

I N D E X

Name Pranav M Std 7 Sec A

Roll No. \_\_\_\_\_ Subject DS Lab - Obs. School/College \_\_\_\_\_

## Q) Swapping Integer Week -1 - Practice Program:

Q1P Current Balance : 1100

Current Balance : 1000

Current Balance : 1000

Sorted Array : Abhinav Pranav Prathmesh

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 18 not found

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);
```

```
}
```

Q1P

a = 3; b = 4

a = 4; b = 3

2) Stack implementation  
Dynamic Memory Allocation: (21/12/23)

```
#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\n2. Pop\n3. Display Stack\n4. Exit\nEnter 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 == 9) {
    printf("Stack Overflow Condition");
    return;
}
```

{

```
stack[pos] = a;
```

{

```
int pop() {
```

```
if (pos == -1) {
```

```
printf("Stack Underflow Condition");
```

```
return (int)NULL;
```

{

```
stack return stack[pos-1];
```

{

```
void display() {
```

```
for (int i=0; i<size; i++) {
```

```
printf("%d", stack[i]);
```

~~printf("\n");~~

{

Op

1. Push

2. Pop

3. Display Stack

olp

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

Enter choice: 4

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

#include &lt;stdio.h&gt;

#include &lt;stdlib.h&gt;

int size = 3;

int main()

int \*arr1 = malloc (size \* sizeof (int));

int \*arr2 = calloc (size, sizeof (int));

~~printf ("arr1 and arr2 can store %d integers\n",~~~~malloc (arr1, 2 \* size \* sizeof (int));~~

for (int i = 0; i &lt; size; i++)

arr2[i] = i;

for (int i = 0; i &lt; 2 \* size; i++)

arr1[i] = 0

```
#include <cs50.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
    // Declaring arrays
    int arr1[3];
    int arr2[3];

    // Inputting values into arr1
    for (int i = 0; i < 3; i++)
    {
        arr1[i] = i + 1;
    }

    // Outputting values from arr1
    printf("arr1: ");
    for (int i = 0; i < 3; i++)
    {
        printf("%d", arr1[i]);
    }
    printf("\n");

    // Inputting values into arr2
    for (int i = 0; i < 3; i++)
    {
        arr2[i] = i + 1;
    }

    // Outputting values from arr2
    printf("arr2: ");
    for (int i = 0; i < 3; i++)
    {
        printf("%d", arr2[i]);
    }
    printf("\n");

    // Freeing memory
    free(arr1);
    free(arr2);

    // Returning 0
    return 0;
}
```

Q1) arr1 and arr2 can store 3 integers

Can now store 6 integers in arr1 and 3 integers in arr2

arr1: 0 0 0 0 0 0

arr2: 1 1 1

Q2)  
2/12/23

Neek - 3

(28/12/23)

1) Infix to postfix

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

```
#define size 20
```

```
void push(char a);
```

```
char pop();
```

```
void display();
```

```
char* postfix(&char exp);
```

```
bool character(char c);
```

```
bool lower_preadence(char op1, char op2);
```

```
bool isEmpty();
```

```
int pos = -1;
```

```
char stack[size];
```

```
int n;
```

```
int main() {
```

```
    printf("Enter size of expression: ");
```

```
    scanf("%d", &n);
```

```
    fflush(stdin);
```

```
    char* exp = (char*) malloc((n+1)*sizeof(char));
```

```
    printf("Enter infix expression: ");
```

```
    scanf("%[^n]s", exp);
```

~~```
    char* postfix_exp = postfix(exp);
```~~~~```
    printf("Postfix expression: %s", postfix_exp);
```~~

```
}
```

```
bool isEmpty() {
```

```
    return pos == -1;
```

```
}
```

```

void push (char a) {
    if (pos == size - 1) {
        return printf ("Stack Overflow condition");
        return;
    }
    stack [++pos] = a;
}

char pop () {
    if (pos == -1) {
        return 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 < size; 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<n; i++) {
    if (op-anda[i] == op1) o1 = i;
    if (op-anda[i] == op2) o2 = i;
}
return o1 < o2
}

```

```

char* postfix(char* exp) {
    char* return_exp = (char*) malloc ((size_t) 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] == '/') push(exp[i]);
        else if (lower-precedence(stack[pos], exp[i])) {
            while (lower-precedence(stack[pos], exp[i])) {
                if (! isEmpty()) push(exp[i]);
                pos++;
            }
            pos--;
            break;
        }
    }
}

```

~~else~~

```

else push(exp[i]);
}
else if (exp[i] == '(') push ('(');
else if (exp[i] == ')') {
    while (stack[pos] != '(' && ! isEmpty())
        return_exp[current++] = pop();
}
}

```

```
while (! isEmpty (1))
```

```
    if (stack [pos] != '(') return-exp [current + ] + pop (1);
```

```
    else pos ++;
```

```
}
```

```
return return-exp;
```

```
}
```

olp Enter size of expression: 9

Enter infix expression: 102+3\*4-5

Postfix expression: 12+34\*++-

## 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("Unknown operator");
        }
    }
}
printf("Final value: %.d", pop());
```

```
bool isOperator (char op) {
    char operators[] = {'<', '>', '+', '*', '-'};
    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");
}
```~~

Q1p 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 {
```

```
void push(int a);
```

```
int pop();
```

```
void display();
```

```
int fpos=1, lpos=1; int queue [size];
```

```
int main()
```

```
int choice;
```

Enqueue | Dequeue | Display | Exit

```
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\n", a);
```

```
break;
```

Case 3:

```
display();
```

```
break;
```

default:

printf("Jdk");

break;

}

printf("Enter choice: ");

scanf("%d", &choice);

}

}

void push(int a){

if (fpos == -1 && rpos == -1){

queue[+rpos] = a;

fpos++;

return;

}

else if (rpos == size - 1){

printf("Queue overflow condition");

return;

}

else{

queue[+rpos] = 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. Insert  
2. Delete  
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 choice: 6

Enter choice: 3

Queue: 2 3 4 RL

Enter choice: 2

Popped integer = 2

Enter choice: 3

Queue: 0 3 4 RL

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, npos=1; int queue[Size];
```

```

int main() {
    int choice;
    printf("1. Enqueue 2. Dequeue 3. Display 4. Exit");
    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;
    }
}

```

```

    printf("Jdt");
    break;
}
}

```

```

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

```

void push (int a){

```

    if (fpos == -1 && rpos == -1) {
        queue[++rpos] = a;
        fpos++;
        return;
    }
}

```

```

else if ((queue[1].size == fpos - 1)) {
    printf("Queue Overflow condition");
    return;
}
}

```

```
else {
```

```
    Apot[i];
```

```
    queue[(fptr).size] = a;
```

```
    return;
```

```
}
```

```
}
```

```
int Pop() { // (fptr.size == Apot.size))
```

```
if (fptr == -1) {
```

```
    printf("Queue Underflow Condition");
```

```
}
```

```
int n = queue[fptr].size;
```

```
queue[fptr].size = (int) NULL;
```

```
fptr++;
```

```
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 choice: 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-1

1) Singly linked list:

(11/11/24)

#include <stdio.h>

#include <stdlib.h>

#include <statloc.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\nEnter 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");
}
```

```

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("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: 3 2

Count: 3

Linked List: 1 2 3

Enter Choice: 1

Enter data and position: 4 1

Count: 4

Linked List: 1 4 3 2

Enter Choice: 1

Enter data and position: 5 2

Count: 5

Linked List: 1 4 5 3 2

Enter Choice: 3

G.P.  
Molky

Week-5 (18/11/24)

1) Single linked List deletion (18/11/24)

#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\n2. Delete\n3. Exit\nChoice: ");

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 = templ = temp->next;  
        temp->next = NULL; free (temp);  
        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 = templ = temp->next;  
        temp->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 ("%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

List 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 positions: 1

Count: 2

Linked List: 3 4

Enter choice: 2

Enter position: 1 → Count: 1

Linked List: 3

Enter choice: 3

Coff  
18/1/24

2) MinStack: (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);  
}
```

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

Stack ListNode \* reverseBetween (StackListNodes \* head,  
int left, int right);

StackListNodes \* l = head;

StackListNodes \* r = head;

int difference = right - left;

if (left == right) return head;

for (int i=0; i < left-1; i++) {

    l = l->next;

for (int i=0; i < right-l; i++) {

    r = r->next;

**paintiff** while (difference > 0) {

        int temp = l->val;

        l->val = r->val;

        r->val = temp;

        l = l->next;

        r = r->next;

    } for (int i=0; i < difference-2; i++) {

        r = r->next;

    difference -= 2;

return head;

}

(S1)  
18/1/24

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 delete 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");

    reverse(); printf ("Reversed Linked List: \n"); display();

    head1 = head; head = NULL;

    insert(3,0); insert(4,1); insert(1,2); display();

    concat (&head1, &head); head1 = head;

    printf ("Concatenating with above linked list given: \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 *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;
}

```

```

3          node *temp;
void sort() {
    int i, j, min-index;
    node *i-node = head, *j-node = head, *min-node = NULL;
    for (int i=0; i<count-1; i++) {
        if (i-node->data > min-node->data) {
            min-index = i;
            min-node = i-node;
        }
        for (int j = i+1; j < count; j++) {
            if (j-node->data < min-node->data) {
                min-index = j;
                min-node = j-node;
            }
        }
        if (min-index != i) {
            int temp = i-node->data;
            int temp = i-node->data - j-node->data;
            min-node->data = temp;
        }
    }
}

```

void rever( )

node \*prev = NULL, \*next=NULL;

while (head != NULL)

next = head->next; head->next = prev;

prev = head; head = next;

}

head = prev;

}

void connect(\*node \*\*head, node \*\*new2)

node \*temp1, \*head1;

temp1->next != NULL

while (temp1->next->next != NULL)

temp1 = temp1->next;

}

temp1->next = \*head2;

}

void display()

( node \*temp = head);

printf("Linked List: ");

while (temp->next != NULL)

printf("%d ", temp->data);

}

printf("\n");

}

Output:

Original Linked List:

Linked List: 2 1 4 3 5

Sorted linked list:

Linked List: 1 2 3 4 5

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\n2. Pop\n3. Exit\nChoice: ");
```

```
    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());
```

```
        } else {
```

```
            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 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->next != 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:

Enter data: 5

Linked list: 2 5 3

Enter choice: 2

Integer popped: 5

Linked list: 2 3

Enter choice: 3

## 3) Queue with LL: (21/1/2023)

#include &lt;stdio.h&gt;

#include &lt;stdlib.h&gt;

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\nEnter choice: ");

scanf("%d", &amp;choice);

while (choice != 3) {

if (choice == 1) {

printf("Enter data: ");

scanf("%d", &amp;data);

insert(data);

} else if (choice == 2) {

3      printf("Enter node value : %d", &delte[1]);

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");

{

Ques 1

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

Last week 7

1) Doubly linked list: (1/2/20)

#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\n2. Delete\n3. Exit\n4. 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);

    }

```
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'\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"); 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", position);
```

```
    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--;
```

```
} else if (position > count || position < 0) {
```

```
    printf ("Unable to delete %d\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: 10

Count: 1

Linked List: 1

Choice: 1

Enter data and position: 21

Count: 2

Linked List: 1 2

Choice: 1

Enter data and position: 32

Count: 3

Linked List: 1 2 3

Enter Choice: 1

Enter data and position: 41

Linked Count: 4

Linked List: 1 4 2 3

Enter Choice: 2

Enter element: 4

Count: 3

Linked List: 1 2 3

Enter Element: 2

Enter element: 3 → Count: 2

Linked List: 1 2

Ent Chanc: 2

Enter element: 1

Count: 1

Linked List: 1 2

Ent Chanc: 3

## 2) Splitting Singly Linked List (1/2/24)

struct ListNode\* splitList Into Two (struct ListNode\* head,  
int k, int: return 3);

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->NULL;

int earlierList = n-k, size = n/k;

int current = 0; if (temp = head); /\* return size = k,

for (int i=0; i < earlierList; i++) {

struct ListNode\* temp1 = temp;

list [current++] = temp;

for (int j=0; j < size; j++) temp = temp->next;

temp->next = NULL; temp = temp->next;

temp1->next = NULL;

}

if (temp == NULL) return list;

for (int i=0; i < k - earlierList; i++) {

struct ListNode\* temp1 = temp;

if (temp1 == NULL) break;

for (int j=0; j < size-1; j++) temp1 = temp1->next;

list [current++] = temp1;

temp1->next = NULL;

} return list;

Week-8Binary Search Tree: (15/2/20)

#include &lt;stdio.h&gt;

#include &lt;stdlib.h&gt;

```
typedef struct Node {
    int data;
    struct Node *left;
    struct Node *right;
}
```

{ node;

node \*root = ~~NULL~~ 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, 22);~~  
~~insert(&root, 10); insert(&root, 30); insert(&root, 150);~~  
~~insert(&root, 300);~~

printf("1. Preorder\n2. Postorder\n3. Postorder\n4. Exit\nChoice:

scanf("%d", &amp;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");

} else {

printf("Enter 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);
}
```

```
3 void preorder(node **root) {
    if (*root != NULL) {
        printf(" %d ", (*root)->data);
        preorder(&((*root)->left));
        preorder(&((*root)->right));
    }
}
```

```
3 void postorder(node **root) {
    if (*root != NULL) {
        postorder(&((*root)->left));
        postorder(&((*root)->right));
        printf(" %d ", (*root)->data);
    }
}
```

```
3 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 30 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 Trace:

##### Preorder:

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

Inorder!

inorder(left) → inorder(left) → inorder(left) → NULL → ~~10~~ → ~~100~~  
 → ~~inorder~~ → ~~inorder~~ → ~~inorder~~ → ~~inorder~~ → ~~inorder~~  
 → ~~inorder~~ → ~~inorder~~ → ~~inorder~~ → ~~inorder~~ → ~~inorder~~  
 → NULL → 100 → inorder(right) → inorder(left) →  
 inorder(left) → NULL → 100 → inorder(right) → NULL →  
 200 → inorder(right) → inorder(left) → NULL → 300 →  
 inorder(right) → NULL

2) rotate Right(): (15/21 zu):

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 → next = head = temp1 → next;

temp1 → next = NULL;

return head;

3

N  
15/21 zu

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; npos = -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, 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++) if (visited[i] == 0)

push (i); visited[i] = 1;

while (fpos != size)

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

if (graph [queue[fpos]][i] == 1 && visited[i] == 0)

push (i);

visited[i] = 1;

} print ("d ", pop());

```

void push (int a) {
    if (lfpn == -1 & rfpn == -1) {
        queue [lfpn] = a;
        lfpn++;
        return;
    } else if (lfpn > rfpn) {
        queue [lfpn] = a;
        return;
    }
}

```

{

```

int pop() {
    int n = queue [lfpn];
    queue [lfpn] = (int) NULL;
    lfpn++;
    return n;
}

```

{

Output:

0 1 3 2 5 6 4



## 2) Depth-first search (22/2/2024):

```

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

```

```

#define size 7
int pos = -1;
int stack (size);

```

```

void push (int a);

```

```

int pop ();

```

```

void play - void dfs (link graph [7], stack);

```

```
int main() {
```

```
    int adjMatrix[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}, {1}
```

```
for (int i = 0; i < 7; i++) stack[i] = NULL;
```

```
dfs(adjMatrix);
```

```
return 0;
```

```
}
```

```
void dfs(int graph[7][7]) {
```

```
int visited[7];
```

```
for (int i = 0; i < 7; i++) visited[i] = 0;
```

```
push(0); visited[0] = 1; printf("0 ");
```

```
while (pos != -1) {
```

```
bool newNode = false;
```

```
for (int i = 0; i < 7; i++) {
```

```
if (graph[stack[pos]][i] == 1 && visited[i] == 0)
```

```
newNode = true;
```

```
push(i);
```

```
visited[i] = 1; printf("%d ", i);
```

```
brace();
```

```
}
```

```
}
```

```
if (!newNode) pop();
```

```
3
```

```
void push(int a) {
```

```
stack[++pos] = a;
```

```
3
```

```
void
```

```

int
fixed pop () {
    return Stack [pos - 1];
}

```

Output:

0 1 2 3 4 5 6

Gege

6

5

4

3

2

1

0

## 3) Swap nodes: (22/2/2u):

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** (a > b) **return** a; **else** **return** b;

}

int height (Struct node \*root) {

**if** (!root) **return** 0;

**return** (1 + max (height (root->left), height (root->right)));

```
void SwapNodeAtLevel (struct node *root, int inc, int level, node **node)
{
    if (!root) return;
    if (level > height) return;
    if ((level + inc) < 0)
        *node = root->left;
    else
        *node = root->right;
    root->left = root->right;
    root->right = *node;
}
```

SwapNodeAtLevel (root->left, inc, level + 1, height);  
SwapNodeAtLevel (root->right, inc, level + 1, height);

```
int tail = 0, head = 0;
void enqueuepush (list<node *> &queue, struct node *root) {
    queue.push_back (root);
}
```

```
struct node *pop (list<node *> &queue) {
    struct node *temp = queue[head];
    queue.pop_front();
    head++;
}
```

```
int main () {
    int nodes_count, i, temp, h, t, min, index, inc, temp1, temp2;
    Scanf ("%d", &nodes_count);
    struct node *root = NULL, *root_temp;
    struct node *q[nodes_count];
    for (i = 0; i < nodes_count; i++) q[i] = NULL;
    i = 0, index = 1;
    root = temp = root->left = create_node(1);
    push (q, root);
    while (index < 2 * nodes_count) {
        root = temp = pop (q);
        Scanf ("%d", &temp1);
        if (temp != -1) {
            if (temp1 < 0)
                SwapNodeAtLevel (root, inc, index, height);
            else
                SwapNodeAtLevel (root, inc, index, height);
        }
        index++;
    }
}
```

root-temp->left = create-node (lchild);  
 push (q, root-temp->left);

{

Scant ('l.d', &temp),

if (temp != null)

root-temp->right = create-node (rchild);

push (q, root-temp->right);

{

index += 2;

{

while (t < n > height (root-pewm)); Scant ('l.d', &temp),

while (t < num) {

Scant ('l.d', &inl); temp = inl;

temp->parent = last (root-pewm, inc, l, n);

inorder (root-pewm);

printf ("\n");

t < num --;

{

return 0;

{

S.P.T  
 2/2/24

Week - 10:Linear Probing Hash Table (2/12/21):

#include &lt;csdian.h&gt;

#include &lt;stdlib.h&gt;

#define size 10

int table [size];

void push (int data);

int pop (int data);

void search (int data);

void display();

int main () {

for (int i=0; i&lt;size; i++) table[i] = -1;

int choice;

printf ("1. Insert\n2. ~~Search~~\n3. ~~Delete~~\n4. Exit\nChoice: ");

scanf ("%d", &amp;choice);

int a;

while (choice != 3) {

Case 1:

printf ("Enter integer to be pushed: ");

scanf ("%d", &amp;a);

push(a); break;

case 2:

printf ("Enter integer to be popped: ");

scanf ("%d", &amp;a);

int res = pop(0);

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 % size;

    while (table[hash] != -1 && hash < (hash + size))

        if (table[hash] == -1) table[hash] = data;

        else printf("Table is full");

}

int pop(int data) {

    int hash = data % size;

    for (int i=0; (table[hash] != data) || (i < size), i++)

        if (table[hash] == data) {

            table[hash] = -1; return 0;

}

    return 1;

}

void display() {

    printf("Table ");

    for (int i=0; i < size, 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: 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: 2

Enter integer to be popped: 13

Enter choice: 2

Enter integer to be deleted: 13

Integer popped

Enter choice: 4

✓  
5/3/24