# ASSIGNMENT : 2 (LINKED LIST)

Q1.

Code: #include<stdio.h>

#include<stdlib.h>

typedef struct poly {

    int coeff;

    int expo;

    struct poly \*next;

} poly;

poly \*create() {

    poly \*head = NULL, \*temp = NULL, \*newNode;

    int ch;

    while (1) {

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

        if (newNode == NULL) {

            printf("Memory allocation failed.\n");

            break;

        }

        printf("Enter the coefficient for node: ");

        scanf("%d", &newNode->coeff);

        printf("Enter the exponent for node: ");

        scanf("%d", &newNode->expo);

        newNode->next = NULL;

        if (head == NULL) {

            head = newNode;

        } else {

            temp->next = newNode;

        }

        temp = newNode;

        printf("Do you want to add another term? (1 for yes, 0 for no): ");

        scanf("%d", &ch);s

        if (ch == 0) {

            break;

        }

    }

    return head;

}

poly \*addpoly(poly \*p1, poly \*p2) {

    poly \*p3 = NULL, \*temp = NULL, \*newNode;

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

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

        if (newNode == NULL) {

            printf("Memory allocation failed.\n");

            break;

        }

        newNode->next = NULL;

        if (p1->expo == p2->expo) {

            newNode->coeff = p1->coeff + p2->coeff;

            newNode->expo = p1->expo;

            p1 = p1->next;

            p2 = p2->next;

        } else if (p1->expo > p2->expo) {

            newNode->coeff = p1->coeff;

            newNode->expo = p1->expo;

            p1 = p1->next;

        } else {

            newNode->coeff = p2->coeff;

            newNode->expo = p2->expo;

            p2 = p2->next;

        }

        if (p3 == NULL) {

            p3 = newNode;

        } else {

            temp->next = newNode;

        }

        temp = newNode;

    }

    while (p1 != NULL) {

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

        if (newNode == NULL) {

            printf("Memory allocation failed.\n");

            break;

        }

        newNode->coeff = p1->coeff;

        newNode->expo = p1->expo;

        newNode->next = NULL;

        if (p3 == NULL) {

            p3 = newNode;

        } else {

            temp->next = newNode;

        }

        temp = newNode;

        p1 = p1->next;

    }

    while (p2 != NULL) {

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

        if (newNode == NULL) {

            printf("Memory allocation failed.\n");

            break;

        }

        newNode->coeff = p2->coeff;

        newNode->expo = p2->expo;

        newNode->next = NULL;

        if (p3 == NULL) {

            p3 = newNode;

        } else {

            temp->next = newNode;

        }

        temp = newNode;

        p2 = p2->next;

    }

    return p3;

}

void display(poly \*p) {

    if (p == NULL) {

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

        return;

    }

    while (p != NULL) {

        printf("%dx^%d", p->coeff, p->expo);

        p = p->next;

        if (p != NULL) {

            printf(" + ");

        }

    }

    printf("\n");

}

int main() {

    poly \*p1 = NULL, \*p2 = NULL, \*p3 = NULL;

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

    p1 = create();

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

    p2 = create();

    printf("Adding polynomials...\n");

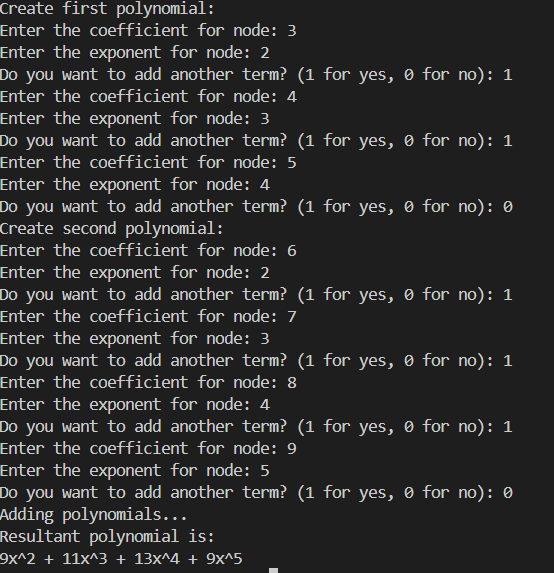
    p3 = addpoly(p1, p2);

    printf("Resultant polynomial is:\n");

    display(p3);

    return 0;

}



Q2 :

Code :

#include <stdio.h>

#include <stdlib.h>

typedef struct process {

int id;

int burst\_time;

struct process\* next;

} process;

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

process\* newProcess = (process\*)malloc(sizeof(process));

newProcess->id = id;

newProcess->burst\_time = burst\_time;

newProcess->next = NULL;

return newProcess;

}

void appendProcess(process\*\* head, int id, int burst\_time) {

process\* newProcess = createProcess(id, burst\_time);

if (\*head == NULL) {

\*head = newProcess;

newProcess->next = \*head;

} else {

process\* temp = \*head;

while (temp->next != \*head) {

temp = temp->next;

}

temp->next = newProcess;

newProcess->next = \*head;

}

}

void timeSharing(process\* head, int time\_slot, int total\_processes) {

process\* current = head;

int completedProcesses = 0;

int total\_time = 0;

while (completedProcesses < total\_processes) {

if (current->burst\_time > 0) {

if (current->burst\_time <= time\_slot) {

total\_time += current->burst\_time;

printf("Process %d completed in %d ns\n", current->id, total\_time);

current->burst\_time = 0;

completedProcesses++;

} else {

current->burst\_time -= time\_slot;

total\_time += time\_slot;

}

}

current = current->next;

}

}

int main() {

process\* head = NULL;

int N, i, burst\_time;

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

scanf("%d", &N);

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

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

scanf("%d", &burst\_time);

appendProcess(&head, i, burst\_time);

}

int time\_slot;

printf("Enter the time slot for each process (in ns): ");

scanf("%d", &time\_slot);

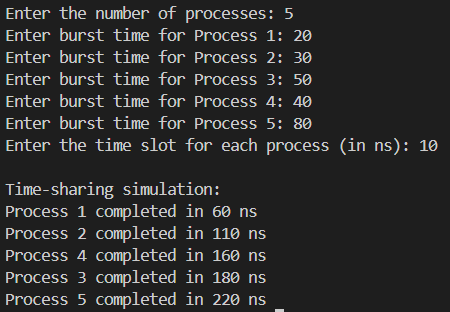
printf("\nTime-sharing simulation:\n");

timeSharing(head, time\_slot, N);

return 0;

}

Output :



Q3:

Code:

#include <stdio.h>

#include <stdlib.h>

typedef struct node {

int data;

struct node\* next;

} node;

node\* createNode(int data) {

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

newNode->data = data;

newNode->next = NULL;

return newNode;

}

void appendNode(node\*\* head, int data) {

node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

} else {

node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

printf("Node with data %d appended.\n", data);

}

void printList(node\* head) {

node\* temp = head;

if (head == NULL) {

printf("The list is empty.\n");

return;

}

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

void reverseList(node\*\* head) {

node\* prev = NULL;

node\* curr = \*head;

node\* next = NULL;

while (curr != NULL) {

next = curr->next;

curr->next = prev;

prev = curr;

curr = next;

}

\*head = prev;

printf("The list has been reversed.\n");

}

int main() {

node\* head = NULL;

int choice, data;

do {

printf("\nMenu:\n");

printf("1. Append Node\n");

printf("2. Print List\n");

printf("3. Reverse List\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to append: ");

scanf("%d", &data);

appendNode(&head, data);

break;

case 2:

printf("Current Linked List:\n");

printList(head);

break;

case 3:

reverseList(&head);

break;

case 4:

printf("Exiting program.\n");

break;

default:

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

}

} while (choice != 4);

return 0;

}

Output:

A screenshot of a computer program

Description automatically generated