**Data Structure Programs**

* **Linear Search**

#include <stdio.h>

void search(int arr[], int num);

int main()

{

int num;

int arr[50];

int element;

printf("Enter the size of array: ");

scanf("%d", &num);

printf("Enter the array elements: ");

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

scanf("%d", &arr[i]);

}

search(arr, num);

return 0;

}

void search(int arr[], int num)

{

int element;

int i;

printf("Enter the element to search: ");

scanf("%d", &element);

for (i = 0; i < num; i++) {

if (arr[i] == element) {

printf("Element is found at position %d\n", i + 1);

return;

}

}

if (i == num) {

printf("Given element is not found!! \n");

}

}

* **Binary Search**

#include <stdio.h>

void search(int arr[], int num);

int main()

{

int arr[50];

int num;

int element;

printf("Enter the size of array: ");

scanf("%d", &num);

printf("Enter the array elements: ");

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

scanf("%d", &arr[i]);

}

search(arr, num);

return 0;

}

void search(int arr[], int num)

{

int element;

int first = 0;

int last = num - 1;

int middle = (first + last) / 2;

printf("Enter the element to search: ");

scanf("%d", &element);

while (first <= last) {

if (arr[middle] == element) {

printf("Element found at position %d\n", middle + 1);

break;

}

else if (element < arr[middle]) {

last = middle - 1;

} else {

first = middle + 1;

}

middle = (first + last) / 2;

}

if (first > last) {

printf("Given element not found!! \n");

}

}

* **Bubble Sort**

#include <stdio.h>

void bubble\_sort(int arr[], int num);

int main()

{

int arr[50];

int num;

printf("Enter the size of array: ");

scanf("%d", &num);

printf("Enter the array elements: ");

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

scanf("%d", &arr[i]);

}

bubble\_sort(arr, num);

return 0;

}

void bubble\_sort(int arr[], int num)

{

int i;

int temp;

for (i = 0; i < num; i++) {

for (int j = i + 1; j < num; j++) {

if (arr[i] >= arr[j]) {

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

printf("Array after sorting: ");

for (i = 0; i < num; i++) {

printf("%d ", arr[i]);

}

printf("\n");

}

* **Insertion Sort**

#include <stdio.h>

void insertion\_sort(int arr[], int num);

int main()

{

int arr[50];

int num;

printf("Enter the size of array: ");

scanf("%d", &num);

printf("Enter the array elements: ");

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

scanf("%d", &arr[i]);

}

insertion\_sort(arr, num);

return 0;

}

void insertion\_sort(int a[], int n)

{

int i, j, temp;

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

temp = a[i];

j = i - 1;

while(j >= 0 && temp <= a[j]) {

a[j+1] = a[j];

j--;

}

a[j+1] = temp;

}

printf("Array after sorting: ");

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

printf("%d ", a[i]);

}

printf("\n");

}

* **Merge Sort**

#include <stdio.h>

void merge(int arr[], int p, int q, int r);

void mergeSort(int arr[], int l, int r);

void printArray(int arr[], int size);

int main()

{

int arr[50];

int size;

printf("Enter the size of array: ");

scanf("%d", &size);

printf("Enter the array elements: ");

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

scanf("%d", &arr[i]);

}

mergeSort(arr, 0, size - 1);

printf("Sorted array: \n");

printArray(arr, size);

}

void merge(int arr[], int p, int q, int r) {

int n1 = q - p + 1;

int n2 = r - q;

int L[n1], M[n2];

for (int i = 0; i < n1; i++)

L[i] = arr[p + i];

for (int j = 0; j < n2; j++)

M[j] = arr[q + 1 + j];

int i, j, k;

i = 0;

j = 0;

k = p;

while (i < n1 && j < n2) {

if (L[i] <= M[j]) {

arr[k] = L[i];

i++;

} else {

arr[k] = M[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = M[j];

j++;

k++;

}

}

void mergeSort(int arr[], int l, int r) {

if (l < r) {

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

void printArray(int arr[], int size) {

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

printf("%d ", arr[i]);

printf("\n");

}

* **Quick Sort**

#include <stdio.h>

void quick\_sort(int arr[], int, int);

int main()

{

int arr[50];

int num;

int i;

printf("Enter the size of array: ");

scanf("%d", &num);

printf("Enter the array elements: ");

for (i = 0; i < num; i++) {

scanf("%d", &arr[i]);

}

quick\_sort(arr, 0, num - 1);

printf("Array after sorting: ");

for (i = 0; i < num; i++) {

printf("%d ", arr[i]);

}

printf("\n");

}

void quick\_sort(int arr[], int first, int last)

{

int pivot;

int i;

int j;

int temp;

if (first < last) {

pivot = first;

i = first;

j = last;

while (i < j) {

while (arr[i] <= arr[pivot] && i < last)

i++;

while (arr[j] > arr[pivot])

j--;

if (i < j) {

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

temp = arr[pivot];

arr[pivot] = arr[j];

arr[j] = temp;

quick\_sort(arr, first, j - 1);

quick\_sort(arr, j + 1, last);

}

}

* **Single Linked list**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdio\_ext.h>

// Function Declaration

void create(char\* name, int id, char gen, float sal, int len);

void display();

void insert(int pos, char\* name, int id, char gen, float sal, int len);

void delete(int pos);

void reverse();

void count();

struct employee {

char\* name;

int id;

char gender;

float salary;

struct employee\* link;

}\*start; // Variable to store address of the starting node

int main()

{

int choice, n, i, len;

char\* name;

int id, pos;

char gen;

float sal;

while(1) {

printf("\n\_\_\_\_\_Enter the choice\_\_\_\_\_\n1. Create\n2. Insert\n3. Delete\n4. Reverse\n5. Display\n6. Exit\n7. Count\n");

scanf("%d", &choice); // Variable to store choice

switch (choice)

{

case 1:

start = NULL;

printf("Enter the number of employee: ");

scanf("%d", &n);

\_\_fpurge(stdin);

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

printf("\nEnter the length of name: ");

scanf("%d", &len);

\_\_fpurge(stdin);

name = (char\*) malloc (len \* sizeof(char));

if (NULL == name) {

printf("Malloc failed for name\n");

exit(1);

}

printf("Enter the name: ");

if (NULL == fgets(name, len + 2, stdin)) {

printf("Fgets failed for name\n");

}

\*(name + strlen(name) - 1) = '\0';

printf("Enter the id: ");

scanf("%d", &id);

\_\_fpurge(stdin);

printf("Enter the gender(F / M / O): ");

scanf("%c", &gen);

\_\_fpurge(stdin);

printf("Enter the Salary: ");

scanf("%f", &sal);

\_\_fpurge(stdin);

create(name, id, gen, sal, len);

}

printf("\_\_\_\_\_Successfully created linked list\_\_\_\_\_\n");

break;

case 2:

printf("Enter the position to insert: ");

scanf("%d", &pos);

\_\_fpurge(stdin);

printf("\nEnter the length of name: ");

scanf("%d", &len);

\_\_fpurge(stdin);

name = (char\*) malloc (len \* sizeof(char));

if (NULL == name) {

printf("Malloc failed for name\n");

exit(1);

}

printf("Enter the name: ");

if (NULL == fgets(name, len + 2, stdin)) {

printf("Fgets failed for name\n");

}

\*(name + strlen(name) - 1) = '\0';

printf("Enter the id: ");

scanf("%d", &id);

\_\_fpurge(stdin);

printf("Enter the gender(F / M / O): ");

scanf("%c", &gen);

\_\_fpurge(stdin);

printf("Enter the Salary: ");

scanf("%f", &sal);

\_\_fpurge(stdin);

insert(pos, name, id, gen, sal, len);

printf("\_\_\_\_\_\_Successfully Inserted\_\_\_\_\_\_\n");

break;

case 3:

printf("Enter the position to delete: ");

scanf("%d", &pos);

\_\_fpurge(stdin);

delete(pos);

printf("\_\_\_\_\_Successfully Deleted\_\_\_\_\_\_\n");

break;

case 4:

reverse();

printf("\_\_\_\_\_Successfully reversed the list\_\_\_\_\_\n");

break;

case 5:

display();

break;

case 6:

exit(1);

case 7:

count();

break;

default:

printf("Enter the valid input\n");

}

}

free(start);

free(name);

start = NULL;

name = NULL;

}

// Function to create a linked list

void create(char\* name, int id, char gen, float sal, int len)

{

struct employee \*tmp = NULL;

struct employee \*q;

tmp = (struct employee\*) malloc (sizeof(struct employee));

if (NULL == tmp) {

printf("Malloc failed for tmp\n");

exit(1);

}

tmp->name = (char\*) malloc (len \* sizeof(char));

if (NULL == tmp->name) {

printf("Malloc failed for tmp->link\n");

exit(1);

}

strcpy(tmp->name, name);

tmp->id = id;

tmp->gender = gen;

tmp->salary = sal;

tmp->link = NULL;

if (start == NULL) {

start = tmp;

} else {

q = start;

while (q->link != NULL) {

q = q->link;

}

q->link = tmp;

}

q = NULL;

}

void insert(int pos, char\* name, int id, char gen, float sal, int len)

{

int i = 1;

struct employee \*tmp = NULL;

struct employee \*q;

tmp = (struct employee\*) malloc (sizeof(struct employee));

if (NULL == tmp) {

printf("Malloc failed for temp\n");

exit(1);

}

tmp->name = (char\*) malloc (len \* sizeof(char));

if (NULL == tmp->name) {

printf("Malloc failed for tmp->name\n");

exit(1);

}

strcpy(tmp->name, name);

tmp->id = id;

tmp->gender = gen;

tmp->salary = sal;

if (start == NULL) {

tmp->link = NULL;

start = tmp;

} else if (pos == 1) {

tmp->link = start;

start = tmp;

} else {

q = start;

while (i != pos - 1) {

q = q->link;

i++;

}

tmp->link = q->link;

q->link = tmp;

}

q = NULL;

}

void delete(int pos)

{

int i = 1;

struct employee \*q;

struct employee \*tmp;

if (pos == 1) {

tmp = start;

start = start->link;

free(tmp);

} else {

q = start;

while (i != pos - 1) {

q = q->link;

i++;

}

tmp = q->link;

q->link = tmp->link;

free(tmp);

}

q = NULL;

}

void reverse()

{

struct employee \*p1;

struct employee \*p2;

struct employee \*p3;

if (start->link == NULL) {

return;

}

p1 = start;

p2 = p1->link;

p3 = p2->link;

p1->link = NULL;

p2->link = p1;

while (p3 != NULL) {

p1 = p2;

p2 = p3;

p3 = p2->link;

p2->link = p1;

}

start = p2;

p1 = NULL;

p2 = NULL;

p3 = NULL;

}

void display()

{

struct employee \*q;

int i = 1;

if (start == NULL) {

printf("List is empty\n");

return;

}

q = start;

while (q != NULL) {

printf("\nEmployee %d:\nName : %s\n", i, q->name);

printf("Id : %d\n", q->id);

printf("Gender : %c\n", q->gender);

printf("Salary : %f\n", q->salary);

q = q->link;

i++;

}

printf("\n");

q = NULL;

}

void count()

{

struct employee \*q;

int count = 0;

if (start == NULL) {

printf("List is empty\n");

return;

}

q = start;

while (q != NULL) {

count++;

q = q->link;

}

printf("\nNumber of employees are %d\n", count);

}

* **Double Linked List**

#include <stdio.h>

#include <stdlib.h>

#include<stdio\_ext.h>

void create(int data, char ch);

void insert(int data, char ch, int pos);

void delete(int pos);

void reverse();

void display();

struct node {

int num;

char alpha;

struct node \*next;

struct node \*prev;

}\*start;

int main()

{

int choice;

int n;

int i;

int data;

char ch;

int pos;

while(1) {

printf("\nEnter the choice\n 1. Create\n 2. Insert\n 3. Delete\n 4. Reverse\n 5. Display\n 6. Exit\n");

scanf("%d", &choice);

switch (choice) {

case 1:

start = NULL;

printf("Enter the number of data: ");

scanf("%d", &n);

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

printf("\nEnter the num: ");

scanf("%d", &data);

\_\_fpurge(stdin);

printf("Enter the alphabet: ");

scanf("%c", &ch);

create(data, ch);

}

break;

case 2:

printf("Enter the pos: ");

scanf("%d", &pos);

printf("Enter the num: ");

scanf("%d", &data);

\_\_fpurge(stdin);

printf("Enter the alphabet: ");

scanf("%c", &ch);

insert(data, ch, pos);

break;

case 3:

printf("Enter the pos: ");

scanf("%d", &pos);

delete(pos);

break;

case 4:

reverse();

break;

case 5:

display();

break;

case 6:

exit(1);

break;

default:

printf("Enter the right choice\n");

}

}

}

void create(int data, char ch)

{

struct node \*tmp;

struct node \*q;

tmp = (struct node\*) malloc (sizeof(struct node));

tmp->num = data;

tmp->alpha = ch;

tmp->next = NULL;

if (start == NULL) {

start = tmp;

tmp->prev = NULL;

} else {

q = start;

while (q->next != NULL) {

q = q->next;

}

q->next = tmp;

tmp->prev = q;

}

}

void insert(int data, char ch, int pos) {

struct node \*tmp;

struct node \*q;

int i = 1;

tmp = (struct node\*) malloc (sizeof(struct node));

tmp->num = data;

tmp->alpha = ch;

if (i == pos) {

tmp->prev = NULL;

tmp->next = start;

start->prev = tmp;

start = tmp;

} else {

q = start;

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

q = q->next;

if ( q == NULL) {

printf("less nodes");

break;

}

i++;

}

tmp->next = q->next;

tmp->prev = q;

if (q->next != NULL) {

q->next->prev = tmp;

}

q->next = tmp;

}

}

void delete(int pos) {

struct node \*q;

struct node \*tmp;

int i = 1;

if (pos == 1) {

tmp = start;

start = tmp->next;

tmp->next->prev = NULL;

free(tmp);

} else {

q = start;

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

q = q->next;

i++;

}

tmp = q->next;

q->next = tmp->next;

if (q->next != NULL) {

q->next->prev = q;

}

free(tmp);

}

}

void reverse()

{

struct node \*tmp;

int i = 1;

tmp = start;

while (tmp->next != NULL) {

tmp = tmp->next;

}

while (tmp != NULL) {

printf("\nNode %d\n Num - %d\n", i, tmp->num);

printf(" Alpha - %c\n", tmp->alpha);

tmp = tmp->prev;

i++;

}

}

void display()

{

struct node \*tmp;

int i = 1;

tmp = start;

while (tmp != NULL) {

printf("\nNode %d\n Num - %d\n", i, tmp->num);

printf(" Alpha - %c\n", tmp->alpha);

tmp = tmp->next;

i++;

}

}

* **Circular single Linked List**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdio\_ext.h>

// Function Declaration

void create(char\* name, int id, char gen, float sal, int len);

void display();

void insert(int pos, char\* name, int id, char gen, float sal, int len);

void delete(int pos);

void reverse();

void search(int id);

struct employee {

char\* name;

int id;

char gender;

float salary;

struct employee\* link;

}\*start; // Variable to store address of the starting node

int main()

{

int choice, n, i, len;

char\* name;

int id, pos;

char gen;

float sal;

while(1) {

printf("\n\_\_\_\_\_Enter the choice\_\_\_\_\_\n1. Create\n2. Insert\n3. Delete\n4. Reverse\n5. Search\n6. Display\n7. Exit\n");

scanf("%d", &choice); // Variable to store choice

switch (choice)

{

case 1:

start = NULL;

printf("Enter the number of employee: ");

scanf("%d", &n);

\_\_fpurge(stdin);

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

printf("\nEnter the length of name: ");

scanf("%d", &len);

\_\_fpurge(stdin);

name = (char\*) malloc (len \* sizeof(char));

if (NULL == name) {

printf("Malloc failed for name\n");

exit(1);

}

printf("Enter the name: ");

if (NULL == fgets(name, len + 2, stdin)) {

printf("Fgets failed for name\n");

}

\*(name + strlen(name) - 1) = '\0';

printf("Enter the id: ");

scanf("%d", &id);

\_\_fpurge(stdin);

printf("Enter the gender(F / M / O): ");

scanf("%c", &gen);

\_\_fpurge(stdin);

printf("Enter the Salary: ");

scanf("%f", &sal);

\_\_fpurge(stdin);

create(name, id, gen, sal, len);

}

printf("\_\_\_\_\_Successfully created linked list\_\_\_\_\_\n");

break;

case 2:

printf("Enter the position to insert: ");

scanf("%d", &pos);

\_\_fpurge(stdin);

printf("\nEnter the length of name: ");

scanf("%d", &len);

\_\_fpurge(stdin);

name = (char\*) malloc (len \* sizeof(char));

if (NULL == name) {

printf("Malloc failed for name\n");

exit(1);

}

printf("Enter the name: ");

if (NULL == fgets(name, len + 2, stdin)) {

printf("Fgets failed for name\n");

}

\*(name + strlen(name) - 1) = '\0';

printf("Enter the id: ");

scanf("%d", &id);

\_\_fpurge(stdin);

printf("Enter the gender(F / M / O): ");

scanf("%c", &gen);

\_\_fpurge(stdin);

printf("Enter the Salary: ");

scanf("%f", &sal);

\_\_fpurge(stdin);

insert(pos, name, id, gen, sal, len);

printf("\_\_\_\_\_\_Successfully Inserted\_\_\_\_\_\_\n");

break;

case 3:

printf("Enter the position to delete: ");

scanf("%d", &pos);

\_\_fpurge(stdin);

delete(pos);

printf("\_\_\_\_\_Successfully Deleted\_\_\_\_\_\_\n");

break;

case 4:

reverse();

printf("\_\_\_\_\_Successfully reversed the list\_\_\_\_\_\n");

break;

case 5:

printf("Enter the employee id to search: ");

scanf("%d", &id);

search(id);

break;

case 6:

display();

break;

case 7:

exit(1);

break;

default:

printf("Enter the valid input\n");

}

}

free(start);

free(name);

start = NULL;

name = NULL;

}

// Function to create a linked list

void create(char\* name, int id, char gen, float sal, int len)

{

struct employee \*tmp = NULL;

struct employee \*q;

tmp = (struct employee\*) malloc (sizeof(struct employee));

if (NULL == tmp) {

printf("Malloc failed for tmp\n");

exit(1);

}

tmp->name = (char\*) malloc (len \* sizeof(char));

if (NULL == tmp->name) {

printf("Malloc failed for tmp->link\n");

exit(1);

}

strcpy(tmp->name, name);

tmp->id = id;

tmp->gender = gen;

tmp->salary = sal;

if (start == NULL) {

start = tmp;

} else {

q = start;

while (q->link != start) {

q = q->link;

}

q->link = tmp;

}

tmp->link = start;

q = NULL;

}

void insert(int pos, char\* name, int id, char gen, float sal, int len)

{

int i = 1;

struct employee \*tmp = NULL;

struct employee \*q;

tmp = (struct employee\*) malloc (sizeof(struct employee));

if (NULL == tmp) {

printf("Malloc failed for temp\n");

exit(1);

}

tmp->name = (char\*) malloc (len \* sizeof(char));

if (NULL == tmp->name) {

printf("Malloc failed for tmp->name\n");

exit(1);

}

strcpy(tmp->name, name);

tmp->id = id;

tmp->gender = gen;

tmp->salary = sal;

if (pos == 1) {

tmp->link = start;

start = tmp;

} else {

q = start;

while (i != pos - 1) {

q = q->link;

i++;

}

tmp->link = q->link;

q->link = tmp;

}

q = NULL;

}

void delete(int pos)

{

int i = 1;

struct employee \*q;

struct employee \*tmp;

q = start;

if (pos == 1) {

tmp = start;

while (q->link != start) {

q = q->link;

}

q->link = start;

start = start->link;

free(tmp);

} else {

q = start;

while (i != pos - 1) {

q = q->link;

i++;

}

tmp = q->link;

q->link = tmp->link;

free(tmp);

}

q = NULL;

}

void reverse()

{

struct employee \*p1;

struct employee \*p2;

struct employee \*p3;

if (start->link == start) {

return;

}

p1 = start;

p2 = p1->link;

p3 = p2->link;

p2->link = p1;

while (p3 != start) {

p1 = p2;

p2 = p3;

p3 = p2->link;

p2->link = p1;

}

start->link = p2;

start = p2;

p1 = NULL;

p2 = NULL;

p3 = NULL;

}

void display()

{

struct employee \*q;

int i = 1;

if (start == NULL) {

printf("List is empty\n");

return;

}

q = start;

while (q->link != start) {

printf("\nEmployee %d:\nName : %s\n", i, q->name);

printf("Id : %d\n", q->id);

printf("Gender : %c\n", q->gender);

printf("Salary : %f\n", q->salary);

q = q->link;

i++;

}

printf("\nEmployee %d:\nName : %s\n", i, q->name);

printf("Id : %d\n", q->id);

printf("Gender : %c\n", q->gender);

printf("Salary : %f\n", q->salary);

printf("\n");

q = NULL;

}

void search(int id)

{

struct employee \*q;

int count = 1;

if (start == NULL) {

printf("List is empty\n");

return;

}

q = start;

while (q != NULL) {

if (q->id == id) {

printf("Employee id found at position %d\n", count);

return;

}

count++;

q = q->link;

}

printf("Employee id not found\n");

}

* **Circular Double Linked List**

#include <stdio.h>

#include <stdlib.h>

#include<stdio\_ext.h>

void create(int data, char ch);

void insert(int data, char ch, int pos);

void delete(int pos);

void reverse();

void display();

struct node {

int num;

char alpha;

struct node \*next;

struct node \*prev;

}\*start;

int main()

{

int choice;

int n;

int i;

int data;

char ch;

int pos;

while(1) {

printf("\nEnter the choice\n 1. Create\n 2. Insert\n 3. Delete\n 4. Reverse\n 5. Display\n 6. Exit\n");

scanf("%d", &choice);

switch (choice) {

case 1:

start = NULL;

printf("Enter the number of data: ");

scanf("%d", &n);

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

printf("\nEnter the num: ");

scanf("%d", &data);

\_\_fpurge(stdin);

printf("Enter the alphabet: ");

scanf("%c", &ch);

create(data, ch);

}

break;

case 2:

printf("Enter the pos: ");

scanf("%d", &pos);

printf("Enter the num: ");

scanf("%d", &data);

\_\_fpurge(stdin);

printf("Enter the alphabet: ");

scanf("%c", &ch);

insert(data, ch, pos);

break;

case 3:

printf("Enter the pos: ");

scanf("%d", &pos);

delete(pos);

break;

case 4:

reverse();

break;

case 5:

display();

break;

case 6:

exit(1);

break;

default:

printf("Enter the right choice\n");

}

}

}

void create(int data, char ch)

{

struct node \*tmp;

struct node \*q;

tmp = (struct node\*) malloc (sizeof(struct node));

tmp->num = data;

tmp->alpha = ch;

if (start == NULL) {

start = tmp;

tmp->prev = start;

} else {

q = start;

while (q->next != start) {

q = q->next;

}

q->next = tmp;

tmp->prev = q;

start->prev = tmp;

}

tmp->next = start;

}

void insert(int data, char ch, int pos) {

struct node \*tmp;

struct node \*q;

int i = 1;

tmp = (struct node\*) malloc (sizeof(struct node));

tmp->num = data;

tmp->alpha = ch;

if (i == pos) {

tmp->prev = start->prev;

tmp->next = start;

start->prev->next = tmp;

start->prev = tmp;

start = tmp;

} else {

q = start;

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

q = q->next;

if ( q == NULL) {

printf("less nodes");

break;

}

i++;

}

tmp->next = q->next;

tmp->prev = q;

if (tmp->next != start) {

q->next->prev = tmp;

}

q->next = tmp;

}

}

void delete(int pos) {

struct node \*q;

struct node \*tmp;

int i = 1;

if (pos == 1) {

tmp = start;

start = tmp->next;

tmp->prev->next = start;

start->prev = tmp->prev;

free(tmp);

} else {

q = start;

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

q = q->next;

i++;

}

tmp = q->next;

q->next = tmp->next;

if (tmp->next != start) {

q->next->prev = q;

}

free(tmp);

}

}

void reverse()

{

struct node \*tmp;

int i = 1;

tmp = start;

while (tmp->next != start) {

tmp = tmp->next;

}

while (tmp != start) {

printf("\nNode %d\n Num - %d\n", i, tmp->num);

printf(" Alpha - %c\n", tmp->alpha);

tmp = tmp->prev;

i++;

}

printf("\nNode %d\n Num - %d\n", i, tmp->num);

printf(" Alpha - %c\n", tmp->alpha);

}

void display()

{

struct node \*tmp;

int i = 1;

tmp = start;

while (tmp->next != start) {

printf("\nNode %d\n Num - %d\n", i, tmp->num);

printf(" Alpha - %c\n", tmp->alpha);

tmp = tmp->next;

i++;

}

printf("\nNode %d\n Num - %d\n", i, tmp->num);

printf(" Alpha - %c\n", tmp->alpha);

}

* **Stack**

#include <stdio.h>

#include <stdlib.h>

#define SIZE 10

void push(int);

void pop();

void display();

int top = -1;

int stack[SIZE];

void main()

{

int value, choice;

while(1) {

printf("\nStack Operations:\n");

printf("1. Push\n2. Pop\n3. Display\n4. Exit");

printf("\nEnter your choice: ");

scanf("%d",&choice);

switch(choice){

case 1: printf("Enter the value to be insert: ");

scanf("%d",&value);

push(value);

break;

case 2: pop();

break;

case 3: display();

break;

case 4: exit(0);

default: printf("\nWrong selection!!! Try again!!!");

}

}

}

void push(int value){

if(top == SIZE-1)

printf("\nStack is Full!!! Insertion is not possible!!!");

else{

top++;

stack[top] = value;

printf("\nInsertion success!!!\n");

}

}

void pop(){

if(top == -1)

printf("\nStack is Empty!!! Deletion is not possible!!!");

else{

printf("\nDeleted : %d\n", stack[top]);

top--;

}

}

void display(){

if(top == -1)

printf("\nStack is Empty!!!");

else{

int i;

printf("\nStack elements are:\n");

for (i = top; i >= 0; i--)

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

}

}

* **Queue**

#include <stdio.h>

#include <stdlib.h>

#define SIZE 10

void enqueue(int);

void dequeue();

void display();

int arr[SIZE];

int front = -1;

int rear = -1;

void main()

{

int value;

int choice;

while(1) {

printf("\n\nQueue Operations: \n");

printf("1. Insertion\n2. Deletion\n3. Display\n4. Exit");

printf("\nEnter your choice: ");

scanf("%d",&choice);

switch(choice) {

case 1: printf("Enter the value to be insert: ");

scanf("%d",&value);

enqueue(value);

break;

case 2: dequeue();

break;

case 3: display();

break;

case 4: exit(0);

default: printf("\nWrong selection!!! Try again!!!");

}

}

}

void enqueue(int value){

if(rear == SIZE-1) {

printf("\nQueue is Full!!! Insertion is not possible!!!");

} else {

if(front == -1)

front = 0;

rear++;

arr[rear] = value;

printf("\nInsertion success!!!");

}

}

void dequeue(){

if(rear == -1) {

printf("\nQueue is Empty!!! Deletion is not possible!!!");

} else {

printf("\nDeleted : %d", arr[front]);

front++;

if(front > rear) {

front = rear = -1;

}

}

}

void display(){

if(rear == -1) {

printf("\nQueue is Empty!!!");

} else {

int i;

printf("\nQueue elements are:\n");

for (i = rear; i >= front; i--) {

printf("%d\t", arr[i]);

}

}

}

* **Stack Using Queue**

#include <stdio.h>

#include <stdlib.h>

#define SIZE 10

void push(int);

void pop();

void top();

void display();

int arr[SIZE];

int front = -1;

int rear = -1;

void main()

{

int value;

int choice;

while(1) {

printf("\n\nQueue Operations: \n");

printf("1. Insertion\n2. Deletion\n3. Top\n4. Display\n5. Exit");

printf("\nEnter your choice: ");

scanf("%d",&choice);

switch(choice) {

case 1: printf("Enter the value to be insert: ");

scanf("%d",&value);

push(value);

break;

case 2: pop();

break;

case 3: top();

break;

case 4: display();

break;

case 5: exit(0);

default: printf("\nWrong selection!!! Try again!!!");

}

}

}

void push(int value){

if(rear == SIZE - 1) {

printf("\nQueue is Full!!! Insertion is not possible!!!");

} else {

if(front == -1)

front = 0;

rear++;

arr[rear] = value;

if (rear > 0) {

for (int i = front; i < rear; i++) {

arr[rear + 1] = arr[front];

for (int j = front; j < rear + 1; j++) {

arr[j] = arr[j + 1];

}

}

}

printf("\nInsertion success!!!");

}

}

void pop(){

if(rear == -1) {

printf("\nQueue is Empty!!! Deletion is not possible!!!");

} else {

printf("\nDeleted : %d", arr[front]);

front++;

if(front > rear) {

front = rear = -1;

}

}

}

void top() {

if(rear == -1) {

printf("\nQueue is Empty!!! Deletion is not possible!!!");

} else {

printf("Top element is: %d", arr[front]);

}

}

void display(){

if(rear == -1) {

printf("\nQueue is Empty!!!");

} else {

int i;

printf("\nQueue elements are:\n");

for(i=front; i<=rear; i++) {

printf("%d\t", arr[i]);

}

}

}

* **Queue Using Stack**

#include <stdio.h>

#include <stdlib.h>

#define SIZE 10

void push(int);

void pop();

void display();

int top = -1;

int stack[SIZE];

void main()

{

int value, choice;

while(1) {

printf("\nStack Operations:\n");

printf("1. Push\n2. Pop\n3. Display\n4. Exit");

printf("\nEnter your choice: ");

scanf("%d",&choice);

switch(choice){

case 1: printf("Enter the value to be insert: ");

scanf("%d",&value);

push(value);

break;

case 2: pop();

break;

case 3: display();

break;

case 4: exit(0);

default: printf("\nWrong selection!!! Try again!!!");

}

}

}

void push(int value){

if(top == SIZE-1)

printf("\nStack is Full!!! Insertion is not possible!!!");

else {

top++;

stack[top] = value;

printf("\nInsertion success!!!\n");

}

}

void pop(){

int arr[SIZE];

int i, j;

if(top == -1)

printf("\nStack is Empty!!! Deletion is not possible!!!");

else {

j = top;

for (i = 0; i <= top; i++) {

arr[i] = stack[j];

j--;

}

printf("\nDeleted : %d\n", arr[top]);

top--;

j = top;

for (i = 0; i <= top; i++) {

stack[i] = arr[j];

j--;

}

}

}

void display(){

if(top == -1)

printf("\nStack is Empty!!!");

else {

int i;

printf("\nStack elements are:\n");

for(i = top; i >= 0; i--) {

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

}

}

}

* **Loop Detection**

#include <stdio.h>

#include <stdlib.h>

void create(int data, int i, int n);

int detect();

void myremove();

void display();

struct node {

int num;

struct node \*next;

}\*start;

int main()

{

int n;

int i;

int data;

int c;

start = NULL;

printf("Enter the number of data: ");

scanf("%d", &n);

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

printf("\nEnter the num: ");

scanf("%d", &data);

create(data, i, n);

}

c = detect();

if (c) {

printf("Loop detected at node %d\n", c);

myremove();

printf("\nLinked list after removing loop\n");

display();

} else {

printf("No loop found\n");

}

}

void create(int data, int i, int n)

{

struct node \*tmp;

struct node \*q;

struct node \*temp;

tmp = (struct node\*) malloc (sizeof(struct node));

tmp->num = data;

if (i == 3) {

temp = tmp;

}

if (i == n) {

tmp->next = temp;

} else {

tmp->next = NULL;

}

if (start == NULL) {

start = tmp;

} else {

q = start;

while (q->next != NULL) {

q = q->next;

}

q->next = tmp;

}

}

int detect()

{

struct node \*slow;

struct node \*fast;

int count = 1;

slow = start;

fast = start;

while (slow != NULL || fast != NULL) {

slow = slow->next;

fast = fast->next->next;

if (slow == fast) {

return count;

}

count++;

}

return 0;

}

void myremove()

{

struct node \*slow;

struct node \*fast;

slow = start;

fast = start;

while (slow != NULL || fast != NULL) {

slow = slow->next;

fast = fast->next->next;

if (slow == fast) {

break;

}

}

fast = start;

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

slow = slow->next;

fast = fast->next;

}

slow->next = NULL;

}

void display()

{

struct node \*tmp;

tmp = start;

while (tmp != NULL) {

printf(" %d ", tmp->num);

tmp = tmp->next;

}

printf("\n");

}

* **Addition**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node\* prev;

struct node\* next;

};

struct node\* create(struct node\*, int);

void display(struct node\*);

struct node\* add(struct node\*, struct node\*);

int main()

{

struct node\* head1 = NULL;

struct node\* head2 = NULL;

struct node\* head3 = NULL;

int num;

int i;

int data;

printf("Enter the number of nodes in link 1: ");

scanf("%d", &num);

for (i = 0; i < num; i++) {

printf("Enter the data:\n");

scanf("%d", &data);

head1 = create(head1, data);

}

printf("\nEnter the number of nodes in link 2: ");

scanf("%d", &num);

for (i = 0; i < num; i++) {

printf("Enter the data:\n");

scanf("%d", &data);

head2 = create(head2, data);

}

printf("\nLinked list 1:\n");

display(head1);

printf("\nLinked list 2:\n");

display(head2);

head3 = add(head1, head2);

printf("\nAfter addition:\n");

display(head3);

}

struct node\* create(struct node\* head, int data)

{

struct node\* tmp;

struct node\* q;

tmp = (struct node\*) malloc (sizeof(struct node));

tmp->data = data;

tmp->prev = NULL;

tmp->next = NULL;

if (head == NULL) {

head = tmp;

} else {

q = head;

while (q->next != NULL) {

q = q->next;

}

q->next = tmp;

tmp->prev = q;

}

return head;

}

void display(struct node\* tmp)

{

while (tmp != NULL) {

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

tmp = tmp->next;

}

printf("\n");

}

struct node\* add(struct node\* head1, struct node\* head2)

{

struct node\* tmp1 = head1;

struct node\* tmp2 = head2;

struct node\* head3 = NULL;

struct node\* tmp = NULL;

struct node\* q = NULL;

int data;

int carry = 0;

while (tmp1->next != NULL) {

tmp1 = tmp1->next;

}

while (tmp2->next != NULL) {

tmp2 = tmp2->next;

}

while (tmp1 != NULL || tmp2 != NULL) {

tmp = (struct node\*) malloc (sizeof(struct node));

if (tmp1 == NULL) {

data = tmp2->data + carry;

} else if (tmp2 == NULL) {

data = tmp1->data + carry;

} else {

data = tmp1->data + tmp2->data + carry;

}

carry = 0;

if (data > 9) {

tmp->data = data % 10;

carry = 1;

} else {

tmp->data = data;

}

tmp->next = NULL;

tmp->prev = NULL;

if (head3 == NULL) {

head3 = tmp;

} else {

q = head3;

while (q->next != NULL) {

q = q->next;

}

q->next = tmp;

tmp->prev = q;

}

if (tmp1 != NULL) {

tmp1 = tmp1->prev;

}

if (tmp2 != NULL) {

tmp2 = tmp2->prev;

}

}

if (carry == 1) {

tmp = (struct node\*) malloc (sizeof(struct node));

tmp->data = 1;

tmp->next = NULL;

tmp->prev = NULL;

q = head3;

while (q->next != NULL) {

q = q->next;

}

q->next = tmp;

tmp->prev = q;

}

return head3;

}