**A. Singly Circular and Doubly Circular Linked Lists**

**1. Implement a Menu Driven Singly Circular Linked List in C and C++ with all essential**

**operations on it.**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*tail = NULL;

struct node \*CreateNode(int val)

{

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

n->data = val;

n->next = NULL;

return n;

}

void InsertAtBeg(int val)

{

struct node \*n = CreateNode(val);

if (!tail)

{

tail = n;

n->next = n;

}

else

{

n->next = tail->next;

tail->next = n;

}

printf("\nInserted at beginning.");

}

void InsertAtEnd(int val)

{

struct node \*n = CreateNode(val);

if (!tail)

{

tail = n;

n->next = n;

}

else

{

n->next = tail->next;

tail->next = n;

tail = n;

}

printf("\nInserted at end.");

}

void InsertAfter(int val, int after)

{

if (!tail)

{

InsertAtBeg(val);

return;

}

struct node \*cur = tail->next;

do

{

if (cur->data == after)

{

struct node \*n = CreateNode(val);

n->next = cur->next;

cur->next = n;

if (cur == tail)

tail = n;

printf("\nInserted after %d.", after);

return;

}

cur = cur->next;

} while (cur != tail->next);

printf("\nValue %d not found.", after);

}

void DeleteFromBeg()

{

if (!tail)

{

printf("\nEmpty.");

return;

}

struct node \*head = tail->next;

if (head == tail)

{

free(head);

tail = NULL;

}

else

{

tail->next = head->next;

free(head);

}

printf("\nDeleted from beginning.");

}

void DeleteFromEnd()

{

if (!tail)

{

printf("\nEmpty.");

return;

}

struct node \*head = tail->next;

if (head == tail)

{

free(tail);

tail = NULL;

}

else

{

struct node \*cur = head;

while (cur->next != tail)

cur = cur->next;

cur->next = head;

free(tail);

tail = cur;

}

printf("\nDeleted from end.");

}

void Display()

{

if (!tail)

{

printf("\nEmpty.");

return;

}

struct node \*cur = tail->next;

printf("\nList: ");

do

{

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

cur = cur->next;

} while (cur != tail->next);

printf("(back to head)");

}

int main()

{

int ch = 0, v, key;

while (ch != 0)

{

printf("\n\n1.InBeg \n2.InEnd \n3.InAfter \n4.DelBeg \n5.DelEnd \n6.Display \n0.Exit \nChoice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Val: ");

scanf("%d", &v);

InsertAtBeg(v);

break;

case 2:

printf("Val: ");

scanf("%d", &v);

InsertAtEnd(v);

break;

case 3:

printf("Val: ");

scanf("%d", &v);

printf("After: ");

scanf("%d", &key);

InsertAfter(v, key);

break;

case 4:

DeleteFromBeg();

break;

case 5:

DeleteFromEnd();

break;

case 6:

Display();

break;

case 0:

printf("\nExit.");

break;

default:

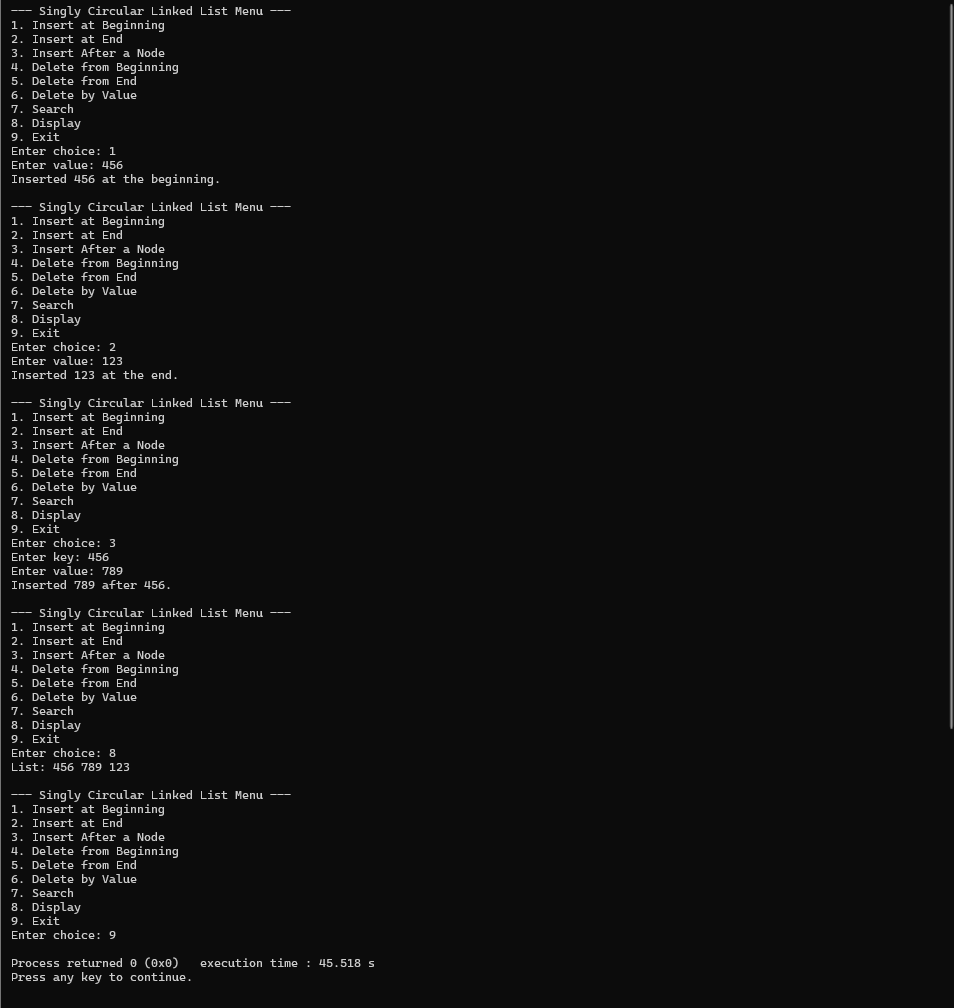
printf("\nInvalid Choice !!");

}

}

return 0;

}



**2. Implement a Menu Driven Doubly Circular Linked List in C and C++ with all essential**

**operations on it.**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*next;

struct node \*prev;

};

struct node \*head = NULL;

struct node \*CreateNode(int val)

{

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

n->data = val;

n->next = n->prev = NULL;

return n;

}

void InsertAtBeg(int val)

{

struct node \*n = CreateNode(val);

if (!head)

{

head = n;

n->next = n;

n->prev = n;

}

else

{

struct node \*tail = head->prev;

n->next = head;

n->prev = tail;

tail->next = n;

head->prev = n;

head = n;

}

printf("\nInserted at beginning.");

}

void InsertAtEnd(int val)

{

if (!head)

{

InsertAtBeg(val);

return;

}

struct node \*n = CreateNode(val);

struct node \*tail = head->prev;

tail->next = n;

n->prev = tail;

n->next = head;

head->prev = n;

printf("\nInserted at end.");

}

void DeleteFromBeg()

{

if (!head)

{

printf("\nEmpty.");

return;

}

if (head->next == head)

{

free(head);

head = NULL;

}

else

{

struct node \*tail = head->prev;

struct node \*temp = head;

head = head->next;

tail->next = head;

head->prev = tail;

free(temp);

}

printf("\nDeleted from beginning.");

}

void DeleteFromEnd()

{

if (!head)

{

printf("\nEmpty.");

return;

}

if (head->next == head)

{

free(head);

head = NULL;

}

else

{

struct node \*tail = head->prev;

struct node \*newTail = tail->prev;

newTail->next = head;

head->prev = newTail;

free(tail);

}

printf("\nDeleted from end.");

}

void DisplayForward()

{

if (!head)

{

printf("\nEmpty.");

return;

}

struct node \*cur = head;

printf("\nForward: ");

do

{

printf("%d <-> ", cur->data);

cur = cur->next;

} while (cur != head);

printf("(back)");

}

void DisplayBackward()

{

if (!head)

{

printf("\nEmpty.");

return;

}

struct node \*tail = head->prev;

struct node \*cur = tail;

printf("\nBackward: ");

do

{

printf("%d <-> ", cur->data);

cur = cur->prev;

} while (cur != tail);

printf("(back)");

}

int main()

{

int ch = 0, v;

while (ch != 0)

{

printf("\n\n1.InBeg \n2.InEnd \n3.DelBeg \n4.DelEnd \n5.DisplayF \n6.DisplayB \n0.Exit\nChoice: ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Val: ");

scanf("%d", &v);

InsertAtBeg(v);

break;

case 2:

printf("Val: ");

scanf("%d", &v);

InsertAtEnd(v);

break;

case 3:

DeleteFromBeg();

break;

case 4:

DeleteFromEnd();

break;

case 5:

DisplayForward();

break;

case 6:

DisplayBackward();

break;

case 0:

printf("\nExit.");

break;

default:

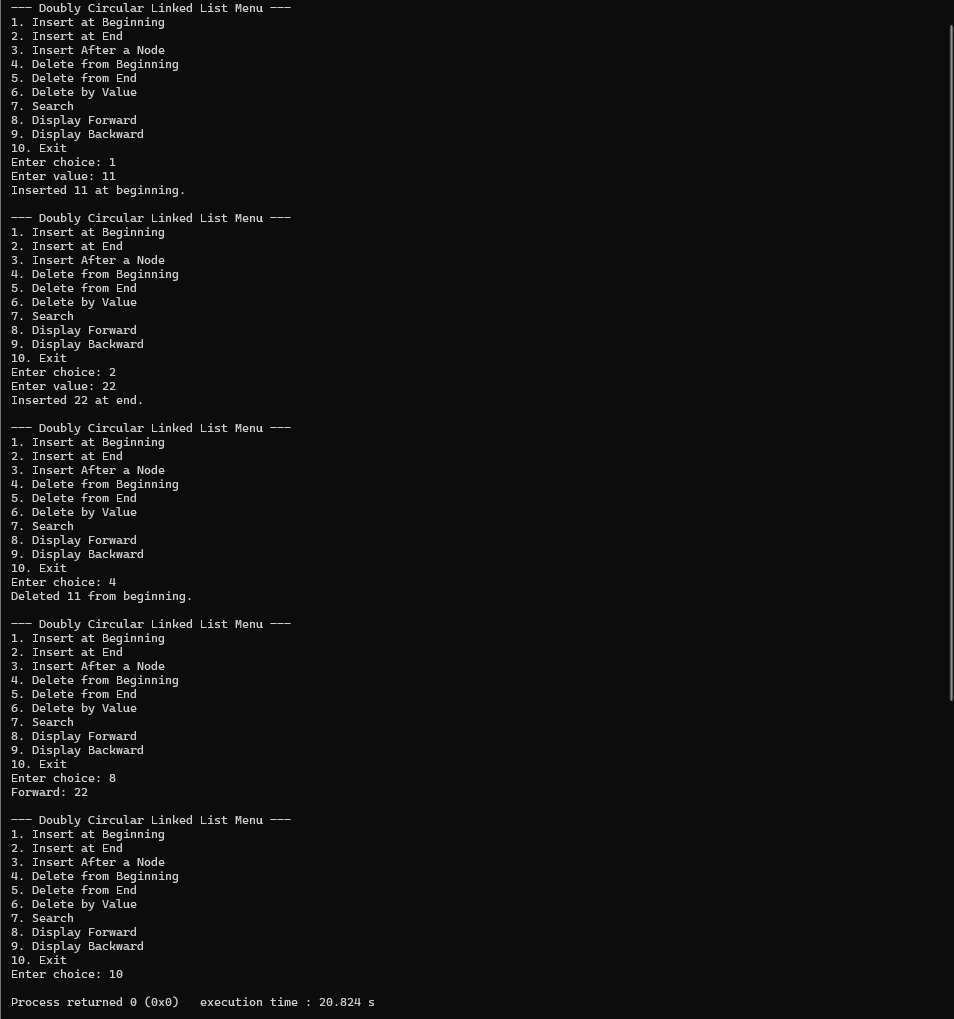
printf("\nInvalid.");

}

}

return 0;

}



**B. Applications of various Linked Lists**

**1. Polynomial Representation and Evaluation in C/C++**

**Use a Singly Linked List to represent and perform operations on polynomials.**

**1. Implement functions to create polynomials and evaluate the value of a polynomial at**

**the user's value.**

**2. Implement a function to add two polynomials**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

// Structure for a term in the polynomial

struct Node {

int coeff; // Coefficient

int pow; // Power

struct Node \*next;

};

// Function to create a new node

struct Node\* createNode(int coeff, int pow) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->coeff = coeff;

newNode->pow = pow;

newNode->next = NULL;

return newNode;

}

// Function to insert a term at the end

void insertTerm(struct Node\*\* poly, int coeff, int pow) {

struct Node\* newNode = createNode(coeff, pow);

if (\*poly == NULL) {

\*poly = newNode;

} else {

struct Node\* temp = \*poly;

while (temp->next != NULL)

temp = temp->next;

temp->next = newNode;

}

}

// Function to create a polynomial

void createPolynomial(struct Node\*\* poly) {

int n, coeff, pow;

printf("Enter number of terms: ");

scanf("%d", &n);

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

printf("Enter coefficient and power: ");

scanf("%d %d", &coeff, &pow);

insertTerm(poly, coeff, pow);

}

}

// Function to display polynomial

void displayPolynomial(struct Node\* poly) {

if (poly == NULL) {

printf("0");

return;

}

while (poly != NULL) {

printf("%d\*x^%d", poly->coeff, poly->pow);

poly = poly->next;

if (poly != NULL)

printf(" + ");

}

printf("\n");

}

// Function to evaluate polynomial for given x

double evaluatePolynomial(struct Node\* poly, double x) {

double result = 0.0;

while (poly != NULL) {

result += poly->coeff \* pow(x, poly->pow);

poly = poly->next;

}

return result;

}

// Function to add two polynomials

struct Node\* addPolynomials(struct Node\* poly1, struct Node\* poly2) {

struct Node\* result = NULL;

struct Node\* p1 = poly1;

struct Node\* p2 = poly2;

while (p1 != NULL && p2 != NULL) {

if (p1->pow > p2->pow) {

insertTerm(&result, p1->coeff, p1->pow);

p1 = p1->next;

}

else if (p1->pow < p2->pow) {

insertTerm(&result, p2->coeff, p2->pow);

p2 = p2->next;

}

else {

insertTerm(&result, p1->coeff + p2->coeff, p1->pow);

p1 = p1->next;

p2 = p2->next;

}

}

// Add remaining terms

while (p1 != NULL) {

insertTerm(&result, p1->coeff, p1->pow);

p1 = p1->next;

}

while (p2 != NULL) {

insertTerm(&result, p2->coeff, p2->pow);

p2 = p2->next;

}

return result;

}

// Main function

int main() {

struct Node\* poly1 = NULL;

struct Node\* poly2 = NULL;

struct Node\* sum = NULL;

double x;

printf("Create first polynomial:\n");

createPolynomial(&poly1);

printf("First Polynomial: ");

displayPolynomial(poly1);

printf("\nCreate second polynomial:\n");

createPolynomial(&poly2);

printf("Second Polynomial: ");

displayPolynomial(poly2);

// Add polynomials

sum = addPolynomials(poly1, poly2);

printf("\nSum of Polynomials: ");

displayPolynomial(sum);

// Evaluate

printf("\nEnter value of x to evaluate first polynomial: ");

scanf("%lf", &x);

printf("Value of first polynomial at x=%.2lf: %.2lf\n", x, evaluatePolynomial(poly1, x));

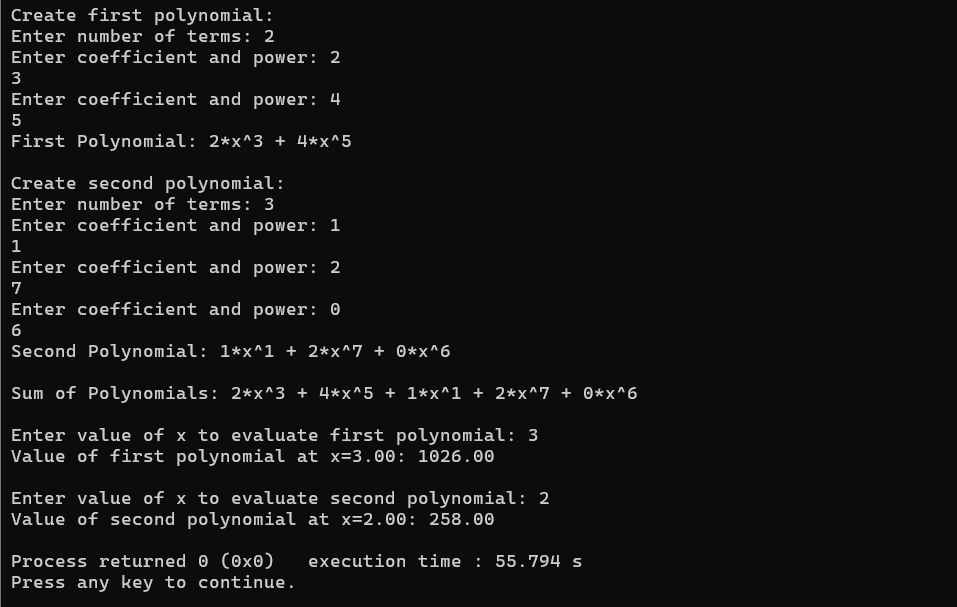
printf("\nEnter value of x to evaluate second polynomial: ");

scanf("%lf", &x);

printf("Value of second polynomial at x=%.2lf: %.2lf\n", x, evaluatePolynomial(poly2, x));

return 0;

}



**2. Music Playlist in C/C++**

**Use a Doubly Linked List to implement a music playlist with the following features:**

**1. Add a song to the playlist**

**2. Remove a song from the playlist**

**3. Play the next song**

**4. Play the previous song**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Node structure for doubly linked list

typedef struct Song

{

char name[100];

struct Song \*prev;

struct Song \*next;

} Song;

Song \*head = NULL; // start of playlist

Song \*tail = NULL; // end of playlist

Song \*current = NULL; // currently playing song

// Function to create a new song node

Song \*createSong(char \*name)

{

Song \*newSong = (Song \*)malloc(sizeof(Song));

strcpy(newSong->name, name);

newSong->prev = newSong->next = NULL;

return newSong;

}

// Add a song to the end of the playlist

void addSong(char \*name)

{

Song \*newSong = createSong(name);

if (head == NULL)

{

head = tail = current = newSong;

}

else

{

tail->next = newSong;

newSong->prev = tail;

tail = newSong;

}

printf("Song '%s' added to playlist.\n", name);

}

// Remove a song by name

void removeSong(char \*name)

{

Song \*temp = head;

while (temp != NULL)

{

if (strcmp(temp->name, name) == 0)

{

if (temp == head)

head = temp->next;

if (temp == tail)

tail = temp->prev;

if (temp->prev)

temp->prev->next = temp->next;

if (temp->next)

temp->next->prev = temp->prev;

if (current == temp)

current = temp->next ? temp->next : temp->prev;

free(temp);

printf("Song '%s' removed from playlist.\n", name);

return;

}

temp = temp->next;

}

printf("Song '%s' not found in playlist.\n", name);

}

// Play next song

void playNext()

{

if (current && current->next)

{

current = current->next;

printf("Playing next song: %s\n", current->name);

}

else

{

printf("You are at the last song. No next song available.\n");

}

}

// Play previous song

void playPrev()

{

if (current && current->prev)

{

current = current->prev;

printf("Playing previous song: %s\n", current->name);

}

else

{

printf("You are at the first song. No previous song available.\n");

}

}

// Display the playlist

void displayPlaylist()

{

Song \*temp = head;

if (!temp)

{

printf("Playlist is empty.\n");

return;

}

printf("\nPlaylist:\n");

while (temp != NULL)

{

if (temp == current)

printf("-> %s (Currently Playing)\n", temp->name);

else

printf(" %s\n", temp->name);

temp = temp->next;

}

}

int main()

{

int choice;

char songName[100];

do

{

printf("\n=== Music Playlist Menu ===\n");

printf("1. Add Song\n");

printf("2. Remove Song\n");

printf("3. Play Next Song\n");

printf("4. Play Previous Song\n");

printf("5. Display Playlist\n");

printf("6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

getchar(); // to consume newline left by scanf

switch (choice)

{

case 1:

printf("Enter song name to add: ");

fgets(songName, sizeof(songName), stdin);

songName[strcspn(songName, "\n")] = 0; // remove newline

addSong(songName);

break;

case 2:

printf("Enter song name to remove: ");

fgets(songName, sizeof(songName), stdin);

songName[strcspn(songName, "\n")] = 0;

removeSong(songName);

break;

case 3:

playNext();

break;

case 4:

playPrev();

break;

case 5:

displayPlaylist();

break;

case 6:

printf("Exiting program...\n");

break;

default:

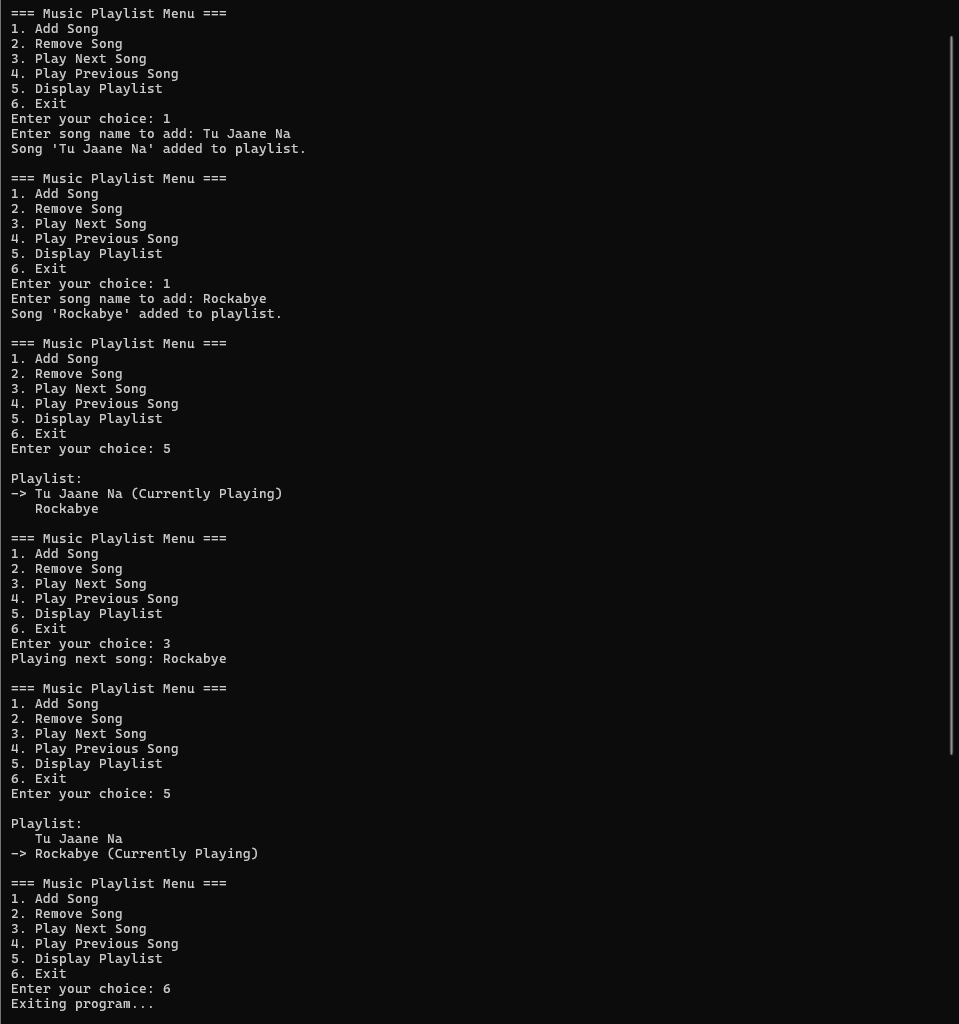
printf("Invalid choice! Try again.\n");

}

} while (choice != 6);

return 0;

}



3.**Round Robin Scheduling in C/C++**

**Use a Circular Linked List to implement a simple Round Robin scheduling algorithm:**

**1. Add processes to the list**

**2. Execute each process for a given time quantum**

**3. Move the executed process to the end of the list**

**4. Remove completed processes**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct Process {

int id;

int burst\_time;

struct Process \*next;

} Process;

Process \*head = NULL;

// Function to create a new process node

Process\* createProcess(int id, int burst\_time) {

Process \*newNode = (Process\*)malloc(sizeof(Process));

newNode->id = id;

newNode->burst\_time = burst\_time;

newNode->next = NULL;

return newNode;

}

// Add process to circular linked list

void addProcess(int id, int burst\_time) {

Process \*newNode = createProcess(id, burst\_time);

if (head == NULL) {

head = newNode;

head->next = head; // circular

} else {

Process \*temp = head;

while (temp->next != head) {

temp = temp->next;

}

temp->next = newNode;

newNode->next = head;

}

printf("Process P%d added with burst time %d\n", id, burst\_time);

}

// Execute Round Robin

void executeProcesses(int quantum) {

if (head == NULL) {

printf("No processes to execute.\n");

return;

}

Process \*curr = head;

Process \*prev = NULL;

while (head != NULL) {

printf("\nExecuting Process P%d (Remaining Burst Time: %d)\n", curr->id, curr->burst\_time);

if (curr->burst\_time > quantum) {

curr->burst\_time -= quantum;

printf("Process P%d executed for %d units, Remaining: %d\n", curr->id, quantum, curr->burst\_time);

prev = curr;

curr = curr->next; // Move to next process

} else {

printf("Process P%d executed for %d units and completed.\n", curr->id, curr->burst\_time);

if (curr == curr->next) { // Only one process left

free(curr);

head = NULL;

break;

}

// Remove the completed process

Process \*temp = curr;

if (prev == NULL) { // removing head

Process \*last = head;

while (last->next != head) {

last = last->next;

}

head = head->next;

last->next = head;

curr = head;

} else {

prev->next = curr->next;

curr = curr->next;

}

free(temp);

}

}

printf("\nAll processes completed.\n");

}

int main() {

int n, i, burst, quantum;

printf("Enter number of processes: ");

scanf("%d", &n);

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

printf("Enter burst time for Process P%d: ", i);

scanf("%d", &burst);

addProcess(i, burst);

}

printf("\nEnter Time Quantum: ");

scanf("%d", &quantum);

executeProcesses(quantum);

return 0;

}

