deletion: (18/1/24)

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int data;
    struct Node* next;
} node;

node* head = NULL; int count = 0;

void insert(int data, int position);
void delete(int position);
void display();

int main() {
    int data, choice, pos;
    printf("1. Insert \n 2. Delete \n 3. Exit \n Choice: ");
    scanf("%d", &choice);
    while (choice != 3) {
        if (choice == 1) {
            printf("Enter data and position: ");
            scanf("%d %d", &data, &pos);
            insert(data, pos);
            printf("Count: %d \n", count);
        } else if (choice == 2) {
            printf("Enter position: ");
            scanf("%d", &pos);
            delete(pos);
            printf("Count: %d \n", count);
        }
        display(); printf("Enter choice: "); scanf("%d", &choice);
    }
    return 0;
}
```

```
Void insert (int data, int position) {
    if (position == 0) {
        node *new_node = (node*) malloc (sizeof(node));
        new_node -> data = data;
        new_node -> next = head;
        head = new_node; count++; return;
    } else if (position == count) {
        node * new_node = malloc (sizeof (node));
        new_node -> data = data;
        new_node -> next = NULL;
        while (temp -> next != NULL) temp = temp -> next;
        temp -> next = new_node;
        count++;
        return;
    } else if (position > count || position < 0) {
        printf ("Unable to insert"); return;
    } else {
        node *temp = head;
        for (int i=0; i < position-1; i++) {
            temp = temp -> next;
        node * new_node = malloc (sizeof (node));
        new_node -> data = data;
        new_node -> next = temp -> next;
        temp -> next = new_node;
        count++; return;
    }
}
Void delete (int position) {
    if (position == 0) {
        node *temp = head;
        head = head -> next;
        free (temp); count--; return;
    }
```

```
else if (position == count-1) {
    node *temp = head;
    for (int i=0; i<count-1; i++) {
        temp = temp->next;
    node *temp1 = temp->next;
    temp->next = NULL; free (temp1);
    count--; return;
} else if (position > count || position < 0) {
    printf ("Unable to delete"); return;
} else {
    node *temp = head;
    for (int i=0; i<position-1; i++) {
        temp = temp->next;
    node* temp1 = temp->next;
    temp->next = temp1->next;
    free (temp1); count--; return;
}
}
void display () {
    node *temp = head;
    printf ("Linked List: ");
    while (temp->next != NULL) {
        printf ("%d ", temp->data);
        temp = temp->next;
    }
    printf ("%d ", temp->data); printf ("\n");
}
```

Output:

1. Insert
2. Delete
3. Exit

Choice: 1
Enter data and position: 1 0
Count: 1
Linked List: 1
Enter Choice: 1
Enter data and position: 2 1
Count: 2
Linked List: 1 2
Enter choice: 1
Enter data and position: 3 1
~~Listo~~ Count: 3
Linked List: 1 3 2
Enter choice: 1
Enter choice: 4 3
Count: 4
Linked List: 1 3 2 4
Enter Choice: 2
Enter position: 0
Count: 3
Linked List: 3 2 4
Enter choice: 2
Enter position: 1
Count: 2
Linked List: 3 4
Enter choice: 2
Enter position: 1 → ~~Count: 1~~
Linked List: 3
Enter choice: 3

Sofi
18/1/24

2) Min Stack: (18/1/23)

```
typedef struct {
    int stack[30000];
    int min;
    int top;
} MinStack;
MinStack* minStackCreate() {
    MinStack* obj = malloc(sizeof(MinStack));
    obj->top = -1;
    obj->min = INT_MAX;
    return obj;
}
void minStackPush(MinStack* obj, int val) {
    if (val <= obj->min) {
        obj->stack[++(obj->top)] = obj->min;
        obj->min = val;
    }
    obj->stack[++(obj->top)] = val; return;
}
void minStackPop(MinStack* obj) {
    if (obj->stack[obj->top] == obj->min) {
        obj->stack[obj->top] = NULL;
        obj->top = -1;
        obj->min = obj->stack[obj->top];
    }
    obj->stack[obj->top] = NULL; obj->top = -1;
}
int minStackTop(MinStack* obj) {
    return obj->stack[obj->top];
}
int minStackGetMin(MinStack* obj) {
    return obj->min;
}
```

```
void minStackFree(MinStack *obj) {
    free(obj);
}
```

5) Reverse Linked List: (18/1/23)

```
struct ListNode* reverseBetween(struct ListNode* head,
                                int left, int right) {
    struct ListNode *l = head;
    struct ListNode *r = head;
    int different = right - left;
    if (left == right) return head;
    for (int i=0; i<left-1; i++) {
        l = l->next;
    for (int i=0; i<right-1; i++) {
        r = r->next;
    printf while (different > 0) {
        int temp = l->val;
        l->val = r->val;
        r->val = temp;
        l = l->next;
        r = l;
        for (int i=0; i<different-1; i++) {
            r = r->next;
        }
        different -= 2;
    }
    return head;
}
```

Sol. 1
18/1/23

Week-6 (25/1/24)

1) Sorting, Reverse and Concat:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int data; struct Node* next;
} node;

node *head = NULL;
node *head1 = NULL;
int count = 0;

void insert(int data, int position);
void display();
void sort();
void reverse();
void concat(node **head1, node *head2);

int main() {
    insert(2,0); insert(1,1); insert(4,2); insert(3,3); insert(5,4);
    printf("Original Linked List: \n");
    display(); sort(); printf("Sorted Linked List: \n"); display();
    reverse(); printf("Reversed Linked List: \n"); display();
    head1 = head; head = NULL;
    insert(3,0); insert(4,1); insert(1,2); display();
    concat(&head1, &head); head = head1;
    printf("Concatenating with above linked list gives: \n");
    display(); return 0;
}

void insert(int data, int position) {
    if (position >= 0) {
```

```
node* new_node = (node*) malloc(sizeof(node));
new_node->data = data;
new_node->next = head; head = new_node;
~~head~~ count++; return;
} else if (position == count){
node *newnode = malloc(sizeof(node));
new_node->data = data;
new_node->next = NULL;
node *temp = head;
while (temp->next != NULL) temp = temp->next;
temp->next = new_node; count++; return;
} ~~else~~ {
~~node *temp;~~
}
void sort(){
int i, j, min_index;
~~no~~node *i_node = head, *j_node = head, *min_node = NULL;
~~for~~ (int i=0; i<count-1; ~~i~~i++, i_node = i_node->next){
min_index = i;
min_node = i_node;
j_node = i_node->next;
for (int j = i+1; j<count; j++, j_node = j_node->next){
if (j_node->data < i_node->data){
min_index = j;
min_node = j_node;
}
}
if (min_index != i){
int temp = i_node->data;
~~int temp~~ = i_node->data = min_node->data;
min_node->data = temp;
}
}
}
```

```
void reverse(){
    node *prev = NULL, *next = NULL;
    while (head != NULL){
        next = head->next; head->next = prev;
        prev = head; head = next;
    }
    head = prev;
}
void concat(node *head1, node *head2){
    node *temp1 = head1;
    while (temp1->next != NULL){
        temp1 = temp1->next;
    }
    temp1->next = head2;
}
void display(){
    node *temp = head;
    printf("Linked List: ");
    while (temp != NULL){
        printf("%d ", temp->data);
    }
    printf("\n");
}
```

Output:

Original Linked List:
Linked List: 2 1 4 3
Sorted Linked List:
Linked List: 1 2 3 4
Reversed Linked List:
Linked List: 5 4 3 2 1
Linked List: 3 4 1
Concatenating with above list give:
Linked List: 5 4 3 2 1 3 4 1

2) Stack with LL: (25/1/24)

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{
    int data; struct Node *next;
} node;

node *head = NULL;
int count = 0;

void insert (int data);
int data ();
void display();

int main(){
    int data, choice;
    printf(" 1. Push \n 2. Pop \n 3. Exit \n Choice: ");
    scanf("%d", &choice);
    while (choice != 3){
        if (choice == 1){
            printf(" Enter data: ");
            scanf("%d", &data);
            insert (data);
        } else if (choice == 2){
            printf(" Integer popped: %d\n", delete());
        } printf(
        display();
        printf(" Enter choice: ");
        scanf("%d", &choice);
    }
    return 0;
}
```

```
void insert(int data){
    node *new_node = (node*)malloc(sizeof(node));
    new_node->data = data;
    new_node->next = head; head = new_node;
    count++; return;
}
int delete(){
    node *temp = head;
    head = head->next;
    int t = temp->data;
    free(temp);
    count--;
    return t;
}
void display(){
    node *temp = head;
    printf("Stack: ");
    while(temp != NULL){
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
```

Output:

1. Push
2. Pop
3. Exit

Choice: 1
Enter data: 3
Linked List: 3
Choice: 1
Enter data: 2
Linked List: 2

Enter Choice: 1
Enter data: 5
Linked list: 5
Enter Choice: 2
Integer popped: 5
Linked list: 2 3
Enter Choice: 3

5) Queue with LL: (2 | 1 | 24)

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Node{
        int data; struct Node *next;
} node;
node *head = NULL;
int count = 0;

void insert(int data);
int delete();
void display();

int main(){
    int data, choice;
    Printf("1. Push\n2. Pop\n3. Exit\nChoice: ");
    Scanf("%d", &choice);
    while(choice != 3){
        if(choice == 1){
            Printf("Enter data: ");
            Scanf("%d", &data);
            insert(data);
        } else if Printf("count (choice == 2){
```

```
        printf("Data popped : %d", delete());
    }
    display();
    printf("Enter choice: ");
    scanf("%d", &choice);
}

void insert(int data){
    node *new_node = malloc(sizeof(node));
    new_node->data = data;
    new_node->next = NULL;
    if(head == NULL){
        head = new_node; count++; return;
    }
    node *temp = head;
    while(temp->next != NULL){
        temp = temp->next;
    }
    temp->next = new_node; count++; return;
}

int delete(){
    node *temp = head;
    head = head->next;
    int t = temp->data;
    free(temp); count--; return t;
}

void display(){
    node *temp = head
    while(temp != NULL){
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
```

Output:

1. Insert
2. Delete
3. Exit

Choice: 1

Enter data: 1

Count: 1

Queue: 1

Enter choice: 1

Enter data: 2

Count: 2

Queue: 1 2

Enter choice: 1

Enter data: 3

Count: 3

Queue: 1 2 3

Enter choice: 2

Integer popped: 1

Count: 2

Queue: 2 3

Enter choice: 2

Integer popped: 2

Count: 1

Queue: 3

Enter choice: 3

papergrid

Week 7

1) Doubly linked list (1/2/24)

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int data; struct Node *next; struct Node *prev;
} node;

node *head = NULL;
int count = 0;

void insert(int data, int position);
void delete(int element);
void display();

int main() {
    int data, pos, choice;
    printf("1. Insert \n 2. Delete \n 3. Exit \n 4 Choice: ");
    scanf("%d", &choice);
    while (choice != 3) {
        if (choice == 1) {
            printf("Enter data and pos: ");
            scanf("%d %d", &data, &pos);
            insert(data, pos);
            printf("Count: %d \n", count);
        } else if (choice == 2) {
            printf("Enter element: ");
            scanf("%d", &pos);
            delete(pos);
            printf("Count: %d \n", count);
        }
        display(); printf("Enter choice: "); scanf("%d", &choice);
    }
    return 0;
}
```

```
void insert (int data, int position){
    if (position == 0){
        node *new_node = malloc (sizeof (node));
        new_node -> data = data;
        new_node -> next = head;
        if (head != NULL) head -> prev = new_node;
        head = new_node; count++; return;
    } else if (position == count){
        node* new_node = malloc (sizeof (node));
        new_node -> data = data;
        new_node -> next = NULL;
        node *temp = head;
        while (temp -> next != NULL) temp = temp -> next;
        temp -> next = new_node;
        new_node -> prev = temp;
        count++; return;
    } else if (position > count || position < 0){
        printf ("Unable to insert at position %n");
        return;
    } else {
        node *temp = head;
        for (int i=0; i<position-1; i++){
            temp = temp -> next;
        node * new_node = malloc (sizeof (node));
        new_node -> data = data;
        new_node -> next = temp -> next;
        new_node -> prev = temp;
        temp -> next -> prev = new_node;
        temp -> next = new_node; count++; return;
    }
}
```

```
void delete(int element){
    int position = 0; node *temp = head;
    if (head == NULL){
        printf("List is empty. Cannot delete"); return;
    }
    for(; position < count; temp = temp->next, position++){
        if (temp->data == element) break;
    if (temp == NULL){
        printf("Element does not exist"); return;
    }
    if (position == 0){
        node *temp = head;
        temp = temp->next; temp->prev = NULL;
        free(head); head = temp; count--; return;
    } else if (position == count-1){
        node *temp = head;
        for(int i=1; i<count-1; i++) temp = temp->next;
        node *temp1 = temp->next;
        temp->next = NULL; free(temp1); count--; return;
    } else if (position > count || position < 0){
        printf("Unable to delete\n"); return;
    } else {
        node *temp = head;
        for(int i=0; i<position; i++) temp = temp->next;
        temp->next->prev = temp->prev;
        temp->prev->next = temp->next;
        free(temp); count--; return;
    }
}

void display(){
    node *temp = head;
    printf("Linked List: ");
```

```
void insert (int data, int position){
    if (position == 0){
        node *new_node = malloc (sizeof (node));
        new_node -> data = data;
        new_node -> next = head;
        if (head != NULL) head -> prev = new_node;
        head = new_node; Count++; return;
    } else if (position == Count){
        node* new_node = malloc (sizeof (node));
        new_node -> data = data;
        new_node -> next = NULL;
        node *temp = head;
        while (temp -> next != NULL) temp = temp -> next;
        temp -> next = new_node;
        new_node -> prev = temp;
        Count++; return;
    } else if (position > Count || position < 0){
        printf ("Unable to insert at position %d");
        return;
    } else {
        node *temp = head;
        for (int i=0; i< position-1; i++){
            temp = temp -> next;
        node * new_node = malloc (sizeof (node));
        new_node -> data = data;
        new_node -> next = temp -> next;
        new_node -> prev = temp;
        temp -> next -> prev = new_node;
        temp -> next = new_node; Count++; return;
    }
}
```

```
void delete (int element) {
    int position = 0; node *temp = head;
    if (head == NULL) {
        printf ("List is empty, Cannot delete"); return;
    }
    for (; position < count; temp = temp->next, position++) {
        if (temp -> data == element) break;
    if (temp == NULL) {
        printf ("Element does not exist\n"); return;
    }
    if (position == 0) {
        node *temp = head;
        temp = temp->next; temp->prev = NULL;
        free (head); head = temp; count--; return;
    } else if (position == count - 1) {
        node *temp = head;
        for (int i=1; i < count-1; i++) temp = temp->next;
        node *temp1 = temp->next;
        temp->next = NULL; free (temp1); count--; return;
    } else if (position > count+1 || position < 0) {
        printf ("Unable to delete\n"); return;
    } else {
        node *temp = head;
        for (int i=0; i<position; i++) temp = temp->next;
        temp->next->prev = temp->prev;
        temp->prev->next = temp->next;
        free (temp); count--; return;
    }
}

void display () {
    node *temp = head;
    printf ("Linked List: ");
```

```
while (temp -> next != NULL) {
    printf("%d ", temp->data);
    temp = temp->next;
}
printf("\n");
}
```

Output:

1. Insert
2. Delete
3. Exit

Choice: 1

Enter data and position: 1 0

Count: 1

Linked List: 1

Choice: 1

Enter data and position: 2 1

Count: 2

Linked List: 1 2

Choice: 1

Enter data and position: 3 2

Count: 3

Linked List: 1 2 3

Enter Choice: 1

Enter data and position: 4 1

Count: 4

Linked List: 1 4 2 3

Enter Choice: 2

Enter element: 4

Count: 3

Linked List: 1 2 3

Enter Choice: 2

Enter element: 3 → Count: 2
Linked list: 1 2
~~Exit~~ Choice: 2
Enter element: 1
Count: 1
Linked List: 1 2
Enter choice: 3

2) Splitting singly linked list: (1|2|24)

```
struct ListNode** splitListIntoParts(struct ListNode* head,
                                     int k, int* returnSize) {
    struct ListNode* temp = head; int n = 0;
    for (; temp != NULL; temp = temp->next, n++);
    struct ListNode** list = (struct ListNode**) malloc
                             (k * sizeof(struct ListNode*));
    for (int i=0; i<k; i++) list[i] = NULL;
    int earlier_list = n%k, size = n/k;
    int current = 0; temp = head; *returnSize = k;
    for (int i=0; earlier_list>0; i++, earlier_list--) {
        struct ListNode* temp1 = temp;
        list[current++] = temp;
        for (int j=0; j<size; j++) temp1 = temp1->next;
        temp = temp1->next; temp1->next = NULL;
    }
    if (temp == NULL) return list;
    for (int i=0; i<k-earlier_list; i++) {
        struct ListNode* temp1 = temp;
        if (temp1 == NULL) break;
        for (int j=0; j<size-1; j++) temp1 = temp1->next;
        list[current++] = temp;
        temp1->next = NULL;
    }
    return list;
}
```

Week-8

1) Binary Search Tree: (15/2/24)

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int data;
    struct Node *left;
    struct Node *right;
} node;
node *root = NULL;
void insert(node **root, int data);
void preorder(node **root);
void postorder(node **root);
void inorder(node **root);

int main() {
    int choice, data;
    insert(&root, 100); insert(&root, 20); insert(&root, 200);
    insert(&root, 10); insert(&root, 30); insert(&root, 150);
    insert(&root, 300);
    printf("1. Preorder\n 2. Postorder\n 3. Postorder\n 4. Exit\n Choice");
    scanf("%d", &choice);
    while (choice != 4) {
        if (choice == 1) {
            preorder(&root); printf("\n");
        } else if (choice == 2) {
            inorder(&root); printf("\n");
        } else if (choice == 3) {
            postorder(&root); printf("\n");
        }
        printf("Enter choice: ");
        scanf("%d", &choice);
```

```
void insert(node **root, int data){
    if (*root == NULL){
        node *new_node = malloc(sizeof(node));
        new_node->data = data;
        new_node->right = NULL;
        new_node->left = NULL;
        *root = new_node; return;
    }
    if (data < (*root)->data){
        insert(&((*root)->left), data);
    } else if (data > (*root)->data){
        insert(&((*root)->right), data);
    }
}
void preorder(node **root){
    if (*root != NULL){
        printf("%d ", (*root)->data);
        preorder(&((*root)->left));
        preorder(&((*root)->right));
    }
}
void postorder(node **root){
    if (*root != NULL){
        postorder(&((*root)->left));
        postorder(&((*root)->right));
        printf("%d ", (*root)->data);
    }
}
void inorder(node **root){
    if (*root != NULL){
        inorder(&((*root)->left));
        printf("%d ", (*root)->data);
        inorder(&((*root)->right));
    }
}
```

Output:

1 Preorder

2 Inorder

3 Postorder

4 Exit

Choice: 1

100 20 10 30 200 150 300

Choice: 2

10 20 30 100 150 200 300

Choice: 3

10 30 20 150 300 200 100

Choice: 4

Function

~~Stack~~ Tree:

Preorder:

100 → preorder (left) → 20 → preorder (left) → 10
→ preorder (left) → NULL → preorder (right) → ~~30~~ NULL → preorder (left) right
→ ~~NULL~~ 30 → preorder (left) → NULL → preorder (right) → NULL
→ preorder (right) → 200 → preorder (left) → 150 → preorder (left)
→ NULL → preorder (right) → NULL → preorder (right) →
300 → preorder (left) → NULL → preorder (right) → NULL

Postorder:

preorder (left) → preorder (left) → preorder (left) → 10 →
preorder (left) → NULL → preorder (right) → NULL → 20 →
NULL
preorder (right) → 30

preorder (left) → preorder (left) → preorder (left) → NULL
→ preorder (right) → NULL → 10 → preorder (right) → 30 preorder (left)
→ NULL → preorder (right) → NULL → 30 → 20 →
preorder (right) → preorder (left) → preorder (left) → NULL →
preorder (right) → NULL → 150 → preorder (right) → preorder (left)
NULL → preorder (right) → NULL → 300 → 200 → 100

Inorder:

inorder (left) → inorder (left) → inorder (left) → NULL → ~~inorder~~ → ~~inorder~~ → ~~inorder~~ → inorder (right) → inorder (left) → NULL → 30 → inorder (right) → NULL → 100 → inorder (right) → inorder (left) → inorder (left) → NULL → 150 → inorder (right) → NULL → 200 → inorder (right) → inorder (left) → NULL → 300 → inorder (right) → NULL

2) Rotate Right (5 | 2 | 24):

```
struct ListNode * rotateRight(struct ListNode *head, int k){
    struct ListNode *temp = head;
    if (head == NULL) return NULL;
    if (head -> next == NULL) return head;
    int size = 1;
    for (; temp -> next != NULL; temp = temp -> next, size++);
    k %= size;
    if (k == 0) return head;
    temp -> next = head;
    struct ListNode *temp1 = head;
    for (int i = 0; i < (size - k - 1); temp1 = temp1 -> next, i++);
    head = temp1 -> next;
    temp1 -> next = NULL;
    return head;
}
```

15/2/24

papergrid

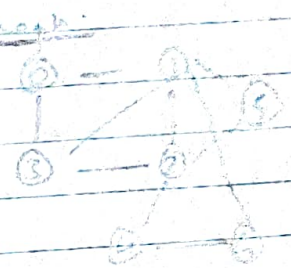
Date: / /

Week-9

1) Breadth-first search (22/2/24):

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define size 7

Void push(int a);
int pop();
void bfs(int graph[7][size]);

int fpos=-1; rpos=-1;
int queue[size];
int main(){
    int adj_matrix[7][7]={
        {0,1,0,1,0,0,0}, {1,0,1,1,0,1,1},
        {0,1,0,1,1,1,0}, {1,1,1,0,0,0,0},
        {0,0,1,0,0,0,1}, {0,1,1,0,0,0,0},
        {0,1,0,0,1,0,0} };
    for(int i=0; i<7; i++) queue[i]=NULL;
    bfs(adj_matrix); return 0;
}

void bfs(int graph[7][size]){
    int visited[size];
    for(int i=0; i<7; i++){ visited[i]=0;
    push(0); visited[0]=1;
    while(fpos != size){
        for(int i=0; i<size; i++){
            if(graph[queue[fpos]][i]==1 && visited[i]!=1)
                push(i);
                visited[i]=1;
        }
    }
    printf("%d ", pop());
```

```
void push (int a){
    if (fpos == -1 && rpos == -1){
        queue [++rpos] = a;
        fpos++; return;
    } else if (rpos == siz-1) {
        queue [++rpos] = a;
        return;
    }
}
int pop(){
    int n = queue [fpos];
    queue [fpos] = (int) NULL;
    fpos++;
    return n;
}
```



Output:

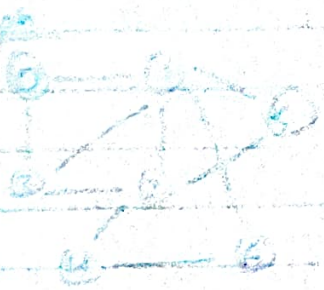
0 1 3 2 5 6 4

2) Depth-first search (22/2/024):

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define siz 7
int pos = -1;
int stack[siz];

void push (int a);
int pop ();
void display void dfs (int graph [7][siz]);
```

```
int main(){
    int adj_matrix[7][7] = {
        {0,1,0,1,0,0,0}, {1,0,1,1,0,1,1},
        {0,1,0,1,1,1,0}, {1,1,1,0,0,0,0},
        {0,0,1,0,0,0,1}, {0,1,1,0,0,0,0},
        {0,1,0,0,1,0,0},};
    for(int i=0; i<7; i++){ stack[i] = NULL;
    dfs(adj_matrix);
    return 0;
}

void dfs(int graph[7][size]){
    int visited[size];
    for(int i=0; i<7; i++) visited[i] = 0;
    push(0); visited[0]=1; printf("0 ");
    while(pos != -1){
        bool new_node = false;
        for(int i=0; i<7; i++){
            if(graph[stack[pos]][i] == 1 && visited[i]==0)
                new_node = true;
                push(i);
                visited[i] = 1; printf("->%d ", i);
                break;
            }
        }
        if(!new_node) pop();
    }
}
void push(int a){
    stack[++pos] = a;
}
void
```

```
int
void pop() {
    return stack[pos--];
}
```



Output:

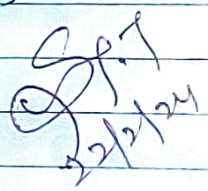
0 1 2 3 4 6

3) Swap nodes: (22/2/24):

```
struct node {
    int data;
    struct node *left;
    struct node *right;
};
struct node *create_node(int val){
    if (val == -1) return NULL;
    struct node *temp = malloc(sizeof(struct node));
    temp->data = val;
    temp->left = NULL;
    temp->right = NULL; return temp;
}
void inorder(struct node *root){
    if (!root) return;
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}
int max(int a, int b){
    if to return (a>b) ? a : b;
}
int height(struct node *root){
    if (!root) return 0;
    return (1 + max(height(root->left), height(root->right)));
```

```
void swap_nodes_at_level(struct node *root, int inc, int
    struct node *tnode;
    if (!root) return;
    if (level > height) return;
    if ( !(level % inc)) {
        tnode = root->left;
        root->left = root->right;
        root->right = tnode;
    }
    swap_nodes_at_level (root->left, inc, level+1, height);
    swap_nodes_at_level (root->right, inc, level+1, height);
}
int tail=0, head=0;
void push (struct node **queue, struct node *root) {
    queue[++tail] = root;
}
struct node* pop( struct node **queue){
    return queue[head++];
}

int main(){
    int nodes_count, i, temp, h, tc_num, index, inc, temp1, temp2;
    scanf("%d", &nodes_count);
    struct node *root_perm, *root_temp;
    struct node *q[nodes_count];
    for (i=0; i<nodes_count; i++) q[i] = NULL;
    i=0, index=1;
    root_temp = root_perm = create_node(1);
    push (q, root_temp);
    while ( index < 2*nodes_count ) {
        root_temp = pop(q);
        scanf("%d", &temp1);
        if (temp1 != -1) {
```

```
        root_temp->left = create_node(temp);
        push(q, root_temp->left);
    }
    scanf("%d", &temp);
    if(temp != -1){
        root_temp->right = create_node(temp);
        push(q, root_temp->right);
    }
    index += 2;
}
while(h = height(root_perm); scanf("%d", &tc_num);
while(tc_num){
    scanf("%d", &inc); temp = inc;
    swap_node_at_level(root_perm, inc, 1, h);
    inorder(root_perm);
    printf("\n");
    tc_num--;
}
return 0;
}
```



Week-10:

Linear Probing Hash Table (24/2/24):

```
#include <stdio.h>
#include <stlib.h>

#define siz 10

int table[siz];

void push(int data);
int pop(int data);
void search(int data);
void display();

int main(){
    for(int i=0; i<siz; i++) table[i] = -1;
    int choice;
    printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n Choice: ");
    scanf("%d", &choice);
    int a;
    while(choice != 3){
        case 1:
            printf("Enter integer to be pushed: ");
            scanf("%d", &a);
            push(a); break;
        case 2:
            printf("Enter integer to be popped: ");
            scanf("%d", &a);
            int res = pop(a);
            if(res == 0) printf("Integer popped\n");
            else printf("Integer not found\n");
            break;
```

```
        case 3:
            display(); break;
        default:
            printf("\n");
    }
    printf("Enter choice: ");
    scanf("%d", &choice);
  }
}

Void push(int data){
    int hash = hash % siz;
    while (table[hash] != -1 && hash <= (hash + siz-1))
    if (table[hash] == -1) table[hash] = data;
    else printf("Table is full");
}

int pop(int data){
    int hash = data % siz;
    for(int i=0, (table[hash] != data) || (i < siz), i++, );
    if (table[hash] == data){
        table[hash] = -1; return 0;
    }
    return -1;
}

Void display(){
    printf("Table ");
    for(int i=0; i<siz; i++) printf("%d ", table[i]);
    printf("\n");
}
```

Output:

1. Insert
2. Delete
3. Display
4. Exit

Choice: 1

Enter integer to be pushed: 1

Enter choice: 1

Enter integer to be ~~popped~~ pushed: 3

Enter choice: 1

Enter integer to be pushed: 90

Enter choice: 3

Table: 90 1 1 3 1 1 1 1 1

Enter Choice: ~~1~~ 1

Enter integer to be ~~popped~~ pushed: 13

Enter ~~inte~~ choice: 2

Enter integer to be ~~deleted~~ popped: 13

Integer popped

Enter choice: 4