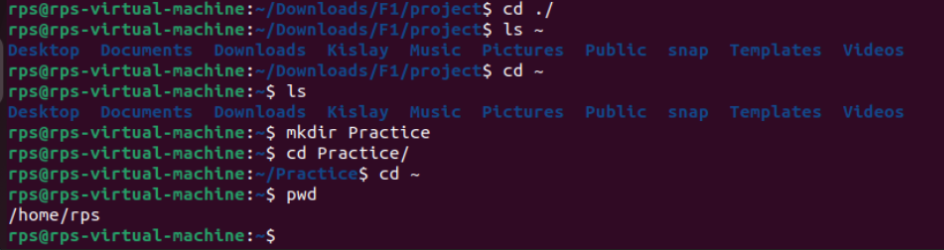
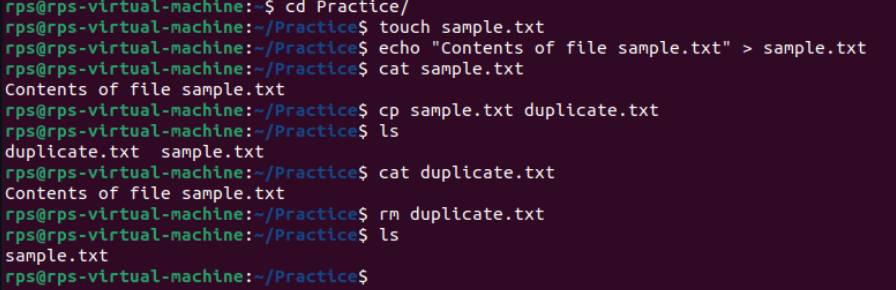
**Linux Commands(9th May)**

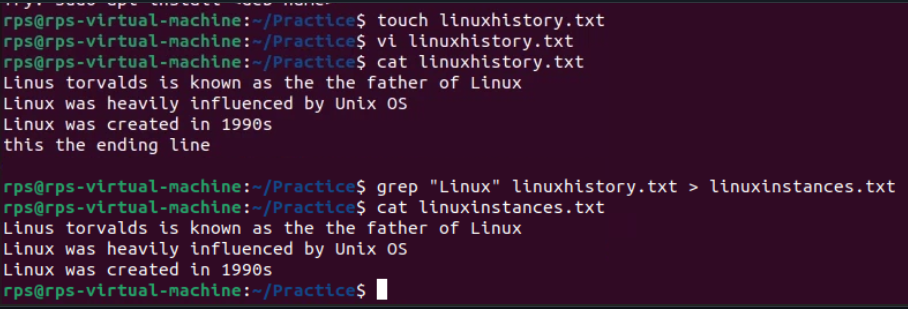
Task 1)



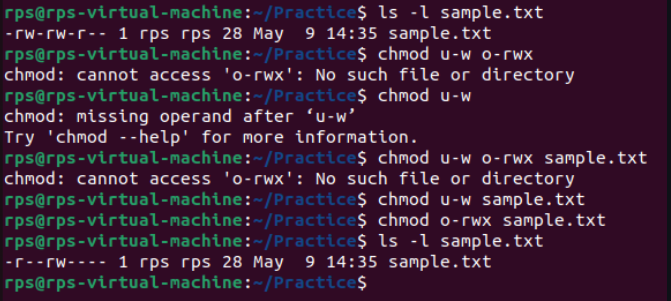
Task 2)



Task 3)



Task 4)



LINUX Command History File Contents:   
  
1 sudo apt update

2 sudo nano /etc/resolv.conf

3 sudo apt update

4 sudo apt upgrade

5 sudo apt install gcc++

6 sudo apt install gcc

7 sudo apt install build-essentials

8 sudo apt install build-essential

9 sudo apt install gdb

10 init 0

11 gcc --version

12 cd "/home/rps/Desktop/C Demo/" && gcc first.c -o first && "/home/rps/Desktop/C Demo/"first

13 sudo su

14 tar xvfz node\_exporter-1.7.0.linux-amd64.tar.gz

15 cd node\_exporter-1.7.0.linux-amd64/

16 sudo mv node\_exporter /usr/local/bin/

17 sudo tee /etc/systemd/system/node\_exporter.service<<EOF

18 [Unit]

19 Description=Node Exporter

20 After=network.target

21

22 [Service]

23 User=rps

24 Group=rps

25 Type=simple

26 ExecStart=/usr/local/bin/node\_exporter

27

28 [Install]

29 WantedBy=multi-user.target

30 EOF

31 sudo systemctl daemon-reload

32 sudo systemctl start node\_exporter

33 sudo systemctl enable node\_exporter

34 sudo systemctl status node\_exporter

35 sudo apt install gcc

36 sudo apt install git

37 sudo apt install make

38 sudo apt install vim

39 sudo apt install g++

40 sudo apt install gedit

41 sudo apt install cmake

42 sudo apt install g++

43 cd

44 sudo apt update -y

45 sudo apt install mysql-server

46 sudo systemctl status mysql.service

47 sudo mysql

48 sudo mysql\_secure\_installation

49 sudo mysql

50 sudo mysql\_secure\_installation

51 sudo mysql -u root -p

52 sudo snap install mysql-workbench-community

53 man

54 man man

55 ls

56 cd

57 man cd

58 man mkdir

59 pwd

60 ls

61 cd ~

62 ls

63 cd Desktop/

64 mkdir F1

65 touch file1

66 touch file2

67 ls

68 cd F1

69 touch file1

70 touch file2

71 lsssssss

72 ls

73 echo "contents of file 1" >> file1

74 echo "contents of file 2" >> file2

75 cat file1

76 cat file2

77 cd ~

78 ls

79 cd Downloads/

80 cd ~/Desktop

81 cp F1 ~/Downloads/

82 cp -r F1 ~/Downloads/

83 su

84 pwd

85 cd F!

86 cd F1

87 ls

88 open .

89 ln file1 file2

90 ln file1 lnkfile

91 cat lnkfile

92 cat lnkfile

93 echo "some more contents" >> lnkfile

94 cat lnkfile

95 cat file1

96 cd ~

97 cd Downloads/

98 ls

99 gzip -c tutorial.pdf > tutorial.gz

100 open .

101 history

102 echo "hello linux"

103 chomod --x

104 ping wwww.google.com

105 ping 8.8.8.8

106 history

107 sudo useradd suprit

108 sudo useradd -m -s /bin/bash suprit

109 sudo apt-get update

110 apt-get update

111 apt-get upgrade

112 sudo apt-get update

113 sudo apt-get upgrade

114 man 1 grep

115 man 7 grep

116 whatis grep

117 help grep

118 help ls

119 man -k ls

120 man -k grep

121 info grep

122 man 7 grep

123 man 3 grep

124 info grep

125 cd ..

126 ls

195 cd .ssh

196 ssh-keygen -t rsa -b 4096 -C "supritraj005@gmail.com"

197 cat id\_rsa.pub

198 cd !

199 cd ~

200 cd /home/rps/

201 cd Downloads/

202 cd F1/project/

203 git init

204 git add .

205 git remote add origin git@github.com:Supritraj5/tasks.git

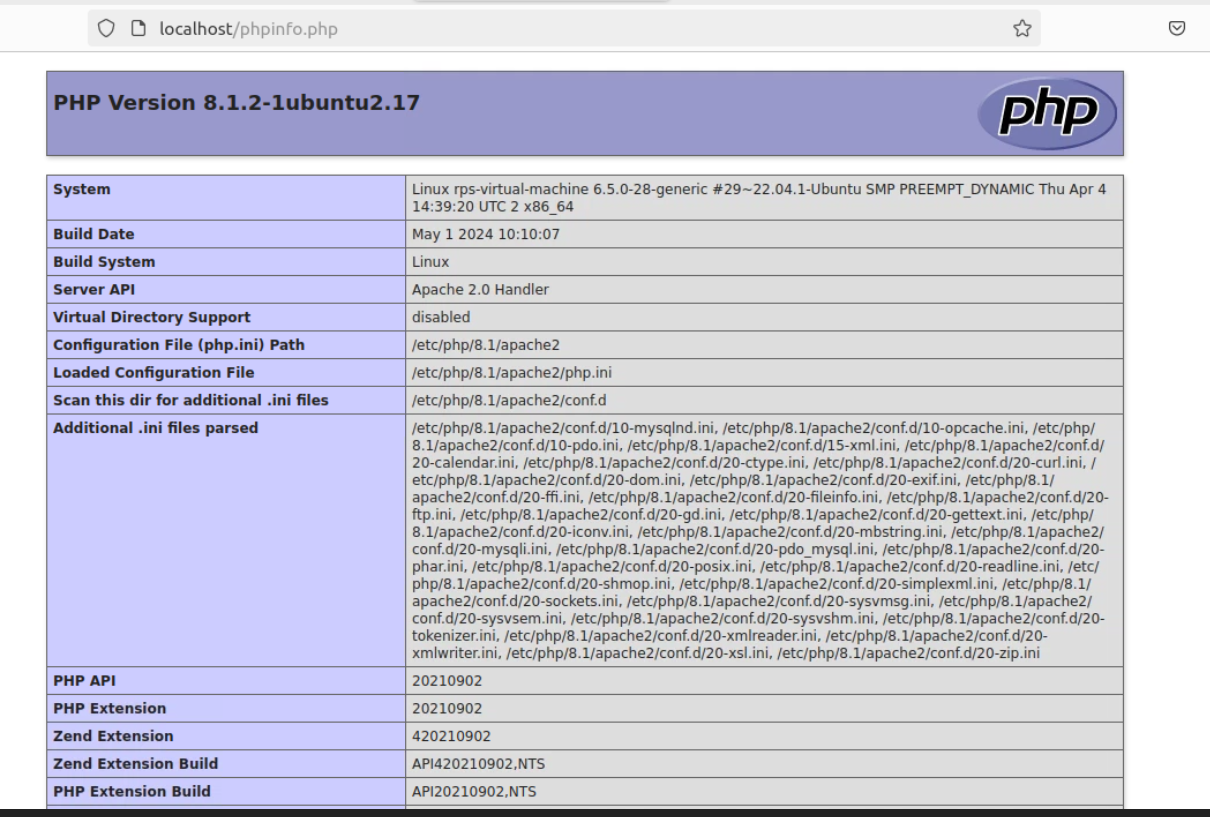
206 git commit -m "fresh commit"

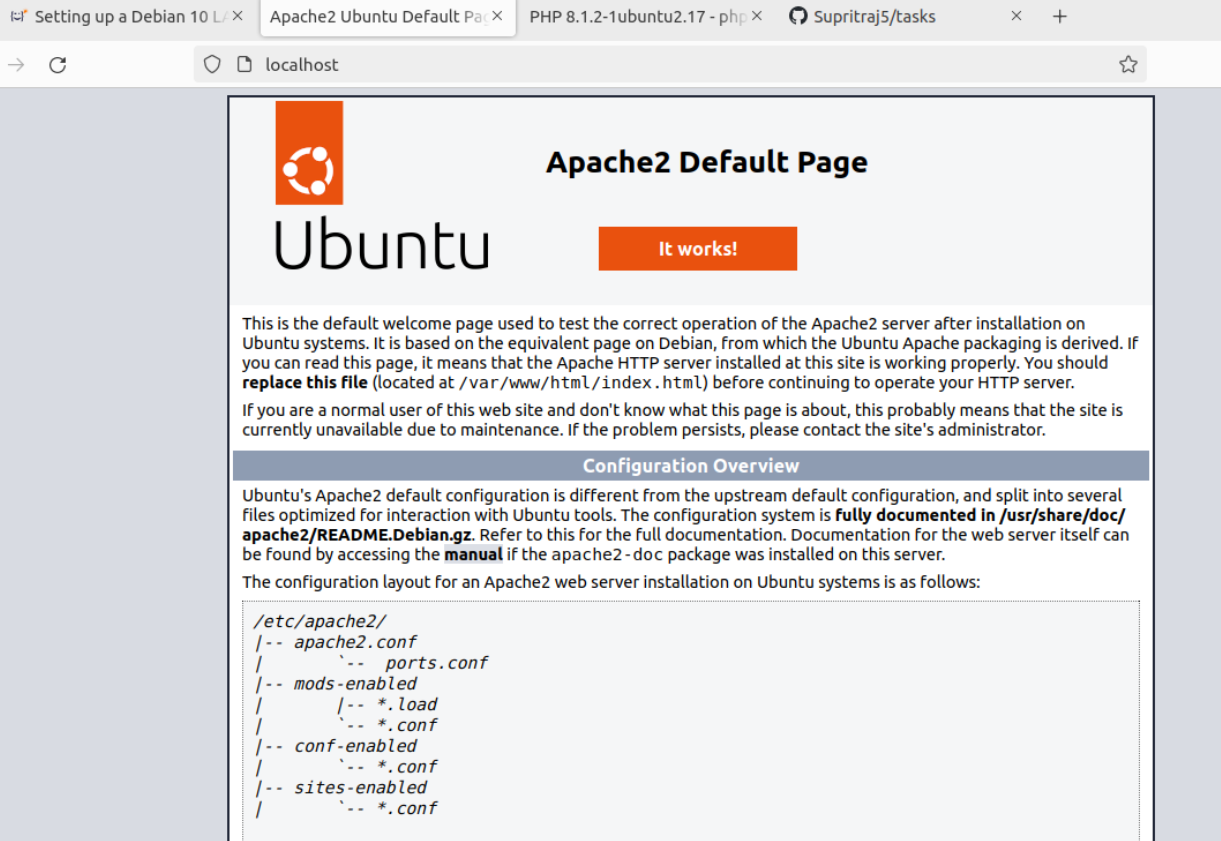
207 git push origin master

208 history

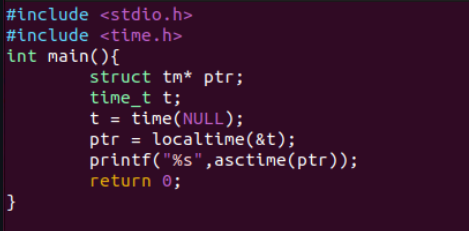
**Server Setup In Remote Machine**

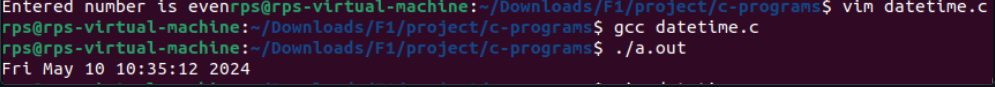
**Php Info**

**  
  
Localhost**

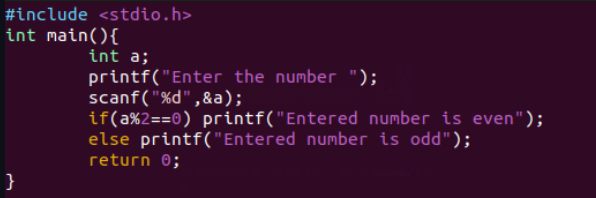
**  
  
C- Programs done on Linux Remote Machine**

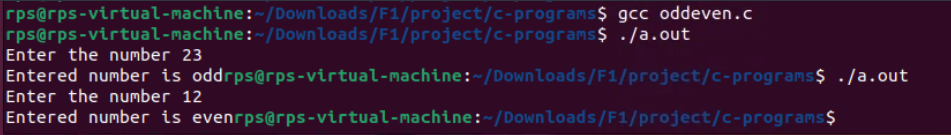
1. **Date and time**

****

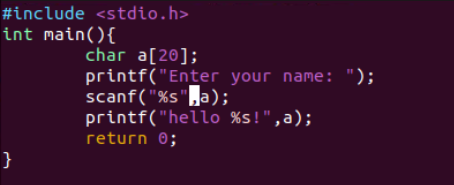
**Output –**

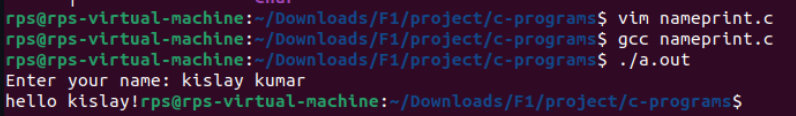
1. **Odd Even**

****

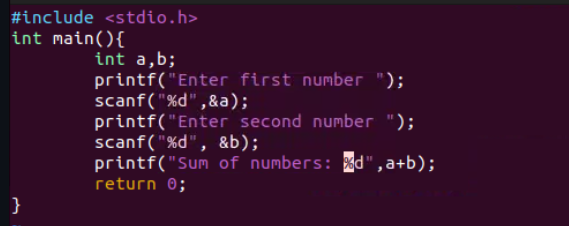
**Output– **

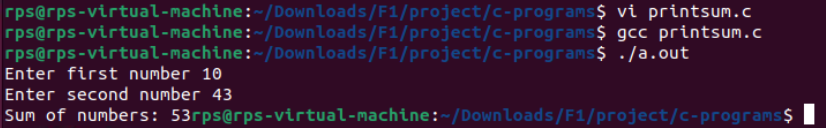
1. **Print Name**

****

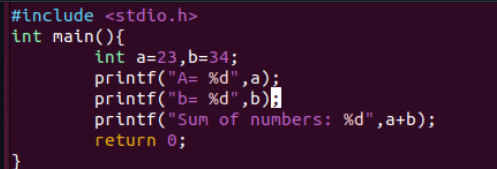
**Output– **

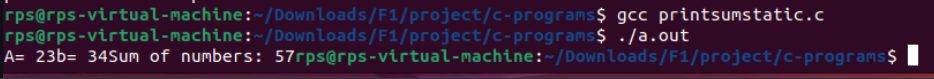
1. **Sum two numbers with user inputs**

****

**Output– **

1. **Sum two static**

****

**Output– **

**Calculator**

#include<stdio.h>

#include<conio.h>

int main() {

int choice;

int runAgain = 0;

int a, b;

float c,d;

while(1) {

printf("onec\n\n");

printf("\n\n\tCALCULATOR\n\n");

printf("\n1) For Addition 2) For Subtraction 3) For Multiplication 4) For Division\nEnter your choice:");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("\n\n\t-------ADDITION-------");

printf("\n\nEnter first number : ");

scanf("%d", &a);

printf("\nEnter second number : ");

scanf("%d", &b);

printf("\nSum of %d and %d = %d", a, b, a + b);

break;

case 2:

printf("\n\n\t-------SUBTRACTION-------");

printf("\n\nEnter first number : ");

scanf("%d", &a);

printf("\nEnter second number : ");

scanf("%d", &b);

printf("\nDifference of %d and %d = %d", a, b, a - b);

break;

case 3:

printf("\n\n\t-------MULTIPLICATION-------");

printf("\n\nEnter first number : ");

scanf("%d", &a);

printf("\nEnter second number : ");

scanf("%d", &b);

printf("\nProduct of %d and %d = %d", a, b, a \* b);

break;

case 4:

printf("\n\n\t-------DIVISION-------");

printf("\n\nEnter Dividend : ");

scanf("%f", &c);

printf("\nEnter Divisor : ");

scanf("%f", &d);

if(d==0.0){

printf("\n Err, Cannot divide by 0");

}

else if(d>0.0{

printf("\nQuotient of %.2f / %.2d = %.2d", a, b, a / b);

}

break;

default:

break;

}

int ans = 0;

printf("\n\nWant to do more calculations, press 1 for yes 0 for no : ");

scanf("%d",&ans);

//getch();

if(ans == 1){

continue;

}else{

printf("bye");

break;

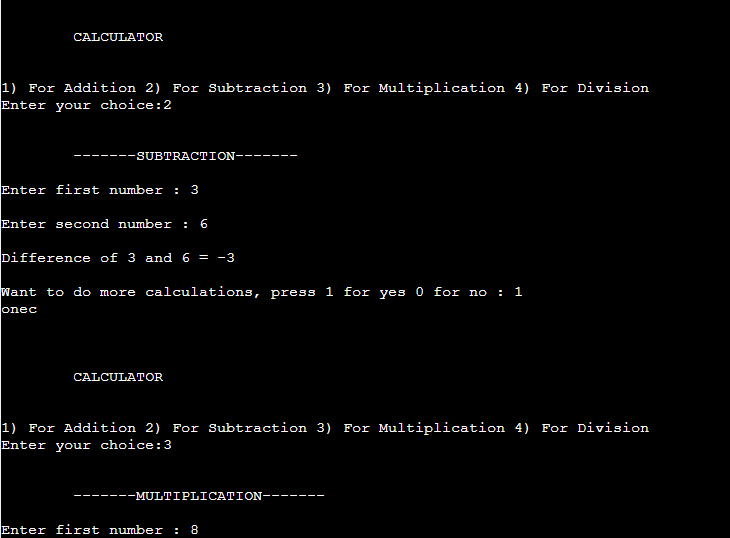
}

} ;

return 0;

}

**Output–**

****

**STACK implementation**

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#define MAX 1000

struct Stack {

int top;

int arr[MAX];

};

struct Stack\* createStack() {

struct Stack\* stack = (struct Stack\*)malloc(sizeof(struct Stack));

stack->top = -1;

return stack;

}

int isFull(struct Stack\* stack) {

return stack->top == MAX - 1;

}

int isEmpty(struct Stack\* stack) {

return stack->top == -1;

}

void push(struct Stack\* stack, int item) {

if (isFull(stack)) {

printf("Stack overflow\n");

return;

}

stack->arr[++stack->top] = item;

printf("%d pushed to stack\n", item);

}

int pop(struct Stack\* stack) {

if (isEmpty(stack)) {

printf("Stack underflow\n");

return INT\_MIN;

}

return stack->arr[stack->top--];

}

int peek(struct Stack\* stack) {

if (isEmpty(stack)) {

printf("Stack is empty\n");

return INT\_MIN;

}

return stack->arr[stack->top];

}

void printStack(struct Stack\* stack) {

if (isEmpty(stack)) {

printf("Stack is empty\n");

return;

}

for (int i = 0; i <= stack->top; i++) {

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

}

printf("\n");

}

int main() {

struct Stack\* stack = createStack();

push(stack, 10);

push(stack, 20);

push(stack, 30);

printf("%d popped from stack\n", pop(stack));

printf("Top element is %d\n", peek(stack));

printf("Elements present in stack: ");

printStack(stack);

return 0;

}

**Queue Implementation**

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#define MAX 1000

struct Queue {

int front, rear, size;

unsigned capacity;

int\* array;

};

struct Queue\* createQueue(unsigned capacity) {

struct Queue\* queue = (struct Queue\*)malloc(sizeof(struct Queue));

queue->capacity = capacity;

queue->front = queue->size = 0;

queue->rear = capacity - 1;

queue->array = (int\*)malloc(queue->capacity \* sizeof(int));

return queue;

}

int isFull(struct Queue\* queue) {

return (queue->size == queue->capacity);

}

int isEmpty(struct Queue\* queue) {

return (queue->size == 0);

}

// Function to add an item to the queue. It changes rear and size

void enqueue(struct Queue\* queue, int item) {

if (isFull(queue)) {

printf("Queue overflow\n");

return;

}

queue->rear = (queue->rear + 1) % queue->capacity;

queue->array[queue->rear] = item;

queue->size = queue->size + 1;

printf("%d enqueued to queue\n", item);

}

int dequeue(struct Queue\* queue) {

if (isEmpty(queue)) {

printf("Queue underflow\n");

return INT\_MIN;

}

int item = queue->array[queue->front];

queue->front = (queue->front + 1) % queue->capacity;

queue->size = queue->size - 1;

return item;

}

int front(struct Queue\* queue) {

if (isEmpty(queue)) {

printf("Queue is empty\n");

return INT\_MIN;

}

return queue->array[queue->front];

}

int rear(struct Queue\* queue) {

if (isEmpty(queue)) {

printf("Queue is empty\n");

return INT\_MIN;

}

return queue->array[queue->rear];

}

void printQueue(struct Queue\* queue) {

if (isEmpty(queue)) {

printf("Queue is empty\n");

return;

}

int i;

for (i = queue->front; i != queue->rear; i = (i + 1) % queue->capacity) {

printf("%d ", queue->array[i]);

}

printf("%d\n", queue->array[i]);

}

int main() {

struct Queue\* queue = createQueue(MAX);

enqueue(queue, 10);

enqueue(queue, 20);

enqueue(queue, 30);

enqueue(queue, 40);

printf("%d dequeued from queue\n", dequeue(queue));

printf("Front item is %d\n", front(queue));

printf("Rear item is %d\n", rear(queue));

printf("Elements present in queue: ");

printQueue(queue);

return 0;

}

**LinkedList Implementation**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* createNode(int data) {

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

newNode->data = data;

newNode->next = NULL;

return newNode;

}

void insertAtBeginning(struct Node\*\* head\_ref, int new\_data) {

struct Node\* new\_node = createNode(new\_data);

new\_node->next = \*head\_ref;

\*head\_ref = new\_node;

}

void insertAtEnd(struct Node\*\* head\_ref, int new\_data) {

struct Node\* new\_node = createNode(new\_data);

if (\*head\_ref == NULL) {

\*head\_ref = new\_node;

return;

}

struct Node\* temp = \*head\_ref;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = new\_node;

}

void deleteNode(struct Node\*\* head\_ref, int key) {

struct Node\* temp = \*head\_ref;

struct Node\* prev = NULL;

if (temp != NULL && temp->data == key) {

\*head\_ref = temp->next;

free(temp);

return;

}

while (temp != NULL && temp->data != key) {

prev = temp;

temp = temp->next;

}

if (temp == NULL) return;

prev->next = temp->next;

free(temp);

}

struct Node\* search(struct Node\* head, int key) {

struct Node\* current = head;

while (current != NULL) {

if (current->data == key) return current;

current = current->next;

}

return NULL;

}

void printList(struct Node\* node) {

while (node != NULL) {

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

node = node->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

insertAtEnd(&head, 1);

insertAtEnd(&head, 2);

insertAtEnd(&head, 3);

insertAtBeginning(&head, 0);

printf("Linked list: ");

printList(head);

deleteNode(&head, 2);

printf("Linked list after deletion of 2: ");

printList(head);

struct Node\* found = search(head, 3);

if (found != NULL) {

printf("Element 3 found in the linked list.\n");

} else {

printf("Element 3 not found in the linked list.\n");

}

return 0;

}

**Doubly Linked List Implementation**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* prev;

struct Node\* next;

};

struct Node\* createNode(int data);

void insertAtBeginning(struct Node\*\* head, int data);

void insertAtEnd(struct Node\*\* head, int data);

void deleteFromBeginning(struct Node\*\* head);

void deleteFromEnd(struct Node\*\* head);

void displayList(struct Node\* head);

int main() {

struct Node\* head = NULL;

insertAtBeginning(&head, 10);

insertAtBeginning(&head, 20);

insertAtEnd(&head, 30);

insertAtEnd(&head, 40);

printf("List after insertions: ");

displayList(head);

deleteFromBeginning(&head);

printf("List after deleting from beginning: ");

displayList(head);

deleteFromEnd(&head);

printf("List after deleting from end: ");

displayList(head);

return 0;

}

struct Node\* createNode(int data) {

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

newNode->data = data;

newNode->prev = NULL;

newNode->next = NULL;

return newNode;

}

void insertAtBeginning(struct Node\*\* head, int data) {

struct Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

} else {

newNode->next = \*head;

(\*head)->prev = newNode;

\*head = newNode;

}

}

void insertAtEnd(struct Node\*\* head, int data) {

struct Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

} else {

struct Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

newNode->prev = temp;

}

}

void deleteFromBeginning(struct Node\*\* head) {

if (\*head == NULL) {

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

return;

}

struct Node\* temp = \*head;

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

if (\*head != NULL) {

(\*head)->prev = NULL;

}

free(temp);

}

void deleteFromEnd(struct Node\*\* head) {

if (\*head == NULL) {

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

return;

}

struct Node\* temp = \*head;

if (temp->next == NULL) {

\*head = NULL;

} else {

while (temp->next != NULL) {

temp = temp->next;

}

temp->prev->next = NULL;

}

free(temp);

}

void displayList(struct Node\* head) {

if (head == NULL) {

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

return;

}

struct Node\* temp = head;

while (temp != NULL) {

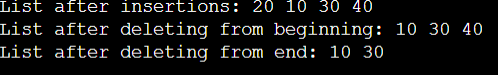
printf("%d ", temp->data);

temp = temp->next;

}

printf("\n");

}

**OUTPUT– **

**Calendar**

#include <stdio.h>

int dayOfWeek(int d, int m, int y)

{ static int t[] = {0, 3, 2, 5, 0, 3, 5, 1, 4, 6, 2, 4};

y -= m < 3;

return (y + y/4 - y/100 + y/400 + t[m-1] + d) % 7;

}

void printCalendar(int month, int year) {

int daysInMonth, i, currentDay;

int days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)) days[1] = 29;

printf(" \*\* Calendar - %d/%d \*\*\n", month, year);

printf(" Sun Mon Tue Wed Thu Fri Sat\n");

currentDay = dayOfWeek(1, month, year);

for (i = 0; i < currentDay; i++)

printf(" ");

for (i = 1; i <= days[month-1]; i++) {

printf("%5d", i);

if (++currentDay > 6) {

currentDay = 0;

printf("\n");

}

}

if (currentDay != 0) printf("\n"); }

int main() {

int month, year;

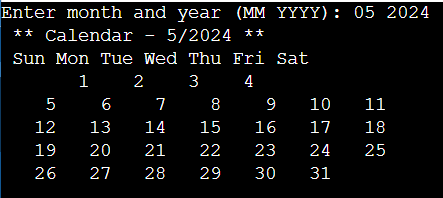
printf("Enter month and year (MM YYYY): ");

scanf("%d %d", &month, &year);

printCalendar(month, year);

return 0;

}

Output – 

**Size Of**

#include <stdio.h>

int main() {

int array[5];

int \*ptr;

ptr = array;

printf("Size of the array : %zu bytes\n", sizeof(array));

printf("Size of the pointer : %zu bytes\n", sizeof(ptr));

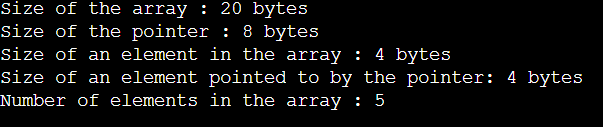
printf("Size of an element in the array : %zu bytes\n", sizeof(array[0]));

printf("Size of an element pointed to by the pointer: %zu bytes\n", sizeof(ptr[0]));

printf("Number of elements in the array : %zu\n", sizeof(array) / sizeof(array[0]));

return 0;

}

Output– 

**Structure Usage**

#include <stdio.h>

struct person{int age;

float weight;

};

int main(){

struct person \*personPtr,

person1;

personPtr = &person1;

printf("Enter age: ");

scanf("%d", &personPtr->age);

printf("Enter weight: ");

scanf("%f", &personPtr->weight);

printf("Displaying:\n");

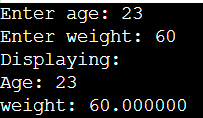
printf("Age: %d\n", personPtr->age);

printf("weight: %f", personPtr->weight);

return 0;

}

Output–

  
  
Tree traversal  
  
#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* newNode(int data) {

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

node->data = data;

node->left = NULL;

node->right = NULL;

return node;

}

void inorderTraversal(struct Node\* node) {

if (node == NULL) {

return;

}

inorderTraversal(node->left);

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

inorderTraversal(node->right);

}

void preorderTraversal(struct Node\* node) {

if (node == NULL) {

return;

}

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

preorderTraversal(node->left);

preorderTraversal(node->right);

}

void postorderTraversal(struct Node\* node) {

if (node == NULL) {

return;

}

postorderTraversal(node->left);

postorderTraversal(node->right);

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

}

int main() {

struct Node\* root = newNode(1);

root->left = newNode(2);

root->right = newNode(3);

root->left->left = newNode(4);

root->left->right = newNode(5);

printf("Inorder traversal: ");

inorderTraversal(root);

printf("\n");

printf("Preorder traversal: ");

preorderTraversal(root);

printf("\n");

printf("Postorder traversal: ");

postorderTraversal(root);

printf("\n");

return 0;

}

Output–

