Q1. Write a C program to create two sorted single linked list and then merge them.

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*insert(struct node \*head, int x)

{

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

nn->data = x;

nn->next = NULL;

if (head == NULL)

{

return nn;

}

struct node \*curr = head;

while (curr->next != NULL)

{

curr = curr->next;

}

curr->next = nn;

return head;

}

void disp(struct node \*head)

{

struct node \*curr = head;

while (curr != NULL)

{

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

curr = curr->next;

}

printf("\n");

}

struct node \*merge(struct node \*l1, struct node \*l2)

{

if (l1 == NULL)

return l2;

if (l2 == NULL)

return l1;

struct node \*NewNode = NULL;

if (l1->data <= l2->data)

{

NewNode = l1;

NewNode->next = merge(l1->next, l2);

}

else

{

NewNode = l2;

NewNode->next = merge(l1, l2->next);

}

return NewNode;

}

int main()

{

struct node \*l1 = NULL;

struct node \*l2 = NULL;

struct node \*nn = NULL;

int ch, x;

printf("\n\nKrishant Chauhan MCA 2A Roll no-32 \n");

printf("1. Insert into List 1\n");

printf("2. Insert into List 2\n");

printf("3. Merge Lists\n");

printf("4. Display Merged List\n");

do

{

printf("\nEnter your choice: ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter the x to insert into List 1: ");

scanf("%d", &x);

l1 = insert(l1, x);

break;

case 2:

printf("Enter the x to insert into List 2: ");

scanf("%d", &x);

l2 = insert(l2, x);

break;

case 3:

nn = merge(l1, l2);

printf("Lists merged successfully!\n");

break;

case 4:

printf("Merged List: ");

if (nn == NULL)

printf("Please Merge them\n");

disp(nn);

break;

default:

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

break;

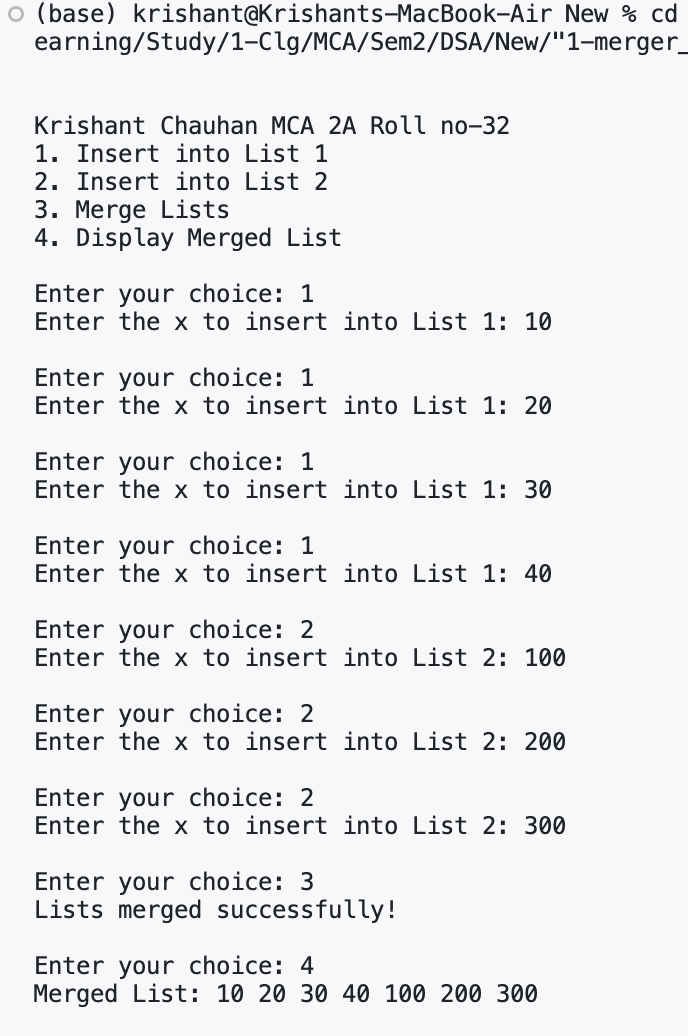
}

} while (ch != 5);

return 0;

}

OUTPUT :-



Q2. Write a C program to create a single linked list ,then write another function InsertNth() which can insert a new node after any node given by the user in that linked list.

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*append(struct node \*head, int x)

{

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

nn->data = x;

nn->next = NULL;

if (head == NULL)

{

return nn;

}

struct node \*curr = head;

while (curr->next != NULL)

{

curr = curr->next;

}

curr->next = nn;

return head;

}

void display(struct node \*head)

{

struct node \*current = head;

while (current != NULL)

{

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

current = current->next;

}

printf("\n");

}

struct node \*insert(struct node \*head, int val, int pos)

{

struct node \*temp, \*newnode;

int i = 1;

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

newnode->data = val;

if (pos == 1)

{

newnode->next = head;

head = newnode;

}

else

{

temp = head;

while (i < pos - 1 && temp != NULL)

{

temp = temp->next;

i++;

}

if (temp == NULL)

{

printf("Invalid position\n");

return head;

}

newnode->next = temp->next;

temp->next = newnode;

}

printf("Value Inserted at %dth Position\n", pos);

return head;

}

int main()

{

struct node \*head = NULL;

int ch, x, pos;

printf("\n\nKrishant Chauhan MCA 2A Roll no-32 \n");

printf("1. Append\n");

printf("2. Insert\n");

printf("3. Display\n");

printf("4. Exit\n");

do

{

printf("Enter your choice: ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter the value to append: ");

scanf("%d", &x);

head = append(head, x);

break;

case 2:

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

scanf("%d", &x);

printf("Enter the position ");

scanf("%d", &pos);

head = insert(head, x, pos);

break;

case 3:

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

display(head);

break;

case 4:

printf("Exit\n");

break;

default:

printf("Invalid choice!\n");

break;

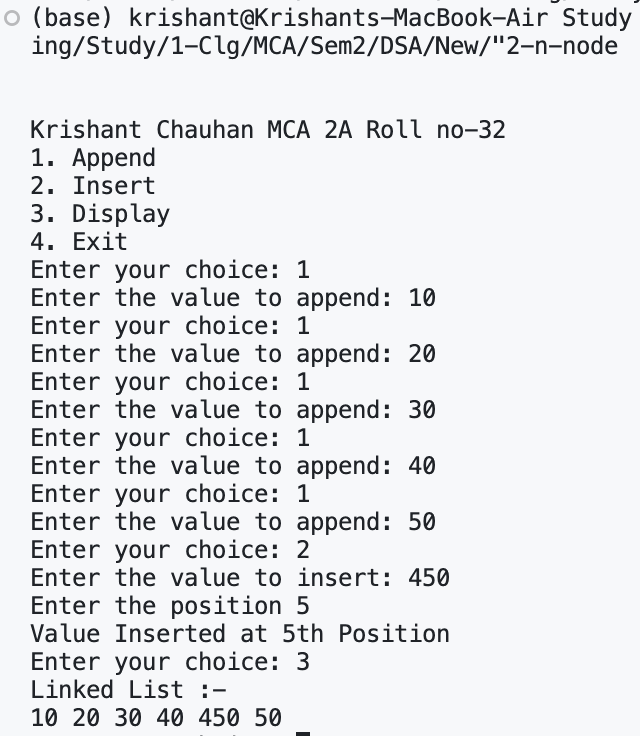
}

} while (ch != 4);

return 0;

}

OUTPUT :-



Q3. Write a C program to create single linked then remove duplicate nodes in unsorted linked list.

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*insert(struct node \*head, int x)

{

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

nn->data = x;

nn->next = NULL;

if (head == NULL)

{

return nn;

}

struct node \*curr = head;

while (curr->next != NULL)

{

curr = curr->next;

}

curr->next = nn;

return head;

}

void disp(struct node \*head)

{

struct node \*curr = head;

while (curr != NULL)

{

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

curr = curr->next;

}

printf("\n");

}

void rem(struct node \*head)

{

struct node \*current = head;

struct node \*runner;

struct node \*duplicate;

while (current != NULL && current->next != NULL)

{

runner = current;

while (runner->next != NULL)

{

if (current->data == runner->next->data)

{

duplicate = runner->next;

runner->next = runner->next->next;

free(duplicate);

}

else

runner = runner->next;

}

current = current->next;

}

}

int main()

{

struct node \*head = NULL;

int choice, value;

printf("\n\nKrishant Chauhan MCA 2A Roll no-32 \n");

printf("1. Insert a node\n");

printf("2. Remove duplicate nodes\n");

printf("3. Display \n");

printf("4. Exit\n");

do

{

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice)

{

case 1:

printf("Enter the value: ");

scanf("%d", &value);

head = insert(head, value);

break;

case 2:

rem(head);

printf("Duplicate nodes removed \n");

break;

case 3:

printf("List: ");

disp(head);

break;

case 4:

printf("Exit\n");

break;

default:

printf("Invalid choice!\n");

break;

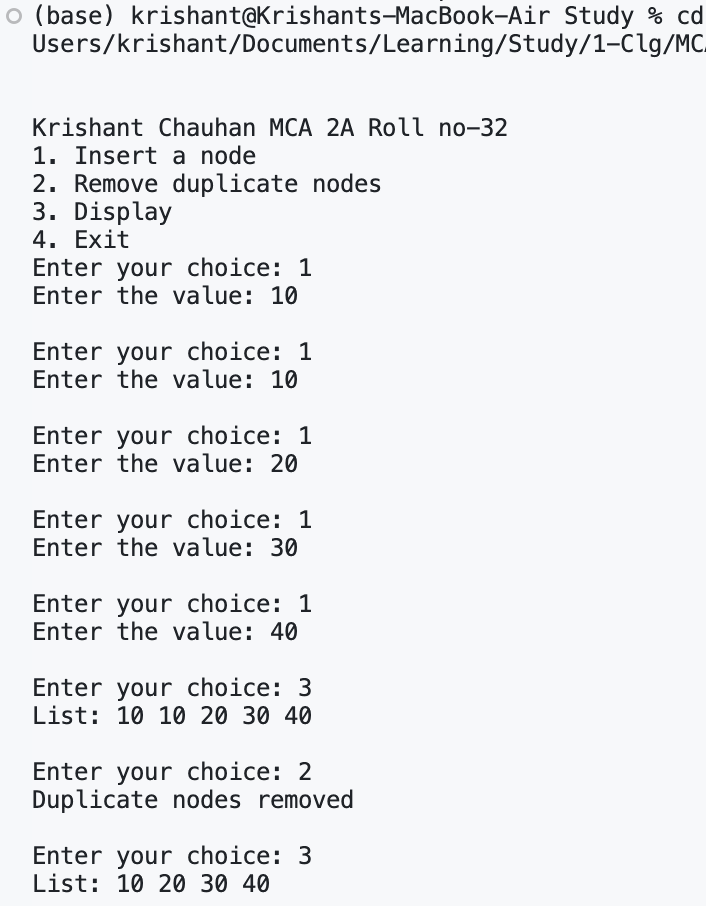
}

printf("\n");

} while (choice != 4);

return 0;

}

OUTPUT :-

Q4. Write a C program to   create a  linked list  P, then  write a ‘C’ function named **split** to create two linked lists Q &  R from P So that  Q contains all elements in odd positions of P and R contains the remaining  elements. Finally print both linked lists i.e. Q and R.

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*append(struct node \*head, int x)

{

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

nn->data = x;

nn->next = NULL;

if (head == NULL)

{

return nn;

}

struct node \*curr = head;

while (curr->next != NULL)

{

curr = curr->next;

}

curr->next = nn;

return head;

}

void display(struct node \*head)

{

struct node \*current = head;

while (current != NULL)

{

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

current = current->next;

}

printf("\n");

}

void insert(struct node \*head, struct node \*\*Q, struct node \*\*R)

{

struct node \*curr = head;

struct node \*qhead = NULL;

struct node \*rhead = NULL;

int flag = 1;

while (curr != NULL)

{

if (flag == 1)

{

if (\*Q == NULL)

{

\*Q = curr;

qhead = \*Q;

curr = curr->next;

}

else

{

(\*Q)->next = curr;

(\*Q) = (\*Q)->next;

curr = curr->next;

}

flag = 0;

}

else if (flag == 0)

{

if (\*R == NULL)

{

\*R = curr;

rhead = \*R;

curr = curr->next;

}

else

{

(\*R)->next = curr;

(\*R) = (\*R)->next;

curr = curr->next;

}

flag = 1;

}

}

if (\*Q != NULL)

{

(\*Q)->next = NULL;

}

if (\*R != NULL)

{

(\*R)->next = NULL;

}

\*Q = qhead;

\*R = rhead;

}

int main()

{

struct node \*head = NULL, \*Q = NULL, \*R = NULL;

int ch, x, pos;

printf("\n\nKrishant Chauhan MCA 2A Roll no-32 \n");

printf("1. Append\n");

printf("2. Split\n");

printf("3. Display\n");

printf("4. Exit\n");

do

{

printf("Enter your choice: ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter the value to append: ");

scanf("%d", &x);

head = append(head, x);

break;

case 2:

insert(head, &Q, &R);

break;

case 3:

printf("Odd Position\n");

display(Q);

printf("Even Position\n");

display(R);

break;

case 4:

printf("Exit\n");

break;

default:

printf("Invalid choice!\n");

break;

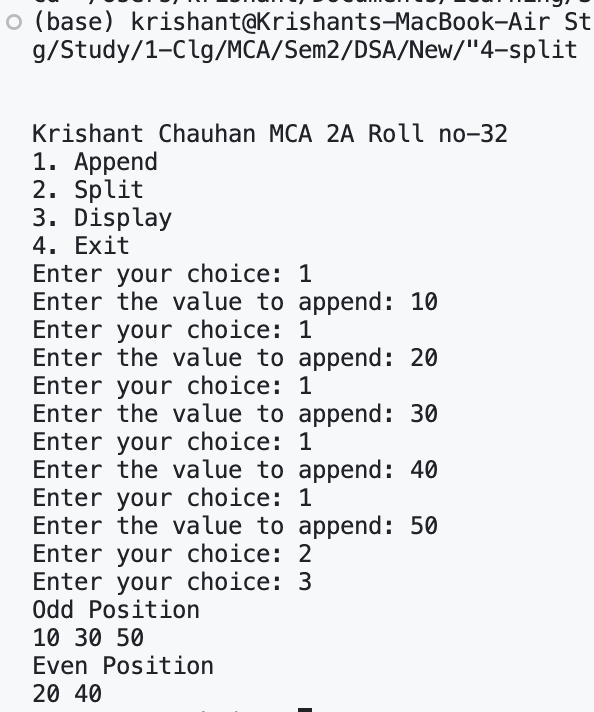
}

} while (ch != 4);

return 0;

}

**OUTPUT :-**



Q5:- W.A.P. to create a binary search tree and perform following operations:

       1) Search a particular key.

       2) Delete a node from the tree.

       3) Count total number of leaf nodes

       4) Count nodes having both children in the binary search tree

        5) Count total numbers of nodes from right left side of root node

#include <stdio.h>

#include <stdlib.h>

typedef struct tree

{

int data;

struct tree \*left, \*right;

} node;

node \*insert(node \*root, int x)

{

if (root == NULL)

{

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

nn->left = nn->right = NULL;

nn->data = x;

return nn;

}

else if (root->data > x)

root->left = insert(root->left, x);

else if (root->data < x)

root->right = insert(root->right, x);

return root;

}

node \*search(node \*root, int key)

{

if (root == NULL || root->data == key)

return root;

if (root->data > key)

return search(root->left, key);

return search(root->right, key);

}

node \*min(node \*root)

{

if (root == NULL)

return NULL;

else if (root->left == NULL)

return root;

else

return min(root->left);

}

node \*deleteNode(node \*root, int key)

{

if (root == NULL)

return root;

if (key < root->data)

root->left = deleteNode(root->left, key);

else if (key > root->data)

root->right = deleteNode(root->right, key);

else

{

if (root->left == NULL)

{

node \*temp = root->right;

free(root);

return temp;

}

else if (root->right == NULL)

{

node \*temp = root->left;

free(root);

return temp;

}

node \*temp = min(root->right);

root->data = temp->data;

root->right = deleteNode(root->right, temp->data);

}

return root;

}

void leafNode(node \*root, int \*ct)

{

if (root != NULL)

{

if (root->left == NULL && root->right == NULL)

(\*ct)++;

leafNode(root->left, ct);

leafNode(root->right, ct);

}

}

void twoChildNode(node \*root, int \*ct)

{

if (root != NULL)

{

if (root->left != NULL && root->right != NULL)

(\*ct)++;

twoChildNode(root->left, ct);

twoChildNode(root->right, ct);

}

}

int countlft(node \*root)

{

if (root == NULL || (root->left == NULL && root->right == NULL))

return 0;

return 1 + countlft(root->left) + countlft(root->right);

}

int main()

{

node \*root = NULL;

int leaf = 0, twoChild = 0;

int choice, n, key;

printf("\n\nKrishant Chauhan MCA 2A Roll no-32 \n");

printf("Choice:\n 1-Insert\n 2-Search\n 3-Delete\n 4-Total Leaf Nodes\n 5-Ct Total Nodes with 2 Children\n 6-Count Nodes on Left Side\n 7-Exit\n\n");

do

{

printf("ENTER YOUR CHOICE : ");

scanf("%d", &choice);

switch (choice)

{

case 1:

{

printf("ENTER THE NODE VALUE: ");

scanf("%d", &n);

root = insert(root, n);

break;

}

case 2:

{

printf("Enter the value to search");

scanf("%d", &key);

node \*result = search(root, key);

if (result != NULL)

printf("Value %d found\n", key);

else

printf("Value %d not found \n", key);

break;

}

case 3:

{

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

scanf("%d", &key);

root = deleteNode(root, key);

printf("Node with value %d deleted \n", key);

break;

}

case 4:

{

leafNode(root, &leaf);

printf("Total No of Leaf Nodes are: %d\n", leaf);

leaf = 0;

break;

}

case 5:

{

twoChildNode(root, &twoChild);

printf("Total No of Nodes with 2 children: %d\n", twoChild);

twoChild = 0;

break;

}

case 6:

{

int leftCount = countlft(root);

printf("Total No of Nodes on the Left Side %d\n", leftCount);

break;

}

case 7:

{

printf("Exit\n");

break;

}

default:

{

printf("Invalid choice!\n");

break;

}

}

} while (choice != 7);

return 0;

}

Q6. Write a program to add of two polynomials of degree n, using linked list

#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

int cof;

int power;

struct node \*next;

} node;

void insert(node \*\*head, int cof, int power)

{

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

nn->cof = cof;

nn->power = power;

nn->next = NULL;

if (\*head == NULL)

{

\*head = nn;

}

else

{

node \*curr = \*head;

while (curr->next != NULL)

{

curr = curr->next;

}

curr->next = nn;

}

}

void display(node \*head)

{

node \*curr = head;

while (curr != NULL)

{

printf("%dx^%d", curr->cof, curr->power);

if (curr->next != NULL)

{

printf(" + ");

}

curr = curr->next;

}

printf("\n");

}

node \*add(node \*p1, node \*p2)

{

node \*result = NULL;

node \*curr = NULL;

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

{

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

nn->next = NULL;

if (p1->power == p2->power)

{

nn->cof = p1->cof + p2->cof;

nn->power = p1->power;

p1 = p1->next;

p2 = p2->next;

}

else if (p1->power > p2->power)

{

nn->cof = p1->cof;

nn->power = p1->power;

p1 = p1->next;

}

else

{

nn->cof = p2->cof;

nn->power = p2->power;

p2 = p2->next;

}

if (result == NULL)

{

result = nn;

curr = result;

}

else

{

curr->next = nn;

curr = curr->next;

}

}

while (p1 != NULL)

{

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

nn->cof = p1->cof;

nn->power = p1->power;

nn->next = NULL;

curr->next = nn;

curr = curr->next;

p1 = p1->next;

}

while (p2 != NULL)

{

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

nn->cof = p2->cof;

nn->power = p2->power;

nn->next = NULL;

curr->next = nn;

curr = curr->next;

p2 = p2->next;

}

return result;

}

int main()

{

node \*p1 = NULL;

node \*p2 = NULL;

int ch, cof, power;

printf("Menu\n");

printf("1. Insert into Equation 1\n");

printf("2. Insert into Equation 2\n");

printf("3. Add Equations\n");

printf("4. Exit\n");

do

{

printf("Enter your choice: ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter coefficient and Exponent of Equation 1: ");

scanf("%d%d", &cof, &power);

insert(&p1, cof, power);

break;

case 2:

printf("Enter coefficient and Exponent of Equation 2: ");

scanf("%d%d", &cof, &power);

insert(&p2, cof, power);

break;

case 3:

if (p1 == NULL || p2 == NULL)

{

printf("Please enter Equation 1 and Equation 2.\n");

break;

}

node \*p3 = add(p1, p2);

printf("\nThe Sum of Two Equations is:\n");

display(p3);

break;

case 4:

printf("Exit\n");

break;

default:

printf("Invalid choice!\n");

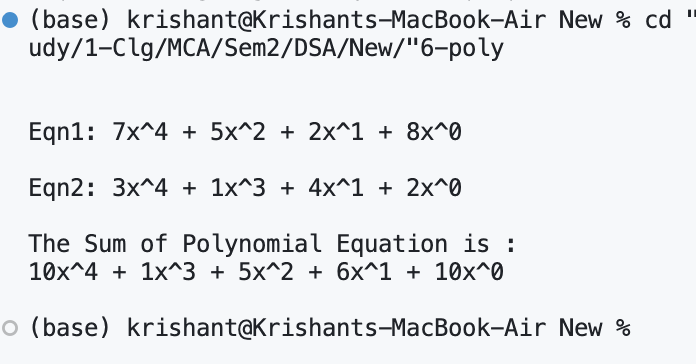
break;

}

} while (ch != 4);

return 0;

}

**OUTPUT :-**

Q7. Write a C program to sort a sequence of characters given by user in an array, using Quick sort technique.

#include <stdio.h>

void swap(int \*a, int \*b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int partition(int arr[], int low, int high)

{

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j <= high - 1; j++)

{

if (arr[j] < pivot)

{

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

void quickSort(int arr[], int low, int high)

{

if (low < high)

{

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

int main()

{

int n = 0;

int choice;

int arr[100];

printf("Menu\n");

printf("1. Enter Characters\n");

printf("2. Sort Characters\n");

printf("3. Exit\n");

do

{

printf("\nEnter your choice: ");

scanf("%d", &choice);

switch (choice)

{

case 1:

printf("Enter ch: ");

char ch;

scanf(" %c", &ch);

arr[n++] = ch - 'a';

break;

case 2:

quickSort(arr, 0, n - 1);

printf("\n\nAfter Sorted Sequence is: ");

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

printf("%c ", arr[i] + 'a');

break;

case 3:

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

break;

default:

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

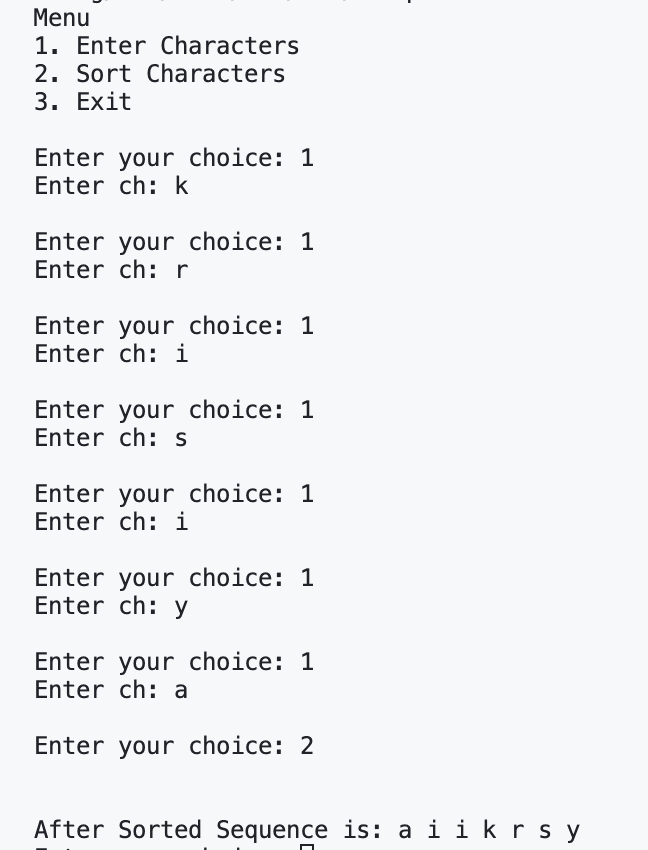
break;

}

} while (choice != 3);

return 0;

}

**OUTPUT :-**

**Q8. Using circular linked list allocate time slots of 10ms for given processes in time sharing Environment and then print which process will be completed in how much time.**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node

{

int id, time;

struct Node \*next;

} node;

node \*append()

{

static int id = 1;

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

printf("Burst Time: ");

scanf("%d", &temp->time);

temp->id = id++;

temp->next = NULL;

return temp;

}

node \*insert(node \*head)

{

node \*temp = append();

if (head == NULL)

{

temp->next = temp;

return temp;

}

temp->next = head->next;

head->next = temp;

head = temp;

return head;

}

node \*delete\_node(node \*head)

{

node \*curr = head;

if (head == NULL)

return head;

if (head == head->next)

{

free(head);

return NULL;

}

curr = curr->next;

head->id = curr->id;

head->time = curr->time;

head->next = curr->next;

free(curr);

return head;

}

node \*process(node \*head)

{

int quantum = 10;

int curr\_time = 0;

while (head != NULL)

{

if (head->time <= quantum)

{

curr\_time += head->time;

printf("Process ID : %d\n", head->id);

printf("Process Done:- %dms\n", curr\_time);

head = delete\_node(head);

}

else

{

curr\_time += quantum;

head->time -= quantum;

head = head->next;

}

}

return head;

}

int main()

{

node \*head = NULL;

int ch;

printf("\n\nKrishant Chauhan MCA 2A Roll no-32 \n");

printf("Menu\n");

printf("1. Insert process\n");

printf("2. Start process\n");

printf("3. Exit\n");

do

{

printf("Enter your choice: ");

scanf("%d", &ch);

switch (ch)

{

case 1:

head = insert(head);

break;

case 2:

if (head != NULL)

head = process(head->next);

break;

case 3:

printf("Exit\n");

break;

default:

printf("Invalid choice!\n");

break;

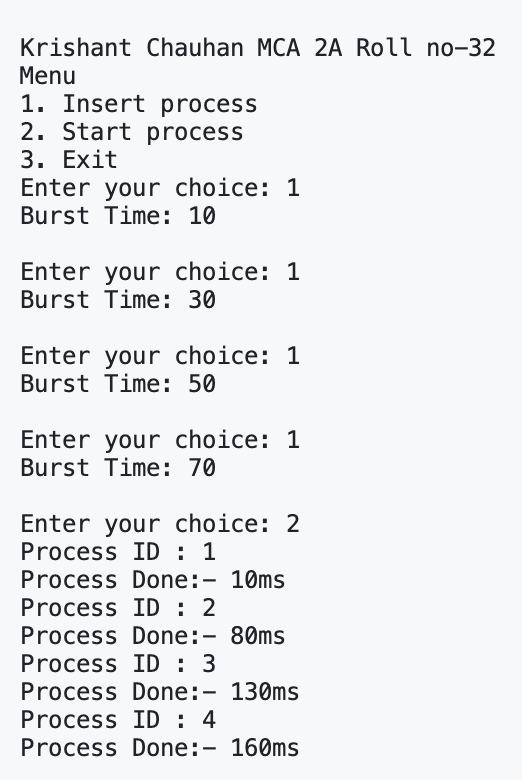
}

printf("\n");

} while (ch != 3);

return 0;

}

Output:-

**Q9. Write a C program to store the details of a weighted graph (Use array of pointers concept).**

#include <stdio.h>

#include <stdlib.h>

#define max 100

typedef struct node

{

int data, weight;

struct node \*next;

} node;

void insert(node \*graph[], int src, int dst, int weight)

{

node \*nn1 = malloc(sizeof(node));

node \*nn2 = malloc(sizeof(node));

nn1->data = dst;

nn1->weight = weight;

nn1->next = graph[src];

graph[src] = nn1;

nn2->data = src;

nn2->weight = weight;

nn2->next = graph[dst];

graph[dst] = nn2;

}

void disp(node \*head, int src)

{

while (head != NULL)

{

printf("%d --> %d (weight: %d)\n", src, head->data, head->weight);

head = head->next;

}

}

int main()

{

node \*graph[max] = {NULL};

int ch, src, dst;

printf("\n\nKrishant Chauhan MCA 2A Roll no-32 \n");

printf("Menu\n");

printf("1. Insert\n");

printf("2. Display\n");

printf("3. Exit\n");

do

{

printf("Enter your choice: ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter source, destination, and weight:- ");

scanf("%d%d%d", &src, &dst, &ch);

insert(graph, src, dst, ch);

break;

case 2:

for (ch = 0; ch < max; ch++)

{

if (graph[ch] != NULL)

{

disp(graph[ch], ch);

printf("\n");

}

}

break;

case 3:

printf("Exit\n");

break;

default:

printf("Invalid choice!\n");

break;

}

printf("\n");

} while (ch != 3);

return 0;

}

Output:-