Experiment No.: 1

Program Description:

Implementation of Linked List using array.

Solution:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX 10
struct List {
float list[MAX];
int length;
} fl;
int menu(void);
void create(void);
void insert(float value, int pos);
void delet(int pos);
void find(float value);
void display(void);
bool islistfull(void);
bool islistempty(void);
int menu() {
int ch;
printf("\n1. Create\n2. Insert\n3. Delete\n4. Count\n5. Find\n6. Display\n7.Exit\n\nEnter your choice: ");
scanf("%d", &ch);
return ch;
```

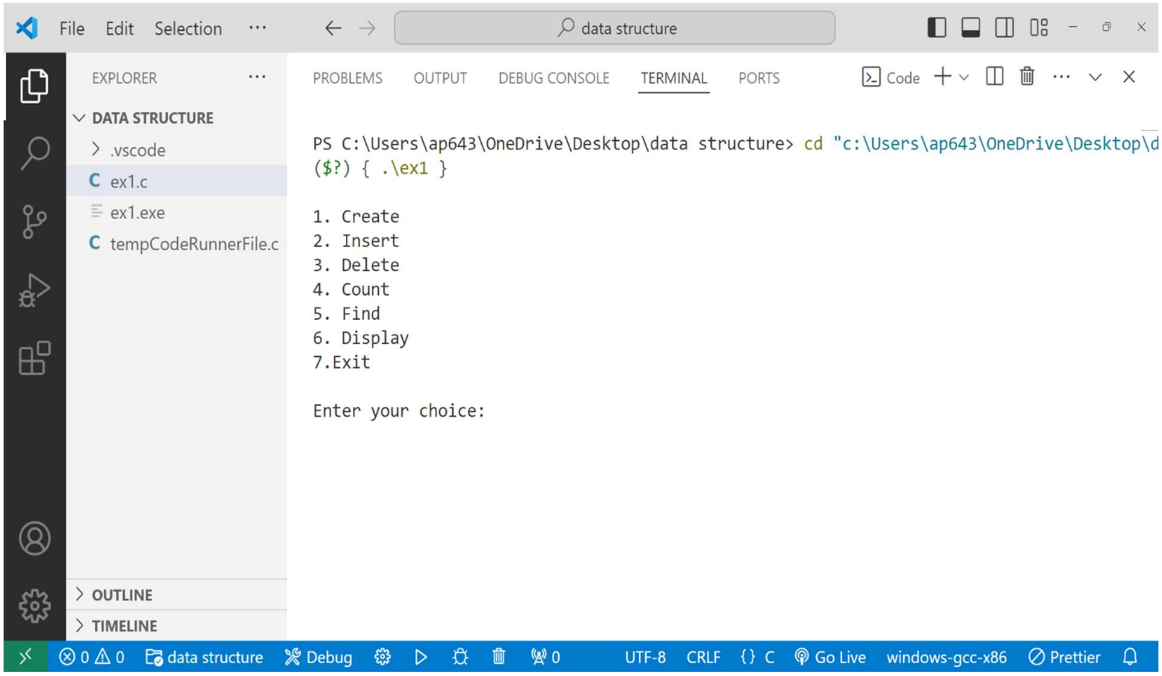
```
}
void create(void) {
float value;
fl.length = 0;
while (true) {
if (islistfull()) {
printf("List is full. Cannot add more elements.\n");
break;
}
printf("Enter value: ");
scanf("%f", &value);
fl.list[fl.length++] = value;
printf("To insert another value press 1, otherwise 0: ");
int flag;
scanf("%d", &flag);
if (flag != 1) break;
}
}
void display(void) {
if (islistempty()) {
printf("List is empty.\n");
return;}
for (int i = 0; i < fl.length; i++) {
printf("Element %d: %.2f\n", i + 1, fl.list[i]);}}
void insert(float value, int pos) {
if (pos <= 0 || pos > fl.length + 1) {
```

```
printf("Invalid position. Valid positions: 1 to %d.\n", fl.length + 1);
return;
}
if (islistfull()) {
printf("List is full. Cannot insert the value.\n");
return;
}
for (int i = fl.length; i >= pos - 1; i--) {
fl.list[i + 1] = fl.list[i];
}
fl.list[pos - 1] = value;
fl.length++;
}
void delet(int pos) {
if (pos <= 0 || pos > fl.length) {
printf("Invalid position. Valid positions: 1 to %d.\n", fl.length);
return;
}
for (int i = pos - 1; i < fl.length - 1; i++) {
fl.list[i] = fl.list[i + 1];
}
fl.length--;
}
void find(float value) {
for (int i = 0; i < fl.length; i++) {
if (fl.list[i] == value) {
```

```
printf("%.2f exists at position %d.\n", value, i + 1);
return;}
}
printf("Value not found.\n");
}
bool islistfull(void) {
return fl.length == MAX;
}
bool islistempty(void) {
}
return fl.length == 0;}
int main() {
int ch, pos;
float value;
fl.length = 0;  // Initialize the list length
while (1) {
ch = menu();
switch (ch) {
case 1:
create();
break;
case 2:
printf("Enter the value to insert: ");
scanf("%f", &value);
printf("Enter the position: ");
scanf("%d", &pos);
```

```
insert(value, pos);
break;
case 3:
printf("Enter the position of the value to delete: ");
scanf("%d", &pos);
delet(pos);
break;
case 4:
printf("Number of elements in the list: %d\n", fl.length);
break;
case 5:
printf("Enter the value to search: ");
scanf("%f", &value);
find(value);
break;
case 6:
display();
break;
case 7:
printf("Exiting...\n");
return 0;
default:
printf("Invalid choice. Please try again.\n");}
}
}
```

Output:


File Edit Selection ···
data structure
EXPLORER
DATA STRUCTURE
.vscode
ex1.c
ex1.exe
tempCodeRunnerFile.c
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Code
PS C:\Users\ap643\OneDrive\Desktop\data structure> cd "c:\Users\ap643\OneDrive\Desktop\d
($?) { .\ex1 }
1. Create
2. Insert
3. Delete
4. Count
5. Find
6. Display
7.Exit
Enter your choice:
0 0 data structure Debug 0 UTF-8 CRLF { } C Go Live windows-gcc-x86 Prettier

Experiment No.: 2

Program Description:

Implementation of Linked List using Pointers.

Output:

```
#include <stdio.h>

#include <stdlib.h>

struct node{

int data;

struct node *next;

};

struct node *start=NULL;

void insertFirst(){

int val;

struct node *new;

new=(struct node*)malloc(sizeof(struct node));

if(new==NULL){

printf("OVERFLOW\n\n");

}

else{

printf("Enter value : ");

scanf("%d",&val);

new->data=val;

new->next=start;

start=new;

printf("Insertion successful\n\n");}

};
```

```
void display(){
struct node *temp;
temp=start;
if(temp==NULL){
printf("UNDERFLOW\n\n");
}
else{
printf("\nSTART -> ");
while(temp->next!=NULL){
printf("%d -> ",temp->data);
temp=temp->next;
}
printf("%d -> ",temp->data);
printf("NULL\n\n");
}
};
void insertLast(){
int val;
struct node *new,*temp;
new=(struct node*)malloc(sizeof(struct node));
if(new==NULL){
printf("OVERFLOW\n\n");
}
else{
printf("Enter value : ");
scanf("%d",&val);
```

```
new->data=val;
new->next=NULL;
temp=start;
while(temp->next!=NULL){
temp=temp->next;
}
temp->next=new;
}
};
int length(){
int count=1;
struct node *temp;
temp=start;
if (temp==NULL)
return 0;
while(temp->next!=NULL){
temp=temp->next;
count++;
}
return count;
};
void insertLoc(){
int val,loc,l,count;
struct node *new,*temp;
printf("Enter location : ");
scanf("%d",&loc);
```

```
l=length();
if(loc>l || loc<=0){
printf("Invalid Location\n\n");
}
else{
new=(struct node*)malloc(sizeof(struct node));
if(new==NULL){
printf("OVERFLOW\n\n");
}
else{
printf("Enter value : ");
scanf("%d",&val);
new->data=val;
temp=start;
count=1;
while(count<loc){
temp=temp->next;
count++;
}
new->next=temp->next;
temp->next=new;
printf("Insertion Successful\n\n");
}
}
};
void deleteFirst(){
```

```
if (start==NULL)
{
printf("UNDERFLOW...\n\n");
}
else{
int val=start->data;
struct node *temp;
temp=start;
start=start->next;
free(temp);
printf("Deletion of %d successful\n\n",val);
}
};
void deleteLast(){
if (start==NULL){
printf("UNDERFLOW...\n\n");
}
else{
int val;
struct node *temp,*prev;
temp=start;
prev=temp;
while(temp->next!=NULL){
prev=temp;
temp=temp->next;
}
```

```
val=temp->data;
prev->next=NULL;
free(temp);
printf("Deletion of %d successful\n\n",val);
}
};
void deleteGiven(){
if (start==NULL){
printf("UNDERFLOW...\n\n");
}
else{
int val;
struct node *temp,*prev;
temp=start;
prev=temp;
printf("Enter value of node to delete : ");
scanf("%d",&val);
while(temp->data!=val || temp->next!=NULL){
prev=temp;
temp=temp->next;
}
if(temp->data!=val){
printf("Node with given value not found...\n\n");
}
else{
prev->next=temp->next;
```

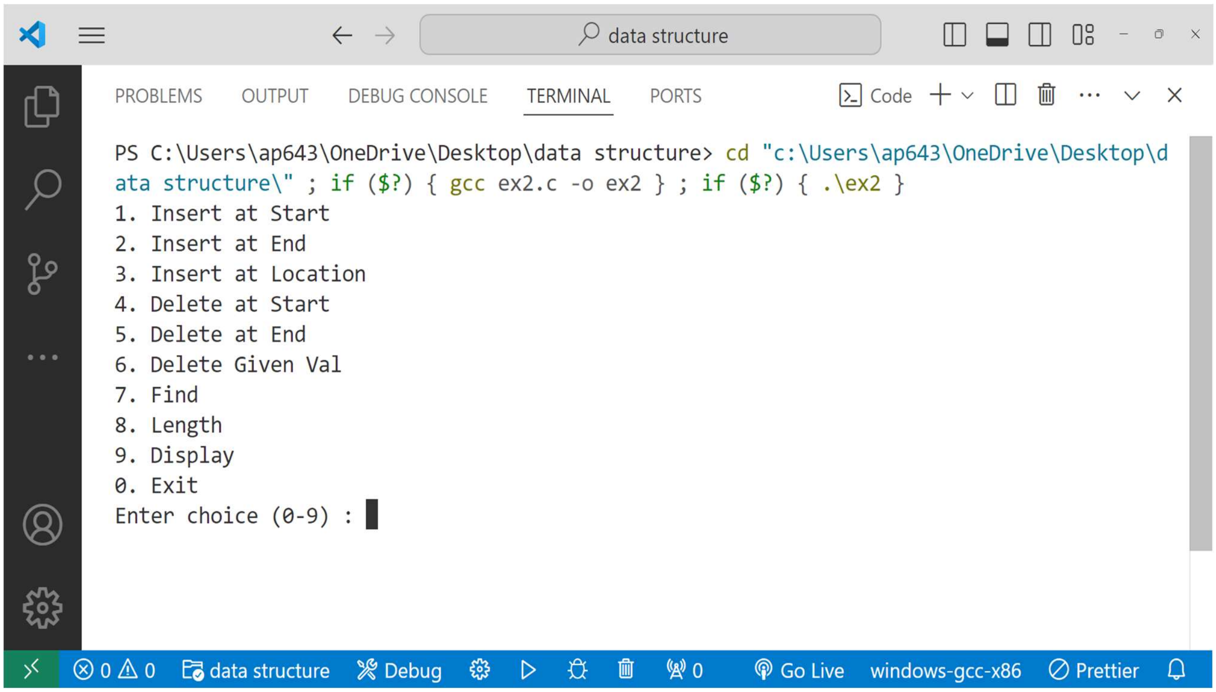
```
free(temp);
printf("Deletion of %d successful\n\n",val);
}
}
};
void find(){
if (start==NULL){
printf("UNDERFLOW...\n\n");
}
else{
int val,count;
struct node *temp;
temp=start;
printf("Enter value of node to find : ");
scanf("%d",&val);
count=0;
while(temp->data!=val || temp->next!=NULL){
temp=temp->next;
count++;
}
if(temp->data!=val){
printf("Node with given value not found...\n\n");
}
else{
count++;
printf("Node with given value found at location %d...\n\n",count);
```

```
}
}
};
int main(){
int choice;
while(choice!=0){
printf("1. Insert at Start\n");
printf("2. Insert at End\n");
printf("3. Insert at Location\n");
printf("4. Delete at Start\n");
printf("5. Delete at End\n");
printf("6. Delete Given Val\n");
printf("7. Find\n");
printf("8. Length \n");
printf("9. Display\n");
printf("0. Exit\n");
printf("Enter choice (0-9) : ");
scanf("%d",&choice);
switch(choice){
case 1:
insertFirst();
break;
case 2:
insertLast();
break;
case 3:
```

```
insertLoc();
break;
case 4:
deleteFirst();
break;
case 5:
deleteLast();
break;
case 6:
deleteGiven();
break;
case 7:
find();
break;
case 8:
printf("\n\nLength : %d\n\n",length());
break;
case 9:
display();
break;
case 0:
printf("Thanks for using our program...\n\n");
break;
default:
printf("Incorrect  choice\n\n");
break;
```

```
}

}

return 0;}
```

Output:



Experiment No.: 3

Program Description:

Implementation of Doubly Linked List using Pointers.

Solution:

```
#include <stdio.h>
#include <stdlib.h>
struct node {
struct node *prev;
struct node *next;
int data;
};
void insertionFirst();
void insertionLast();
void insertionLoc();
void deleteFirst();
void deleteLast();
void deleteLoc();
void printList();
void searchList();
struct node *head = NULL;
int main() {
int choice = 0;
while (choice != 9) {
printf("\nDoubly Linked List Menu\n");
printf("1. Insert at beginning\n");
printf("2. Insert at last\n");
```

```
printf("3. Insert at any random location\n");
printf("4. Delete from beginning\n");
printf("5. Delete from last\n");
printf("6. Delete the node after the given data\n");
printf("7. Search\n");
printf("8. Show\n");
printf("9. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
insertionFirst();
break;
case 2:
insertionLast();
break;
case 3:
insertionLoc();
break;
case 4:
deleteFirst();
break;
case 5:
deleteLast();
break;
case 6:
```

```
deleteLoc();
break;
case 7:
searchList();
break;
case 8:
printList();
break;
case 9:
exit(0);
break;
default:
printf("Invalid Choice! Please try again.\n");
}
}
return 0;
}
void insertionFirst() {
struct node *ptr = (struct node *)malloc(sizeof(struct node));
int item;
if (ptr == NULL) {
printf("\nOVERFLOW!!!");
} else {
printf("\nEnter value to insert: ");
scanf("%d", &item);
ptr->data = item;
```

```
ptr->prev = NULL;
if (head == NULL) {
ptr->next = NULL;
head = ptr;
} else {
ptr->next = head;
head->prev = ptr;
head = ptr;
}
printf("\nNode inserted successfully.\n");
}
}
void insertionLast() {
struct node *ptr = (struct node *)malloc(sizeof(struct node));
int item;
if (ptr == NULL) {
printf("\nOVERFLOW!!!");
} else {
printf("\nEnter value to insert: ");
scanf("%d", &item);
ptr->data = item;
if (head == NULL) {
ptr->next = NULL;
ptr->prev = NULL;
head = ptr;
} else {
```

```
struct node *temp = head;
while (temp->next != NULL) {
temp = temp->next;
}
temp->next = ptr;
ptr->prev = temp;
ptr->next = NULL;
}
printf("\nNode inserted successfully.\n");
}
}
void insertionLoc() {
struct node *ptr = (struct node *)malloc(sizeof(struct node));
int item, loc, i;
if (ptr == NULL) {
printf("\nOVERFLOW!!!");
} else {
printf("\nEnter the location: ");
scanf("%d", &loc);
printf("Enter value: ");
scanf("%d", &item);
ptr->data = item;
struct node *temp = head;
for (i = 0; i < loc - 1; i++) {
if (temp == NULL) {
printf("\nThere are less than %d elements.\n", loc);
```

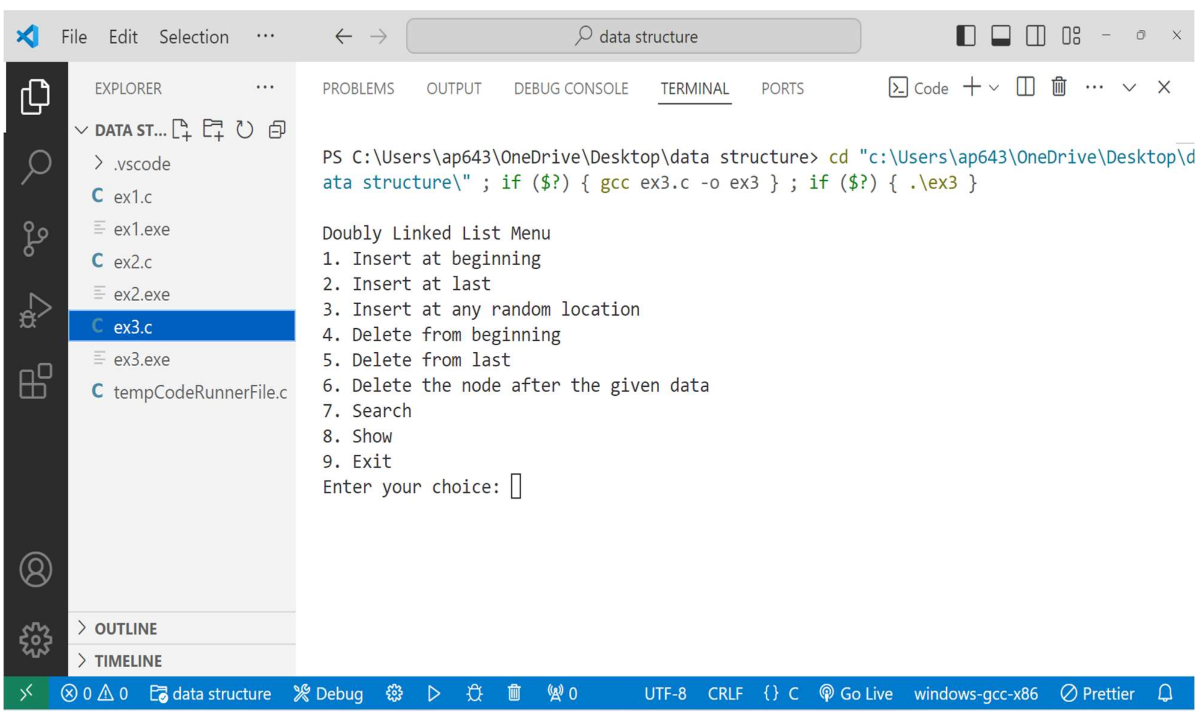
```
return;
}
temp = temp->next;
}
ptr->next = temp->next;
ptr->prev = temp;
if (temp->next != NULL) {
temp->next->prev = ptr;
}
temp->next = ptr;
printf("\nNode inserted successfully.\n");
}
}
void deleteFirst() {
if (head == NULL) {
printf("\nUNDERFLOW!!!");
} else {
struct node *ptr = head;
if (head->next == NULL) {
head = NULL;
} else {
head = head->next;
head->prev = NULL;
}
free(ptr);
printf("\nNode deleted successfully.\n");
```

```
}
}
void deleteLast() {
if (head == NULL) {
printf("\nUNDERFLOW!!!");
} else {
struct node *ptr = head;
if (head->next == NULL) {
head = NULL;
} else {
while (ptr->next != NULL) {
ptr = ptr->next;
}
ptr->prev->next = NULL;
}
free(ptr);
printf("\nNode deleted successfully.\n");
}
}
void deleteLoc() {
int val;
printf("\nEnter the data after which the node is to be deleted: ");
scanf("%d", &val);
struct node *ptr = head;
while (ptr != NULL && ptr->data != val) {
ptr = ptr->next;
```

```
}
if (ptr == NULL || ptr->next == NULL) {
printf("\nCan't delete. Node not found or no next node exists.\n");
} else {
struct node *temp = ptr->next;
ptr->next = temp->next;
if (temp->next != NULL) {
temp->next->prev = ptr;
}
free(temp);
printf("\nNode deleted successfully.\n");
}
}
void printList() {
struct node *ptr = head;
printf("\nThe Doubly Linked List is: START ⇄ ");
while (ptr != NULL) {
printf("%d ⇄ ", ptr->data);
ptr = ptr->next;
}
printf("NULL\n");
}
void searchList() {
int item, pos = 1, found = 0;
printf("\nEnter the item to search: ");
scanf("%d", &item);
```

```
struct node *ptr = head;
while (ptr != NULL) {
if (ptr->data == item) {
printf("\nItem %d found at position %d.\n", item, pos);
found = 1;
break;
}
ptr = ptr->next;
pos++;
}
if (!found) {
printf("\nItem %d not found in the list.\n", item);
}}
```

Output:



Experiment No.: 4

Program Description:

Implementation of Circular Single Linked List using Pointers.

Solution:

```
#include <stdio.h>

#include <stdlib.h>

struct Node{

int data;

struct Node *next;

};

struct Node *tail = NULL;

void create(){

int i, x;

printf("How Many Elements You want to add: ");

scanf("%d", &x);

for (i = 1; i <= x; i++)

{

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

printf("Enter data for node %d of the linked list: ", i);

scanf("%d", &newNode->data);

newNode->next = NULL;

if (tail == NULL)

{

tail = newNode;

tail->next = newNode;

}
```

```
else
{
newNode->next = tail->next;
tail->next = newNode;
tail = newNode;
}
}
}
void insertAtBeg()
{
struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
printf("Enter the data you want to insert: ");
scanf("%d", &newNode->data);
newNode->next = NULL;
if (tail == NULL)
{
tail = newNode;
tail->next = newNode;
}
else
{
newNode->next = tail->next;
tail->next = newNode;
}
}
```

```
void insertAtEnd()
{
struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
printf("Enter the data you want to insert: ");
scanf("%d", &newNode->data);
newNode->next = NULL;
if (tail == NULL)
{
tail = newNode;
tail->next = newNode;
}
else
{
newNode->next = tail->next;
tail->next = newNode;
tail = newNode;
}
}

void insertAtPos()
{
int pos, i = 1;
struct Node *newNode, *temp;
printf("Enter the Postion: ");
scanf("%d", &pos);
```

```
if (pos == 1)
{
insertAtBeg();
}
else
{
struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
printf("Enter data to be inserted: ");
scanf("%d", &newNode->data);
newNode->next = NULL;
temp = tail->next;
while (i < pos - 1)
{
temp = temp->next;
i++;
}
newNode->next = temp->next;
temp->next = newNode;
}
}
void insert_after()
{
int c, d;
struct Node *temp = tail->next;
struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
printf("Enter the data:");
```

```
scanf("%d", &d);
newNode->data = d;
newNode->next = NULL;
printf("Enter the value after which the data has to be inserted:");
scanf("%d", &c);
while (temp->data != c)
{
temp = temp->next;
}
newNode->next = temp->next;
temp->next = newNode;
}
void insert_before()
{
int c, d;
struct Node *newNode, *ptr, *preptr;
newNode = (struct Node *)malloc(sizeof(struct Node));
printf("Enter the data:");
scanf("%d", &d);
newNode->data = d;
printf("Enter the value before which the data has to be inserted:");
scanf("%d", &c);
ptr = tail->next;
while (ptr->data != c)
{
preptr = ptr;
```

```
ptr = ptr->next;
}
preptr->next = newNode;
newNode->next = ptr;
}
void delAtBeg()
{
struct Node *temp = tail->next;
if (tail == NULL)
{
printf("Linked List is Empty\n");
}
else
{
tail->next = temp->next;
free(temp);
}
}
void delAtEnd()
{
struct Node *current = tail->next;
struct Node *prev;
if (tail == NULL)
{
printf("Linked List is Empty\n");
}
```

```
else
{
while (current->next != tail->next)
{
prev = current;
current = current->next;
}
prev->next = tail->next;
tail = prev;
free(current);
}
}
void delAtPos()
{
struct Node *nextNode;
struct Node *current = tail->next;
int pos, i = 1;
printf("Enter the position you want to delete: ");
scanf("%d", &pos);
if (pos == 1)
{
delAtBeg();
}
else
{
while (i < pos - 1)
```

```
{
current = current->next;
i++;
}
nextNode = current->next;
current->next = nextNode->next;
free(nextNode);
}
}
void delete_after()
{
int c;
printf("Enter the value after which the data has to be deleted:");
scanf("%d", &c);
struct Node *ptr, *preptr, *temp;
if (tail == NULL)
{
printf("Linked List is empty\n");
}
else
{
ptr = tail->next;
preptr = ptr;
while (preptr->data != c)
{
preptr = ptr;
```

```
ptr = ptr->next;
}
temp = ptr;
preptr->next = temp->next;
free(temp);
}
}
void delete_before()
{
int c;
struct Node *ptr, *preptr;
printf("Enter the value before which the data has to be deleted:");
scanf("%d", &c);
ptr = tail->next;
preptr = ptr;
while (ptr->next->data != c)
{
preptr = ptr;
ptr = ptr->next;
}
preptr->next = ptr->next;
free(ptr);
}

void updateAtPos()
{
```

```
int pos, d, i = 1;
struct Node *temp = tail->next;
printf("Enter The Position to be Updated in the list:");
scanf("%d", &pos);
printf("Enter the updated data:");
scanf("%d", &d);
while (i < pos)
{
temp = temp->next;
i++;
}
temp->data = d;
}
void update_before(){
int c, d;
struct Node *prev;
struct Node *temp = tail->next;
printf("Enter the data after which the data has to be updated:");
scanf("%d", &c);
printf("Enter the updated data: ");
scanf("%d", &d);
prev = temp;
while (temp->data != c)
{
prev = temp;
temp = temp->next;
```

```
}
prev->data = d;
}
void update_after()
{
int c, d;
struct Node *temp = tail->next;
printf("Enter the data after which the data has to be updated:");
scanf("%d", &c);
printf("Enter the updated data: ");
scanf("%d", &d);
while (temp->data != c)
{
temp = temp->next;
}
temp->next->data = d;
}
void search(){
int x, i = 1;
struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
newNode = tail->next;
if (newNode == NULL)
{
printf("Linked List is empty\n");
}
else
```

```
{
printf("Enter the data you want to search\n");
scanf("%d", &x);
while (newNode->data != x)
{
newNode = newNode->next;
i++;
}
printf("Node found at %d\n", i);
}
}
void get_length()
{
int count = 0;
struct Node *temp = tail->next;
if (tail == NULL)
{
printf("Linked List is empty\n");
}
else
{
do
{
count++;
temp = temp->next;
} while (temp != tail->next);
```

```
}
printf("Length of Linked List is %d\n", count);
}
void display()
{
struct Node *temp = tail->next;
if (tail == NULL)
{
printf("Linked List is Empty\n");
}
else
{
do
{
printf("%d ", temp->data);
temp = temp->next;
} while (temp != tail->next);
}
printf("\n");
}

void reverse()
{
struct Node *prev, *nextNode;
struct Node *current = tail->next;
nextNode = current->next;
```

```
if (tail == NULL)
{
printf("Linked List is Empty\n");
}
else
{
while (current != tail)
{
prev = current;
current = nextNode;
nextNode = current->next;
current->next = prev;
}
nextNode->next = tail;
tail = nextNode;
}
display();
}
int main()
{
int opt;
while (1)
{
printf("\nwhich operation do you want to perform?\n");
printf("1.Create a Linked List\n");
printf("2.Display\n");
```

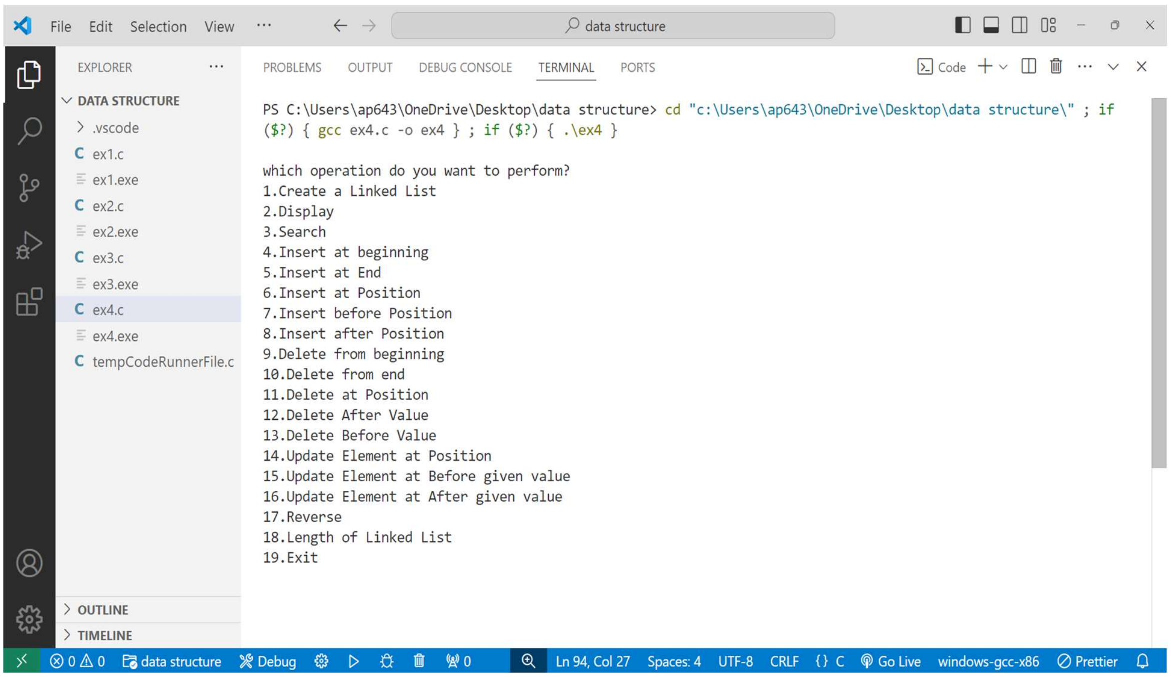
```
printf("3.Search\n");
printf("4.Insert at beginning\n");
printf("5.Insert at End\n");
printf("6.Insert at Position\n");
printf("7.Insert before Position\n");
printf("8.Insert after Position\n");
printf("9.Delete from beginning\n");
printf("10.Delete from end\n");
printf("11.Delete at Position\n");
printf("12.Delete After Value\n");
printf("13.Delete Before Value\n");
printf("14.Update Element at Position\n");
printf("15.Update Element at Before given value\n");
printf("16.Update Element at After given value\n");
printf("17.Reverse\n");
printf("18.Length of Linked List\n");
printf("19.Exit\n");
scanf("%d", &opt);
switch (opt)
{
case 1:
create();
break;
case 2:
display();
break;
```

```
case 3:
search();
break;
case 4:
insertAtBeg();
break;
case 5:
insertAtEnd();
break;
case 6:
insertAtPos();
break;
case 7:
insert_before();
break;
case 8:
insert_after();
break;
case 9:
delAtBeg();
break;
case 10:
delAtEnd();
break;
case 11:
delAtPos();
```

```
break;
case 12:
delete_after();
break;
case 13:
delete_before();
break;
case 14:
updateAtPos();
break;
case 15:
update_before();
break;
case 16:
update_after();
break;
case 17:
reverse();
break;
case 18:
get_length();
break;
case 19:
exit(0);
default:
printf("Unknown Choice !!\n");
```

```
}
}
return 0;
}
```

Output:



Experiment No.: 5

Program Description:

Implementation of Circular Doubly Linked List using Pointers.

Solution:

```
#include <stdio.h>
#include <stdlib.h>
struct Node{
int data;
struct Node *next;
struct Node *prev;
};
struct Node *head, *tail;
void create(){
int i, x;
printf("How Many Elements You want to add: ");
scanf("%d", &x);
for (i = 1; i <= x; i++){
struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
printf("Enter data for node %d of the linked list: ", i);
scanf("%d", &newNode->data);
if (head == NULL){
head = tail = newNode;
head->next = head->prev = newNode;}
else{
tail->next = newNode;
newNode->prev = tail;
```

```
newNode->next = head;
head->prev = newNode;
tail = newNode;
}
}
}
void insertAtBeg(){
struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
printf("Enter data to be inserted: ");
scanf("%d", &newNode->data);
if (head == NULL){
head = tail = newNode;
newNode->next = newNode->prev = head;
}
else{
newNode->next = head;
head->prev = newNode;
newNode->prev = tail;
tail->next = newNode;
head = newNode;
}
}
void insertAtEnd(){
struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
printf("Enter data to be inserted: ");
scanf("%d", &newNode->data);
```

```
if (head == NULL){
head = tail = newNode;
newNode->next = newNode->prev = head;
}
else{
newNode->prev = tail;
tail->next = newNode;
newNode->next = head;
head->prev = newNode;
tail = newNode;
}
}
void insertAtPos(){
int pos, i = 1;
struct Node *temp = head;
struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
printf("Enter the postion at which data is inserted: ");
scanf("%d", &pos);
if (pos == 1){
insertAtBeg();
}
else{
printf("Enter the postion at which data is inserted: ");
scanf("%d", &newNode->data);
while (i < pos - 1){
temp = temp->next;
```

```
i++;
}
newNode->prev = temp;
newNode->next = temp->next;
temp->next->prev = newNode;
temp->next = newNode;
}
}
void deleteAtBeg(){
struct Node *temp = head;
if (head == NULL){
printf("Linked List in Empty\n");
}
else if (head == tail)
{
head = tail = 0;
free(temp);
}
else{
head = head->next;
head->prev = tail;
tail->next = head;
free(temp);
}
}
void deleteAtEnd(){
```

```
struct Node *temp = tail;
if (head == NULL)
{
printf("Linked List in Empty\n");
}
else if (head == tail){
head = tail = 0;
free(temp);
}
else{
tail = tail->prev;
tail->next = head;
head->prev = tail;
free(temp);
}
}
void deleteAtPos()
{
int pos, i = 0;
struct Node *temp = head;
printf("Enter position: \n");
scanf("%d", &pos);
if (pos == 1){
deleteAtBeg();
}
else{
```

```
while (i < pos - 1){
temp = temp->next;
i++;
}
(temp->prev)->next = temp->next;
(temp->next)->prev = temp->prev;
if (temp->next == head){
tail = temp->prev;
free(temp);
}
else{
free(temp);
}
}
}
void update(){
struct Node *temp = head;
int c, d;
printf("Enter the value to be updated: \n");
scanf("%d", &c);
printf("Enter the data:");
scanf("%d", &d);
while (temp->data != c){
temp = temp->next;
}
temp->data = d;
```

```
}
void updateAtPos(){
struct Node *temp = head;
int pos, i = 0, x;
printf("Enter position to be updated: \n");
scanf("%d", &pos);
printf("Enter the data: \n");
scanf("%d", &x);
while (i < pos - 1){
temp = temp->next;
i++;
}
temp->data = x;
}
void get_length(){
int count = 1;
struct Node *temp = head;
if (head == NULL){
printf("Linked List is empty\n");
}
else{
while (temp != tail){
temp = temp->next;
count++;
}
}
```

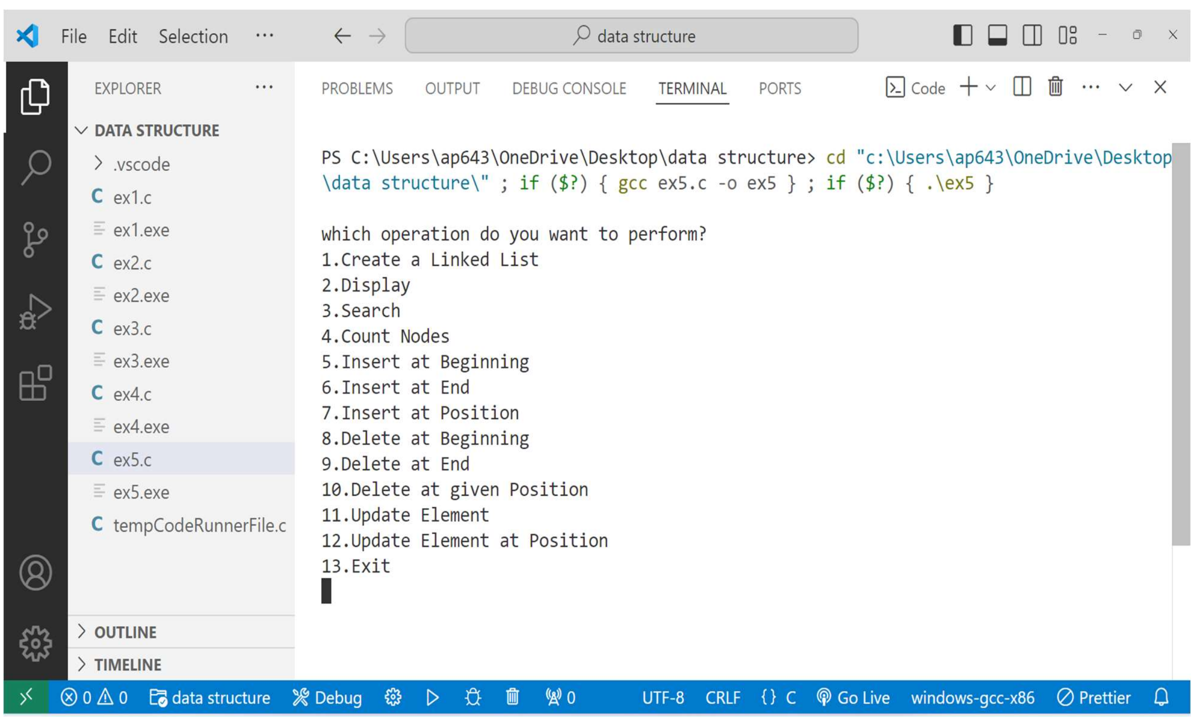
```
printf("Length of Linked List is %d\n", count);
}
void search(){
int x, i = 1;
struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
newNode = tail->next;
if (newNode == NULL){
printf("Linked List is empty\n");
}
else{
printf("Enter the data you want to search\n");
scanf("%d", &x);
while (newNode->data != x){
newNode = newNode->next;
i++;
}
printf("Node found at %d\n", i);}
}
void display(){
struct Node *temp = head;
if (head == NULL){
printf("Linked List is Empty\n");}
else{
do{
printf("%d ", temp->data);
temp = temp->next;
```

```
} while (temp != tail->next);
}
printf("\n");}
int main(){
int opt;
while (1){
printf("\nwhich operation do you want to perform?\n");
printf("1.Create a Linked List\n");
printf("2.Display\n");
printf("3.Search\n");
printf("4.Count Nodes\n");
printf("5.Insert at Beginning\n");
printf("6.Insert at End\n");
printf("7.Insert at Position\n");
printf("8.Delete at Beginning\n");
printf("9.Delete at End\n");
printf("10.Delete at given Position\n");
printf("11.Update Element \n");
printf("12.Update Element at Position\n");
printf("13.Exit\n");
scanf("%d", &opt);
switch (opt){
case 1:
create();
break;
case 2:
```

```
display();
break;
case 3:
search();
break;
case 4:
get_length();
break;
case 5:
insertAtBeg();
break;
case 6:
insertAtEnd();
break;
case 7:
insertAtPos();
break;
case 8:
deleteAtBeg();
break;
case 9:
deleteAtEnd();
break;
case 10:
deleteAtPos();
break;
```

```
case 11:
update();
break;
case 12:
updateAtPos();
break;
case 13:
exit(0);
default:
printf("Unknown Choice !!\n");}
}
return 0;}
```

Output:



Section-B (Stack)

Experiment No.: 1

Program Description:

Implementation of Stack using Array.

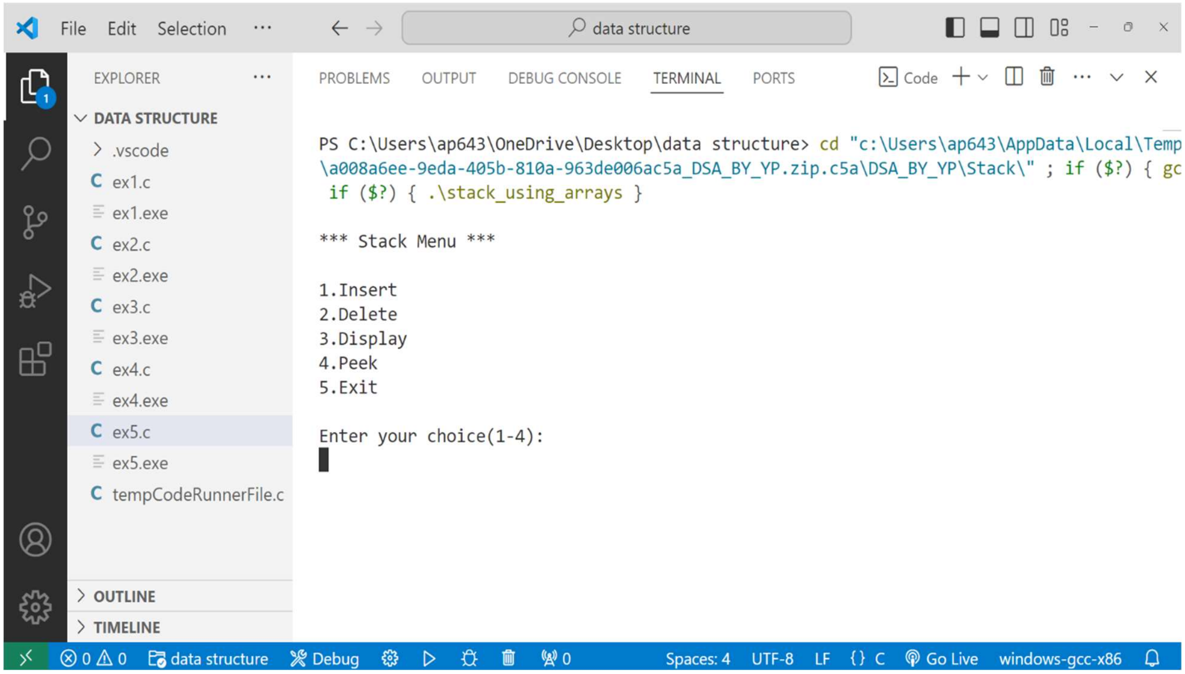
Solution:

```
#include <stdio.h>
#include <stdlib.h>
#define N 50
int stack[N];
int top = -1;
void push(){
int x;
printf("Enter The Element to push: ");
scanf("%d", &x);
if (top == N - 1){
printf("Stack overflow\n");
}
else{
top++;
stack[top] = x;
printf("%d pushed to Stack\n", x);
}
}
void pop(){
int item;
if (top == -1){
```

```
printf("Stack underflow\n");
}
else{
int item = stack[top];
top--;
printf("Popped: %d\n", item);
}
}
void peek(){
if (top == -1){
printf("Stack is empty\n");
}
else{
printf("Top Element: %d\n", stack[top]);
}
}
void display(){
int i;
printf("Displaying Stack....\n");
for (int i = top; i >= 0; i--)
{
printf("%d ", stack[i]);
}
printf("\n");
}
```

```
int main(){
int ch;
while (1){
printf("\n*** Stack Menu ***");
printf("\n\n1.Insert\n2.Delete\n3.Display\n4.Peek\n5.Exit");
printf("\n\nEnter your choice(1-4):");
scanf("%d", &ch);
switch (ch){
case 1:
push();
break;
case 2:
pop();
break;
case 3:
display();
break;
case 4:
peek();
break;
case 5:
exit(0);
default:
printf("\nWrong Choice!!");}
}
return 0;}
```

Output:


File Edit Selection ...
data structure
EXPLORER
DATA STRUCTURE
.vscode
ex1.c
ex1.exe
ex2.c
ex2.exe
ex3.c
ex3.exe
ex4.c
ex4.exe
ex5.c
ex5.exe
tempCodeRunnerFile.c
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Code
PS C:\Users\ap643\OneDrive\Desktop\data structure> cd "c:\Users\ap643\AppData\Local\Temp
\a008a6ee-9eda-405b-810a-963de006ac5a_DSA_BY_YP.zip.c5a\DSA_BY_YP\Stack\" ; if ($?) { gc
if ($?) { .\stack_using_arrays }
*** Stack Menu ***
1.Insert
2.Delete
3.Display
4.Peek
5.Exit
Enter your choice(1-4):
0 0 data structure Debug 0 Spaces: 4 UTF-8 LF {} C Go Live windows-gcc-x86

Experiment No.: 2

Program Description:

Implementation of Stack using Pointers**.**

Solution:

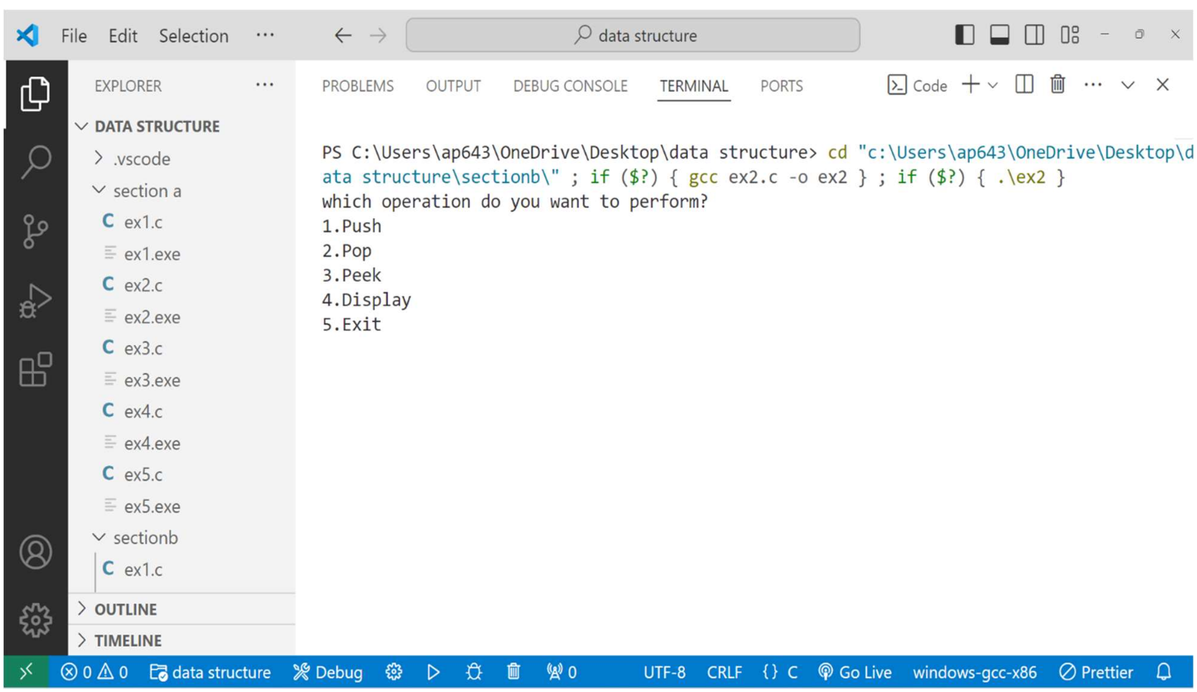
```
#include <stdio.h>
#include <stdlib.h>
struct Stack{
int data;
struct Stack *next;
};
struct Stack *top = NULL;
void push(){
int x;
printf("Enter Element to be pushed\n");
scanf("%d", &x);
struct Stack *newNode = (struct Stack *)malloc(sizeof(struct Stack));
newNode->data = x;
newNode->next = top;
top = newNode;
}
void pop(){
struct Stack *temp = (struct Stack *)malloc(sizeof(struct Stack));
temp = top;
if (top == NULL){
printf("Stack is empty\n");
}
```

```
else{
printf("Popped Element is %d\n", top->data);
top = top->next;
free(temp);
}
}
void peek(){
if (top == NULL)
{
printf("Stack is empty\n");
}
else
{
printf("Top element is %d\n", top->data);
}
}
void display(){
struct Stack *temp = (struct Stack *)malloc(sizeof(struct Stack));
temp = top;
if (top == NULL)
{
printf("Stack is empty\n");
}
else
{
while (temp != NULL){
```

```
printf("%d ", temp->data);
temp = temp->next;
}
}
printf("\n");
}
int main(){
int opt;
while (1)
{
printf("which operation do you want to perform?\n");
printf("1.Push\n");
printf("2.Pop\n");
printf("3.Peek\n");
printf("4.Display\n");
printf("5.Exit\n");
scanf("%d", &opt);
switch (opt){
case 1:
push();
break;
case 2:
pop();
break;
case 3:
peek();
```

```
break;
case 4:
display();
break;
case 5:
exit(0);
break;
default:
printf("Unknown operation\n");}
}
return 0;}
```

Output:



Experiment No.: 3

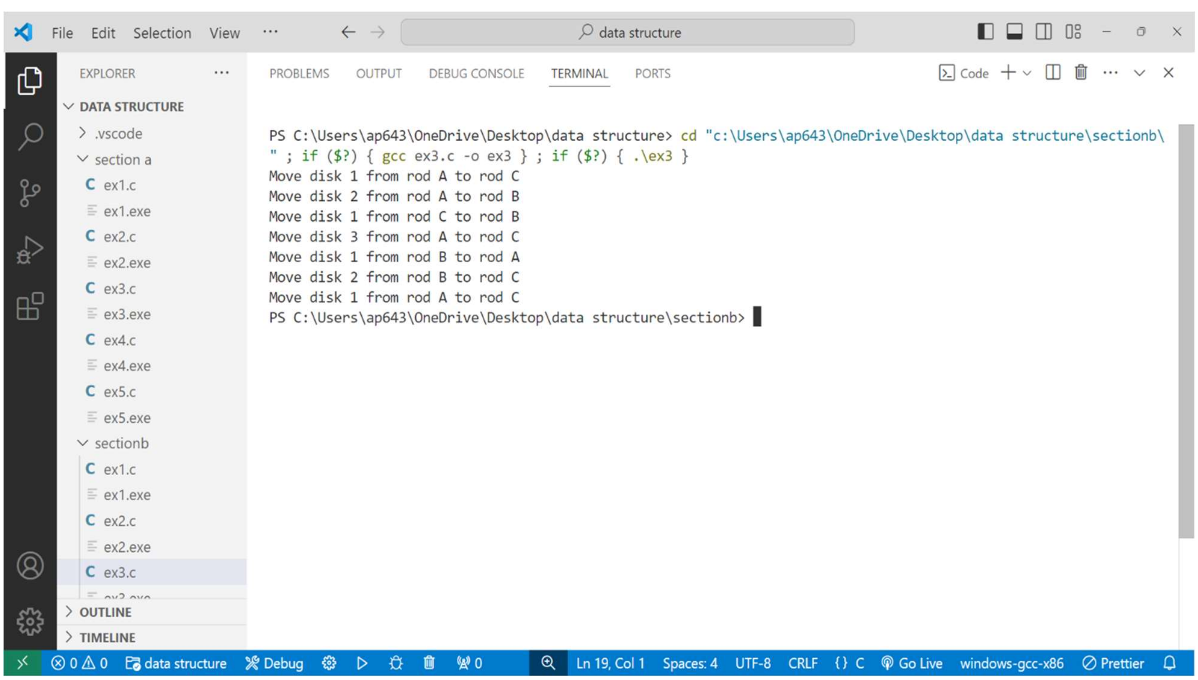
Program Description:

Program for Tower of Hanoi using recursion.

Solution:

```
#include <stdio.h>

void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod) {

if (n == 0) {

return;}

towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);

printf("Move disk %d from rod %c to rod %c\n", n, from_rod, to_rod);

towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);}

int main() {

int N = 3;

towerOfHanoi(N, 'A', 'C', 'B');

return 0;}
```

Output:



Experiment No.: 4

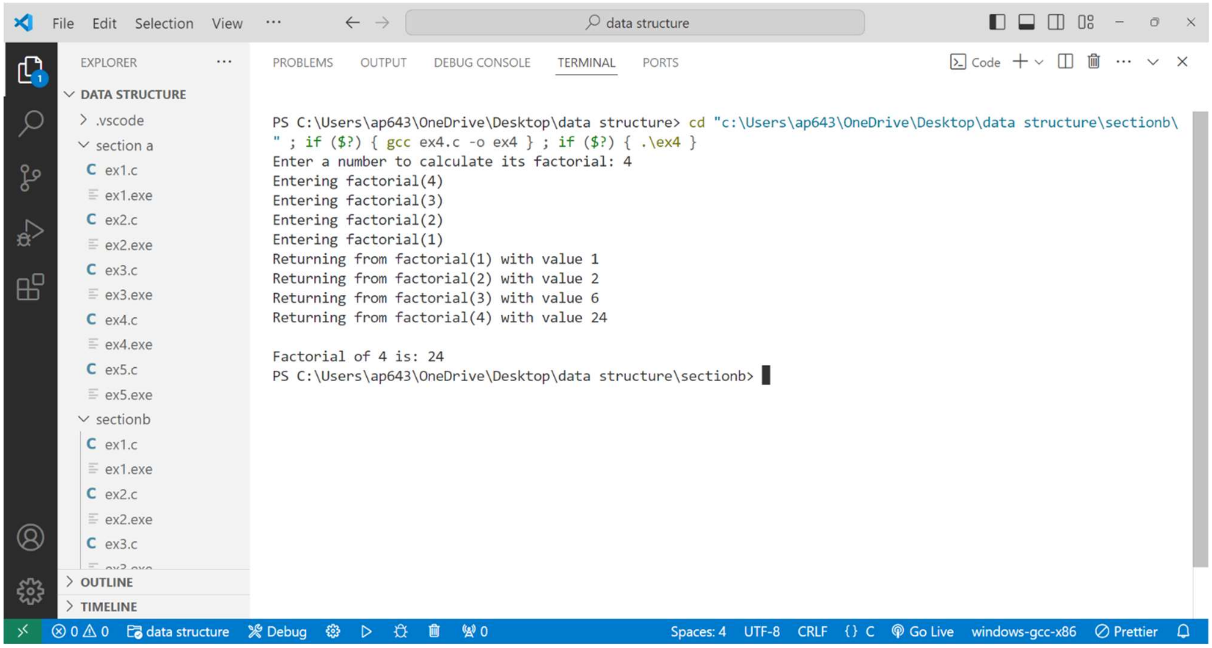
Program Description:

Program to find out factorial of given number using recursion. Also show the various states of stack using in this program.

Solution:

```
#include <stdio.h>
int factorial(int n) {
printf("Entering factorial(%d)\n", n);
if (n == 0 || n == 1) {
printf("Returning from factorial(%d) with value 1\n", n);
return 1;
}
int result = n * factorial(n - 1);
printf("Returning from factorial(%d) with value %d\n", n, result);
return result;
}
int main() {
int num;
printf("Enter a number to calculate its factorial: ");
scanf("%d", &num);
if (num < 0) {
printf("Factorial of a negative number is not defined.\n");
} else {
int result = factorial(num);
printf("\nFactorial of %d is: %d\n", num, result)}
return 0;}
```

Output:


File Edit Selection View
data structure
EXPLORER
DATA STRUCTURE
.vscode
section a
ex1.c
ex1.exe
ex2.c
ex2.exe
ex3.c
ex3.exe
ex4.c
ex4.exe
ex5.c
ex5.exe
sectionb
ex1.c
ex1.exe
ex2.c
ex2.exe
ex3.c
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Code
PS C:\Users\ap643\OneDrive\Desktop\data structure> cd "c:\Users\ap643\OneDrive\Desktop\data structure\sectionb\" ; if ($?) { gcc ex4.c -o ex4 } ; if ($?) { .\ex4 }
Enter a number to calculate its factorial: 4
Entering factorial(4)
Entering factorial(3)
Entering factorial(2)
Entering factorial(1)
Returning from factorial(1) with value 1
Returning from factorial(2) with value 2
Returning from factorial(3) with value 6
Returning from factorial(4) with value 24
Factorial of 4 is: 24
PS C:\Users\ap643\OneDrive\Desktop\data structure\sectionb>
0 0 data structure Debug 0
Spaces: 4 UTF-8 CRLF { } C Go Live windows-gcc-x86 Prettier

Section-C (Queue)

Experiment No.: 1

Program Description:

Implementation of Queue using Array.

Solution:

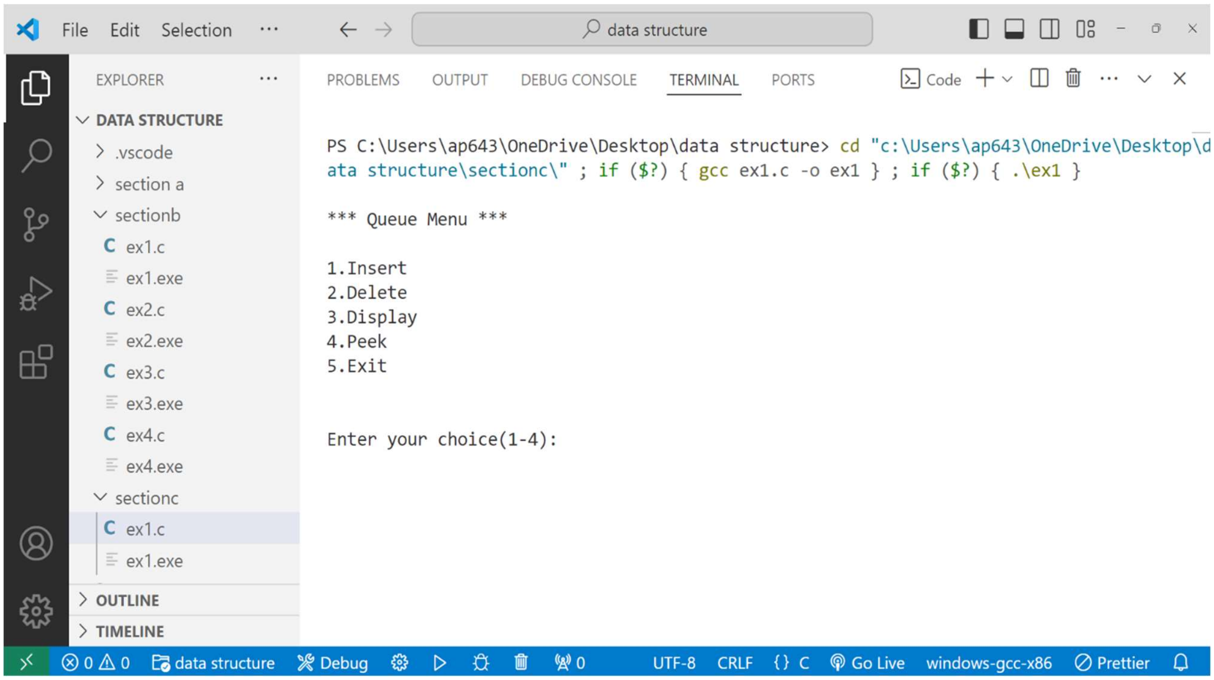
```
#include <stdio.h>

#include <stdlib.h>

#define N 50

int queue[N];

int front = -1;

int rear = -1;

void enqueue(){

int x;

printf("Enter the Element to be inserted:\n");

scanf("%d", &x);

if (rear == N - 1){

printf("Queue is Full!\n");

}

else if (front == -1 && rear == -1)

{

rear = front = 0;

queue[rear] = x;

}

else{

rear++;

queue[rear] = x;}
```

```
}
void dequeue(){
if (front == -1 && rear == -1){
printf("Queue is Empty!\n");
}
else if (front == rear){
front = rear = -1;
}
else{
front++;
}
}
void display(){
if (front == -1 && rear == -1){
printf("Queue is Empty\n");
}
else{
for (int i = front; i < rear + 1; i++){
printf("%d ", queue[i]);
}
}
printf("\n");
}
void peek(){
if (front == -1 && rear == -1){
printf("Queue is Empty\n");}
```

```
else{
printf("Front Element: %d", queue[front]);}
}
int main(){
int ch;
while (1){
printf("\n*** Queue Menu ***");
printf("\n\n1.Insert\n2.Delete\n3.Display\n4.Peek\n5.Exit\n");
printf("\n\nEnter your choice(1-4):");
scanf("%d", &ch);
switch (ch){
case 1:
enqueue();
break;
case 2:
dequeue();
break;
case 3:
display();
break;
case 4:
peek();
break;
case 5:
exit(0);
default:
```

```
printf("\nWrong Choice!!");}
}
return 0;
}
```

Output:



Experiment No.: 2

Program Description:

Implementation of Queue using Pointers.

Solution:

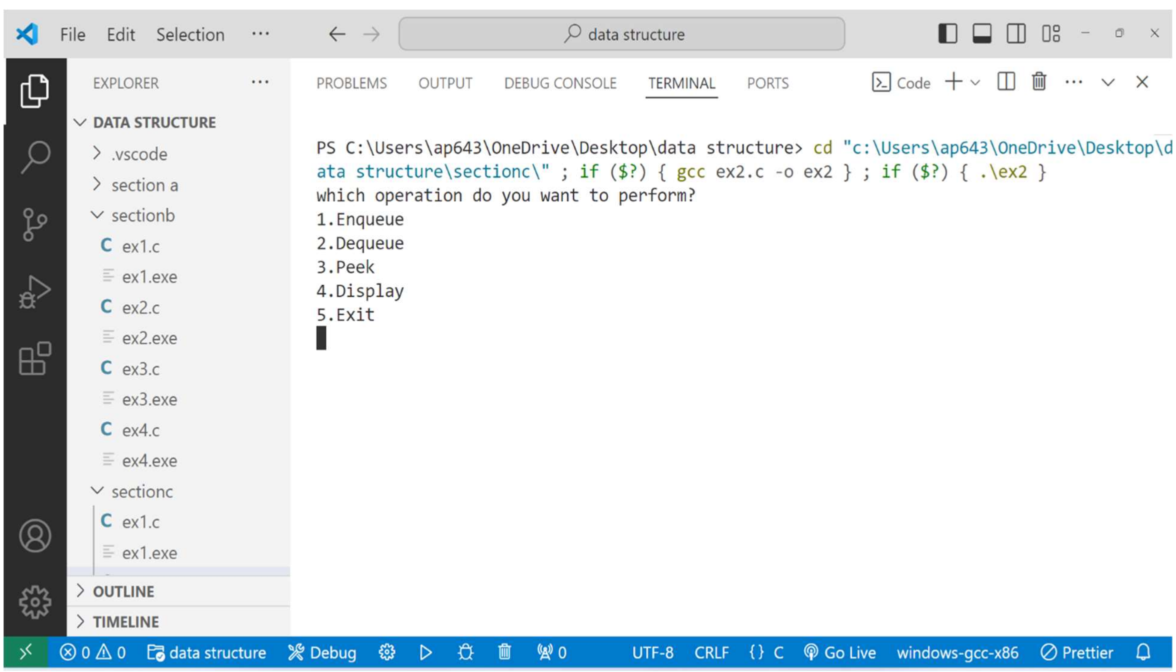
```
#include <stdio.h>

#include <stdlib.h>

struct Queue{

int data;

struct Queue *next;

};

struct Queue *front = NULL;

struct Queue *rear = NULL;

void enqueue(int x){

struct Queue *newNode = (struct Queue *)malloc(sizeof(struct Queue));

newNode->data = x;

newNode->next = NULL;

if (front == NULL && rear == NULL){

front = rear = newNode;

}

else{

rear->next = newNode;

rear = newNode;

}

}
```

```
void dequeue(){
struct Queue *temp = front;
if (front == NULL && rear == NULL){
printf("Queue is empty\n");
}
else{
printf("Element dequeued is %d\n", front->data);
front = front->next;
free(temp);
}
}
void peek(){
if (front == NULL && rear == NULL){
printf("Queue is empty\n");
}
else{
printf("Top Element is %d\n", front->data);
}
}
void display(){
struct Queue *temp = front;
if (front == NULL && rear == NULL)
{
printf("Queue is empty\n");
}
else{
```

```
while (temp != NULL){
printf("%d ", temp->data);
temp = temp->next;
}
}
printf("\n");
}
int main(){
int opt, value, popped = 0;
while (1){
printf("which operation do you want to perform?\n");
printf("1.Enqueue\n");
printf("2.Dequeue\n");
printf("3.Peek\n");
printf("4.Display\n");
printf("5.Exit\n");
scanf("%d", &opt);
switch (opt){
case 1:
printf("enter the value\n");
scanf("%d", &value);
enqueue(value);
break;
case 2:
dequeue();
break;
```

```
case 3:
peek();
break;
case 4:
display();
break;
case 5:
exit(0);
break;
}
}
return 0;}
```

Output:



Experiment No.: 3

Program Description:

Implementation of Circular Queue using Array.

Solution:

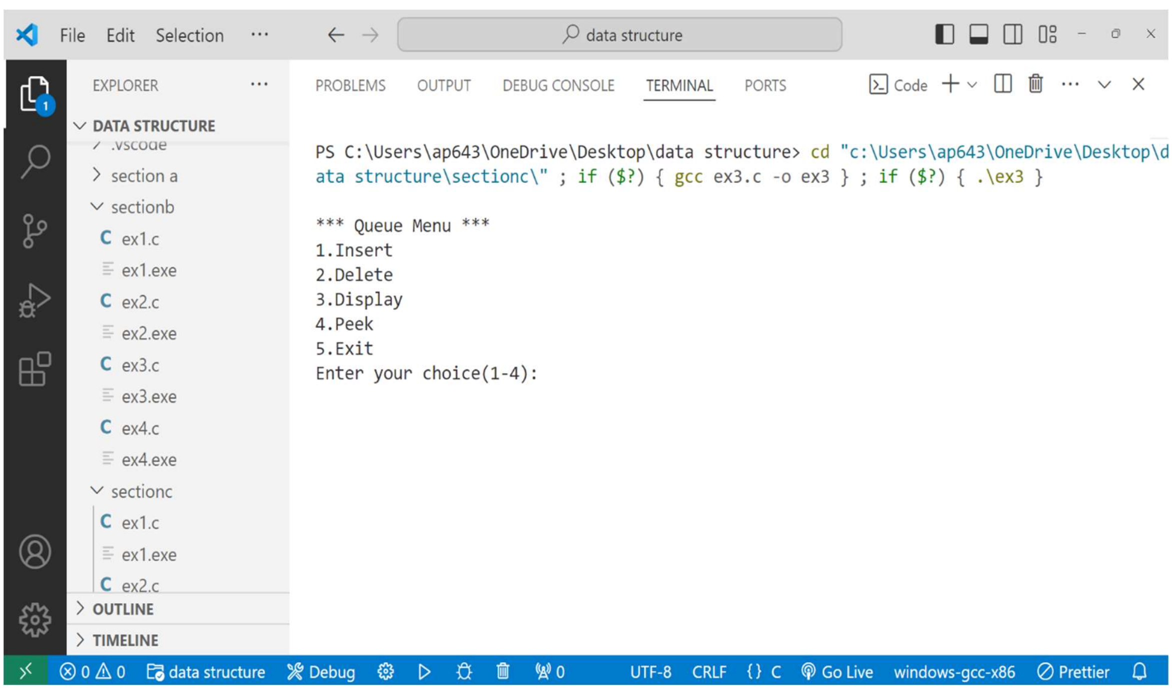
```
#include <stdio.h>

#include <stdlib.h>

#define N 50

int queue[N];

int front = -1;

int rear = -1;

void enqueue(){

int x;

printf("Enter data to be enqueued: ");

scanf("%d", &x);

if (front == -1 && rear == -1){

front = rear = 0;

queue[rear] = x;

}

else if ((rear + 1) % N == front){

printf("Queue is full\n");

}

else{

rear = (rear + 1) % N;

queue[rear] = x;

}

}
```

```
void dequeue(){
if ((front == -1) && (rear == -1)){
printf("\nQueue is underflow..\n");
}
else if (front == rear){
printf("\nThe Dequeued element is %d\n", queue[front]);
front = -1;
rear = -1;
}
else{
printf("\nThe Dequeued element is %d\n", queue[front]);
front = (front + 1) % N;}
}
void display(){
int i = front;
if (front == -1 && rear == -1){
printf("Queue is Empty\n");
}
else{
printf("Queue is: ");
while (i != rear){
printf("%d ", queue[i]);
i = (i + 1) % N;
}
printf("%d ", queue[rear]);
}
```

```
printf("\n");
}
void peek(){
if (front == -1 && rear == -1){
printf("Queue is Empty\n");
}
else{
printf("Front Element: %d", queue[front]);}
}
int main(){
int ch;
int x;
while (1){
printf("\n*** Queue Menu ***");
printf("\n1.Insert\n2.Delete\n3.Display\n4.Peek\n5.Exit");
printf("\nEnter your choice(1-4):");
scanf("%d", &ch);
switch (ch)
{
case 1:
enqueue();
break;
case 2:
dequeue();
break;
case 3:
```

```
display();
break;
case 4:
peek();
break;
case 5:
exit(0);
break;
default:
printf("\nWrong Choice!!");}
}
}
```

Output:



Section-D (Trees & Graphs)

Experiment No.: 1

Program Description:

Implementation of Binary Search Tree.

Solution:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct treeNode {
int data;
struct treeNode *left;
struct treeNode *right;
} treeNode;
treeNode* FindMin(treeNode *node) {
if (node == NULL) {
return NULL;
}
while (node->left != NULL) {
node = node->left;
}
return node;
}
treeNode* FindMax(treeNode *node) {
if (node == NULL) {
return NULL;
}
```

```
while (node->right != NULL) {
node = node->right;
}
return node;
}
treeNode* Insert(treeNode *node, int data) {
if (node == NULL) {
treeNode *temp = (treeNode *)malloc(sizeof(treeNode));
temp->data = data;
temp->left = temp->right = NULL;
return temp;
}
if (data < node->data) {
node->left = Insert(node->left, data);
} else if (data > node->data) {
node->right = Insert(node->right, data);
}
return node;
}
treeNode* Delete(treeNode *node, int data) {
if (node == NULL) {
printf("Element not found\n");
return NULL;
}
if (data < node->data) {
node->left = Delete(node->left, data);
```

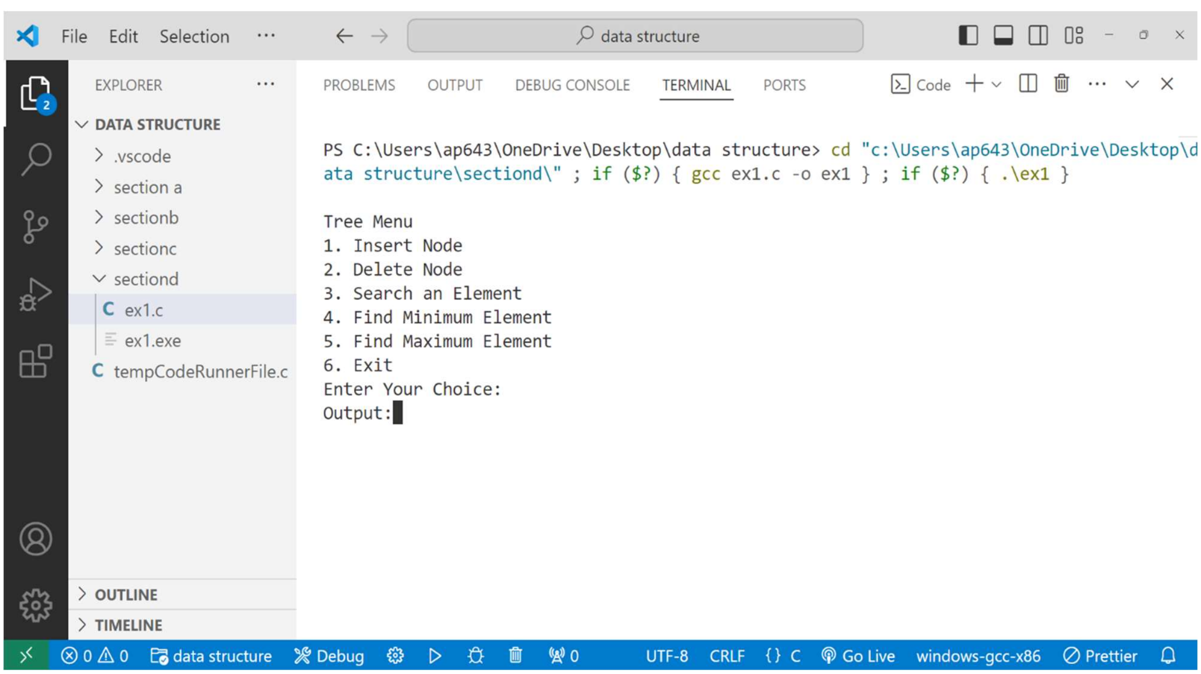
```
} else if (data > node->data) {
node->right = Delete(node->right, data);
} else {
if (node->left == NULL && node->right == NULL) {
free(node);
return NULL;
} else if (node->left == NULL) {
treeNode *temp = node->right;
free(node);
return temp;
} else if (node->right == NULL) {
treeNode *temp = node->left;
free(node);
return temp;
} else {
treeNode *temp = FindMin(node->right);
node->data = temp->data;
node->right = Delete(node->right, temp->data);
}
}
return node;
}
treeNode* Find(treeNode *node, int data) {
if (node == NULL) {
return NULL;
}
```

```
if (data < node->data) {
return Find(node->left, data);
} else if (data > node->data) {
return Find(node->right, data);
} else {
return node;
}
}
int main() {
treeNode *root = NULL;
treeNode *temp;
int choice, val;
while (1) {
printf("\nTree Menu");
printf("\n1. Insert Node");
printf("\n2. Delete Node");
printf("\n3. Search an Element");
printf("\n4. Find Minimum Element");
printf("\n5. Find Maximum Element");
printf("\n6. Exit");
printf("\nEnter Your Choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
printf("\nEnter value to insert: ");
scanf("%d", &val);
```

```
root = Insert(root, val);
printf("Insertion Successful\n");
break;
case 2:
printf("\nEnter value to delete: ");
scanf("%d", &val);
root = Delete(root, val);
break;
case 3:
printf("\nEnter value to search: ");
scanf("%d", &val);
temp = Find(root, val);
if (temp != NULL) {
printf("Element %d Found\n", val);
} else {
printf("Element %d Not Found\n", val);
}
break;
case 4:
temp = FindMin(root);
if (temp != NULL) {
printf("Minimum Element: %d\n", temp->data);
} else {
printf("Tree is Empty\n");
}
break;
```

```
case 5:
temp = FindMax(root);
if (temp != NULL) {
printf("Maximum Element: %d\n", temp->data);
} else {
printf("Tree is Empty\n");}
break;
case 6:
printf("Exiting Program...\n");
exit(0);
default:
printf("Invalid Choice! Please try again.\n");}
}
return 0;}
```

Output:



Experiment No.: 2

Program Description:

Conversion of BST PreOrder/PostOrder/InOrder.

Solution:

```
#include <stdio.h>

#include <stdlib.h>

typedef struct treeNode {

int data;

struct treeNode *left, *right;

} treeNode;

treeNode* createNode(int data) {

treeNode *newNode = (treeNode *)malloc(sizeof(treeNode));

newNode->data = data;

newNode->left = newNode->right = NULL;

return newNode;}

treeNode* insert(treeNode *root, int data) {

if (root == NULL)

return createNode(data);

if (data < root->data)

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

else if (data > root->data)

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

return root;

}

treeNode* findMin(treeNode *root) {

while (root && root->left != NULL)
```

```
root = root->left;
return root;
}
treeNode* findMax(treeNode *root) {
while (root && root->right != NULL)
root = root->right;
return root;
}
treeNode* deleteNode(treeNode *root, int data) {
if (root == NULL) {
printf("Element %d not found.\n", data);
return root;
}
if (data < root->data)
root->left = deleteNode(root->left, data);
else if (data > root->data)
root->right = deleteNode(root->right, data);
else{
if (root->left == NULL) {
treeNode *temp = root->right;
free(root);
return temp;
} else if (root->right == NULL) {
treeNode *temp = root->left;
free(root);
return temp;}
```

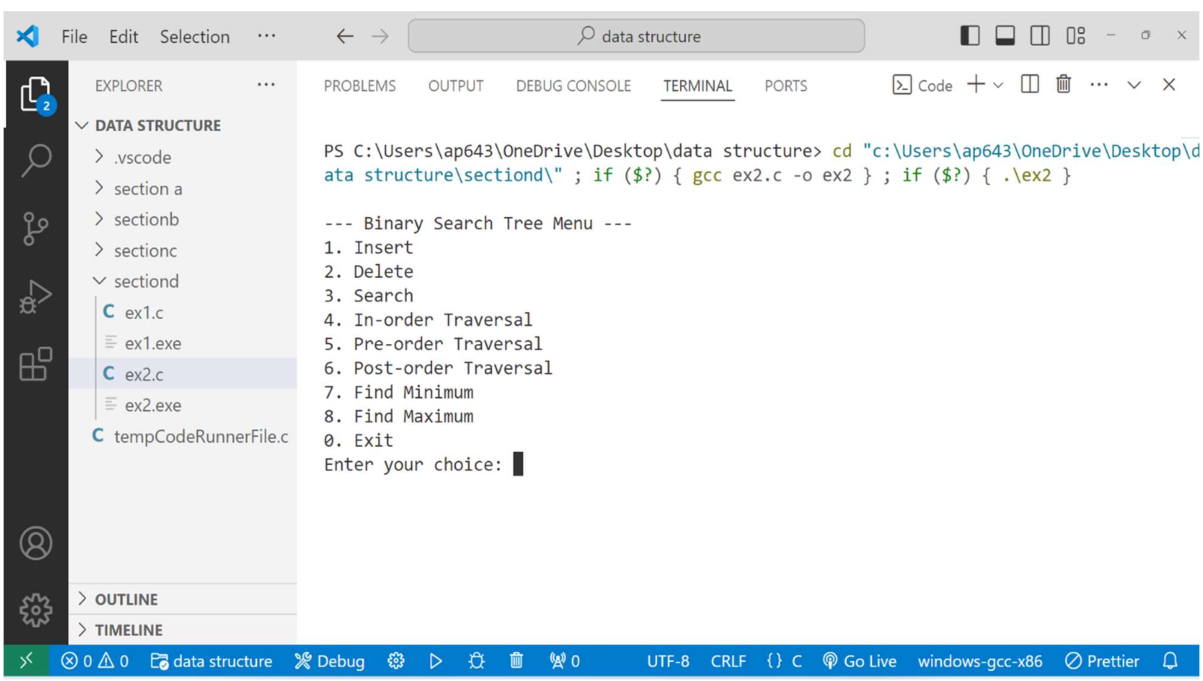
```
treeNode *temp = findMin(root->right);
root->data = temp->data;
root->right = deleteNode(root->right, temp->data);
}
return root;}
treeNode* search(treeNode *root, int data) {
if (root == NULL || root->data == data)
return root;
if (data < root->data)
return search(root->left, data);
return search(root->right, data);
}
void inOrder(treeNode *root) {
if (root == NULL) return;
inOrder(root->left);
printf("%d ", root->data);
inOrder(root->right);
}
void preOrder(treeNode *root) {
if (root == NULL) return;
printf("%d ", root->data);
preOrder(root->left);
preOrder(root->right);
}
void postOrder(treeNode *root) {
if (root == NULL) return;
```

```
postOrder(root->left);
postOrder(root->right);
printf("%d ", root->data);
}
int main() {
treeNode *root = NULL;
int choice, value;
while (1) {
printf("\n--- Binary Search Tree Menu ---\n");
printf("1. Insert\n2. Delete\n3. Search\n4. In-order Traversal\n");
printf("5. Pre-order Traversal\n6. Post-order Traversal\n");
printf("7. Find Minimum\n8. Find Maximum\n0. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
printf("Enter value to insert: ");
scanf("%d", &value);
root = insert(root, value);
printf("Inserted %d into the BST.\n", value);
break;
case 2:
printf("Enter value to delete: ");
scanf("%d", &value);
root = deleteNode(root, value);
break;
```

```
case 3:
printf("Enter value to search: ");
scanf("%d", &value);
if (search(root, value))
printf("Element %d found in the BST.\n", value);
else
printf("Element %d not found in the BST.\n", value);
break;
case 4:
printf("In-order Traversal: ");
inOrder(root);
printf("\n");
break;
case 5:
printf("Pre-order Traversal: ");
preOrder(root);
printf("\n");
break;
case 6:
printf("Post-order Traversal: ");
postOrder(root);
printf("\n");
break;
case 7:
if (root)
printf("Minimum element: %d\n", findMin(root)->data);
```

```
else
printf("The tree is empty.\n");
break;
case 8:
if (root)
printf("Maximum element: %d\n", findMax(root)->data);
else
printf("The tree is empty.\n");
break;
case 0:
printf("Exiting program. Goodbye!\n");
return 0;
default:
printf("Invalid choice! Please try again.\n");}}
}
```

Output:



Experiment No.: 3

Program Description:

Implementation of Kruskal Algorithm.

Solution:

```
#include <stdio.h>
#include <stdlib.h>
// Comparator function to use in sorting
int comparator(const void* p1, const void* p2){
const int(*x)[3] = p1;
const int(*y)[3] = p2;
return (*x)[2] - (*y)[2];
}
// Initialization of parent[] and rank[] arrays
void makeSet(int parent[], int rank[], int n)
{
for (int i = 0; i < n; i++) {
parent[i] = i;
rank[i] = 0;
}
}
// Function to find the parent of a node
int findParent(int parent[], int component){
if (parent[component] == component)
return component;
return parent[component]= findParent(parent, parent[component]);
}
```

```
// Function to unite two sets
void unionSet(int u, int v, int parent[], int rank[], int n)
{
// Finding the parents
u = findParent(parent, u);
v = findParent(parent, v);
if (rank[u] < rank[v]) {
parent[u] = v;
}
else if (rank[u] > rank[v]) {
parent[v] = u;
}
else {
parent[v] = u;
// Since the rank increases if
// the ranks of two sets are same
rank[u]++;
}
}
// Function to find the MST
void kruskalAlgo(int n, int edge[n][3]){
// First we sort the edge array in ascending order
// so that we can access minimum distances/cost
qsort(edge, n, sizeof(edge[0]), comparator);
int parent[n];
int rank[n];
```

```
// Function to initialize parent[] and rank[]
makeSet(parent, rank, n);
// To store the minimun cost
int minCost = 0;
printf(
"Following are the edges in the constructed MST\n");
for (int i = 0; i < n; i++) {
int v1 = findParent(parent, edge[i][0]);
int v2 = findParent(parent, edge[i][1]);
int wt = edge[i][2];
// If the parents are different that
// means they are in different sets so
// union them
if (v1 != v2) {
unionSet(v1, v2, parent, rank, n);
minCost += wt;
printf("%d -- %d == %d\n", edge[i][0],
edge[i][1], wt);
}
}
printf("Minimum Cost Spanning Tree: %d\n", minCost);
}
// Driver code
int main(){
int edge[5][3] = { { 0, 1, 10 },
{ 0, 2, 6 },
```

{ 0, 3, 5 },

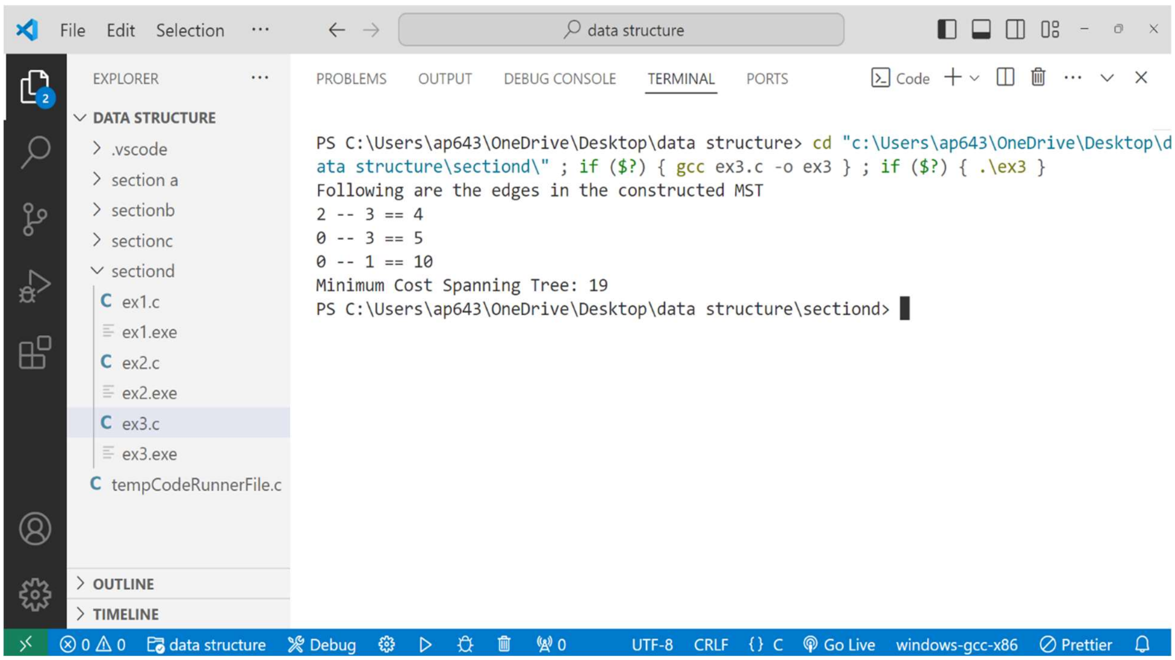
{ 1, 3, 15 },

{ 2, 3, 4 } };

kruskalAlgo(5, edge);

return 0;}

Output:



Experiment No.: 4

Program Description:

Implementation of Prim Algorithm

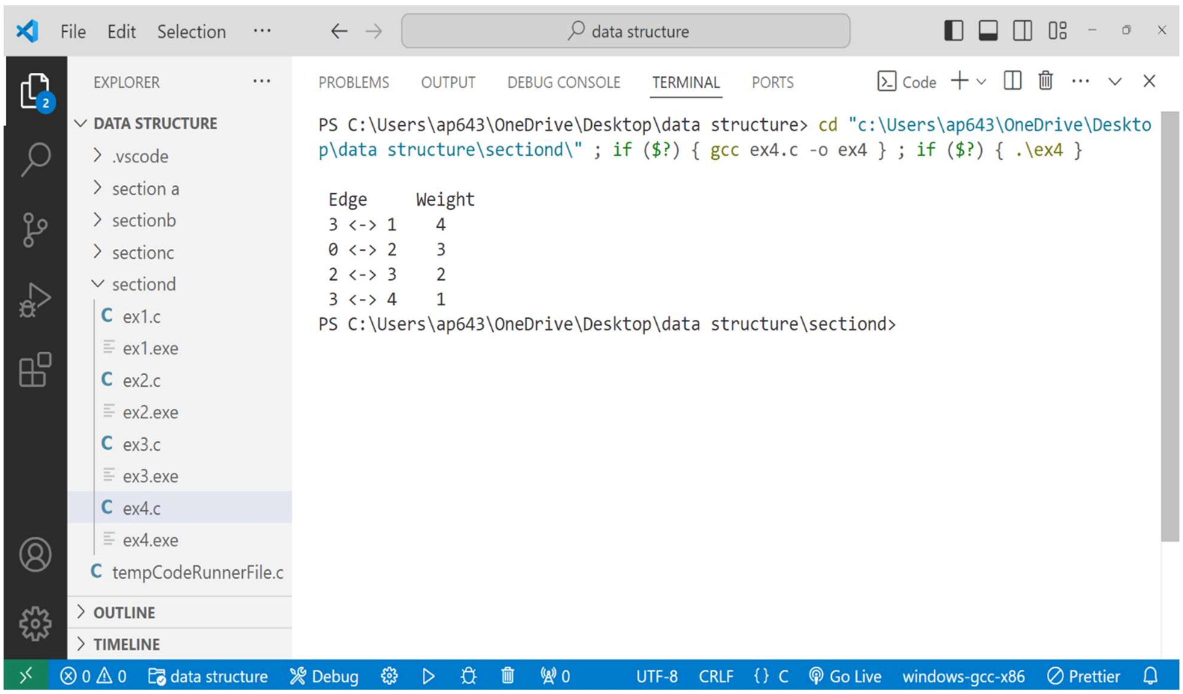
Solution:

```
#include <stdio.h>

#include <limits.h>

#define vertices 5

int minimum_key(int k[], int mst[]){

int minimum  = INT_MAX, min,i;

/*iterate over all vertices to find the vertex with minimum key-value*/

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

if (mst[i] == 0 && k[i] < minimum )

minimum = k[i], min = i;

return min;

}

/* create prim() method for constructing and printing the MST.

The g[vertices][vertices] is an adjacency matrix that defines the graph for MST.*/

void prim(int g[vertices][vertices]){

/* create array of size equal to total number of vertices for storing the MST*/

int parent[vertices];

/* create k[vertices] array for selecting an edge having minimum weight*/

int k[vertices];

int mst[vertices];

int i, count,edge,v; /*Here 'v' is the vertex*/

for (i = 0; i < vertices; i++){
```

```
k[i] = INT_MAX;

mst[i] = 0;

}

k[0] = 0; /*It select as first vertex*/

parent[0] = -1;   /* set first value of parent[] array to -1 to make it root of
MST*/

for (count = 0; count < vertices-1; count++){

/*select the vertex having minimum key and that is not added in the MST yet
from the set of vertices*/

edge = minimum_key(k, mst);

mst[edge] = 1;

for (v = 0; v < vertices; v++){

if (g[edge][v] && mst[v] == 0 && g[edge][v] <  k[v]){

parent[v]  = edge, k[v] = g[edge][v];

}

}

}

/*Print the constructed Minimum spanning tree*/

printf("\n Edge \t  Weight\n");

for (i = 1; i < vertices; i++)

printf(" %d <-> %d    %d \n", parent[i], i, g[i][parent[i]]);

}

int main(){

int g[vertices][vertices] = {{0, 0, 3, 0, 0},

{0, 0, 10, 4, 0},

{3, 10, 0, 2, 6},

{0, 4, 2, 0, 1},
```

```
{0, 0, 6, 1, 0},
};
prim(g);
return 0;
}
```

Output:



Experiment No.: 5

Program Description:

Implementation of Dijkstra Algorithm.

Solution:

```
#include <stdio.h>
#define INFINITY 9999
#define MAX 10
void Dijkstra(int Graph[MAX][MAX], int n, int start);
void Dijkstra(int Graph[MAX][MAX], int n, int start) {
int cost[MAX][MAX], distance[MAX], pred[MAX];
int visited[MAX], count, mindistance, nextnode, i, j;
// Creating cost matrix
for (i = 0; i < n; i++)
for (j = 0; j < n; j++)
if (Graph[i][j] == 0)
cost[i][j] = INFINITY;
else
cost[i][j] = Graph[i][j];
for (i = 0; i < n; i++) {
distance[i] = cost[start][i];
pred[i] = start;
visited[i] = 0;
}
distance[start] = 0;
visited[start] = 1;
count = 1;
```

```
while (count < n - 1) {
mindistance = INFINITY;
for (i = 0; i < n; i++)
if (distance[i] < mindistance && !visited[i]) {
mindistance = distance[i];
nextnode = i;
}
visited[nextnode] = 1;
for (i = 0; i < n; i++)
if (!visited[i])
if (mindistance + cost[nextnode][i] < distance[i]) {
distance[i] = mindistance + cost[nextnode][i];
pred[i] = nextnode;
}
count++;
}
// Printing the distance
for (i = 0; i < n; i++)
if (i != start) {
printf("\nDistance from source to %d: %d", i, distance[i]);
}
}
int main() {
int Graph[MAX][MAX], i, j, n, u;
n = 7;
```

Graph[0][0] = 0;

Graph[0][1] = 0;

Graph[0][2] = 1;

Graph[0][3] = 2;

Graph[0][4] = 0;

Graph[0][5] = 0;

Graph[0][6] = 0;

Graph[1][0] = 0;

Graph[1][1] = 0;

Graph[1][2] = 2;

Graph[1][3] = 0;

Graph[1][4] = 0;

Graph[1][5] = 3;

Graph[1][6] = 0;

Graph[2][0] = 1;

Graph[2][1] = 2;

Graph[2][2] = 0;

Graph[2][3] = 1;

Graph[2][4] = 3;

Graph[2][5] = 0;

Graph[2][6] = 0;

Graph[3][0] = 2;

Graph[3][1] = 0;

Graph[3][2] = 1;

Graph[3][3] = 0;

Graph[3][4] = 0;

Graph[3][5] = 0;

Graph[3][6] = 1;

Graph[4][0] = 0;

Graph[4][1] = 0;

Graph[4][2] = 3;

Graph[4][3] = 0;

Graph[4][4] = 0;

Graph[4][5] = 2;

Graph[4][6] = 0;

Graph[5][0] = 0;

Graph[5][1] = 3;

Graph[5][2] = 0;

Graph[5][3] = 0;

Graph[5][4] = 2;

Graph[5][5] = 0;

Graph[5][6] = 1;

Graph[6][0] = 0;

Graph[6][1] = 0;

Graph[6][2] = 0;

Graph[6][3] = 1;

Graph[6][4] = 0;

Graph[6][5] = 1;

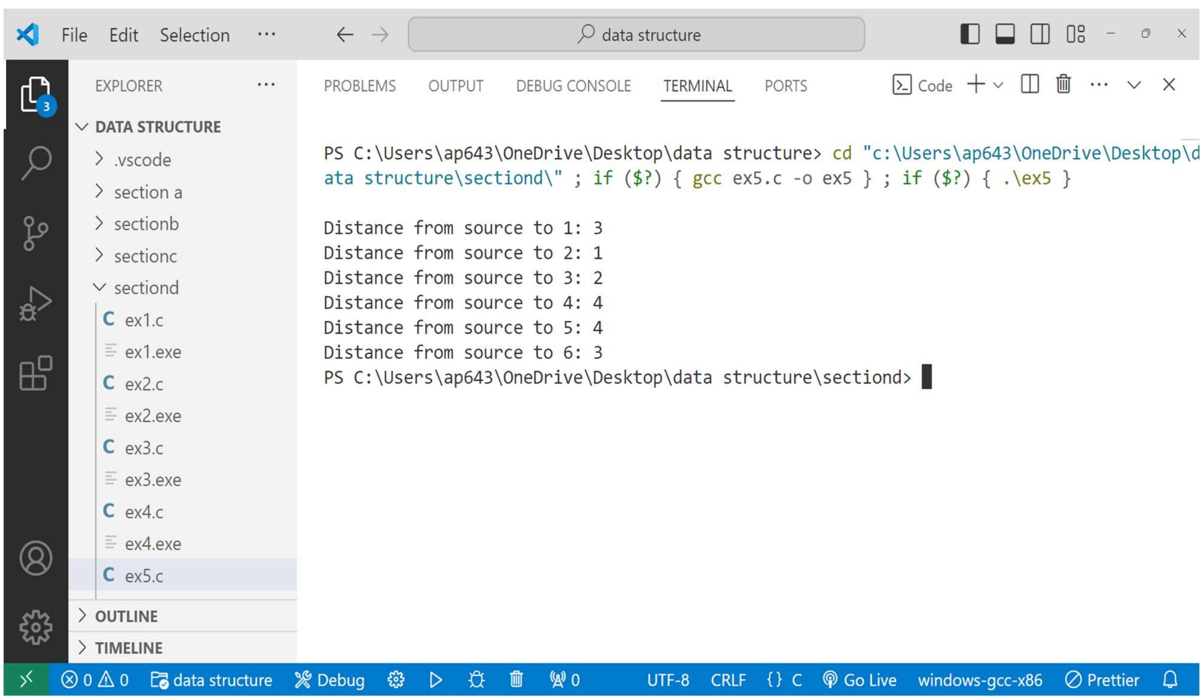
Graph[6][6] = 0;

u = 0;

Dijkstra(Graph, n, u);

return 0;}

Output:



Section-E (Sorting & Searching)

Experiment No.: 1

Program Description:

Implementation of Sorting

a. Bubble

b. Selection

c. Insertion

d. Quick

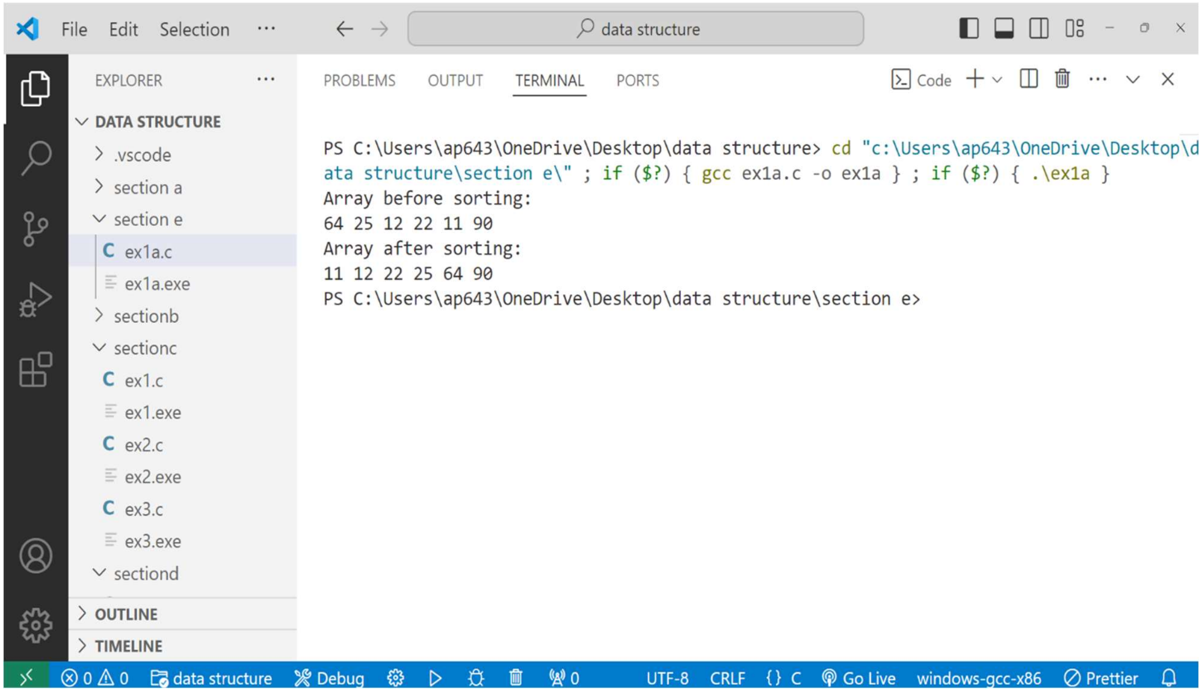
e. Merge

Solution(a):

```
#include <stdio.h>
void BubbleSort(int a[], int n)
{
int flag = 0;
for (int i = 0; i < n - 1; i++)
{
flag = 0;
for (int j = 0; j < n - i - 1; j++)
{
if (a[j] > a[j + 1])
{
int temp = a[j];
a[j] = a[j + 1];
a[j + 1] = temp;
```

```c
flag = 1;
}
}
if (flag == 0)
break;
}
}
void printArray(int a[], int n)
{
for (int i = 0; i < n; i++)
{
printf("%d ", a[i]);
}
}
int main()
{
int a[] = {64, 25, 12, 22, 11, 90};
int n = sizeof(a) / sizeof(a[0]);
printf("Array before sorting:\n");
printArray(a, n);
BubbleSort(a, n);
printf("\nArray after sorting:\n");
printArray(a, n);
return 0;
}
```

Output:



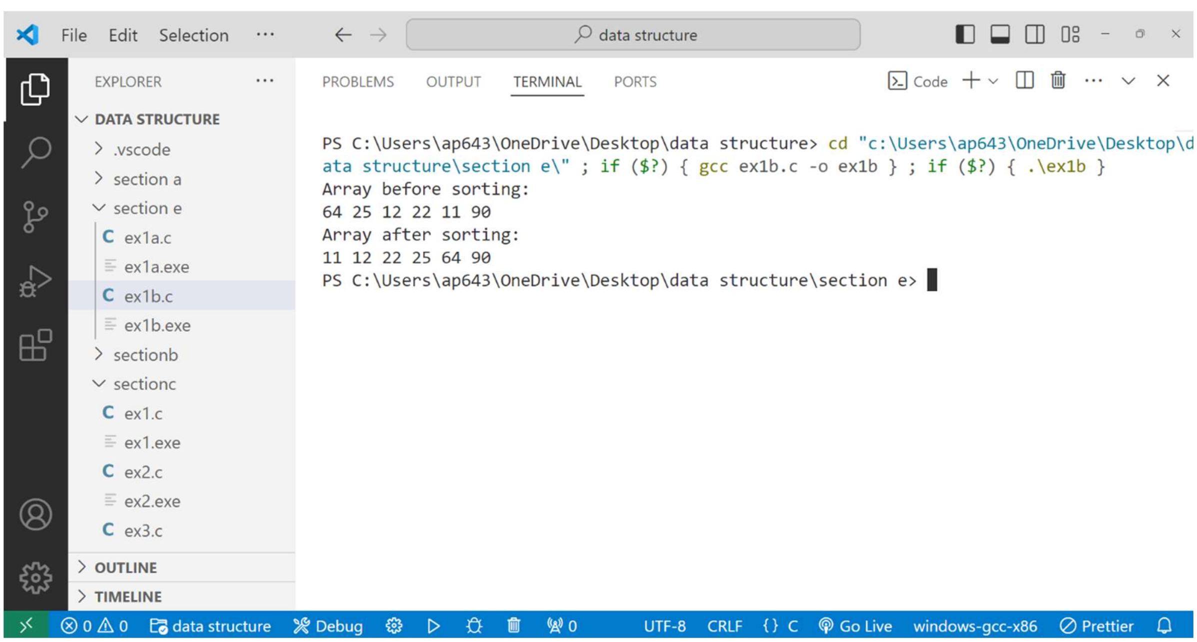
File Edit Selection ...
data structure
EXPLORER
DATA STRUCTURE
.vscode
section a
section e
ex1a.c
ex1a.exe
sectionb
sectionc
ex1.c
ex1.exe
ex2.c
ex2.exe
ex3.c
ex3.exe
sectiond
OUTLINE
TIMELINE
PROBLEMS OUTPUT TERMINAL PORTS
Code
PS C:\Users\ap643\OneDrive\Desktop\data structure> cd "c:\Users\ap643\OneDrive\Desktop\data structure\section e\" ; if ($?) { gcc ex1a.c -o ex1a } ; if ($?) { .\ex1a }
Array before sorting:
64 25 12 22 11 90
Array after sorting:
11 12 22 25 64 90
PS C:\Users\ap643\OneDrive\Desktop\data structure\section e>
0 0 data structure Debug 0 UTF-8 CRLF {} C Go Live windows-gcc-x86 Prettier

Solution(b):

```
#include <stdio.h>
void SelectionSort(int a[], int n)
{
int min;
for (int i = 0; i < n; i++)
{
min = i;
for (int j = i + 1; j < n; j++)
{
if (a[j] < a[min])
{
min = j;
}
}
if (min != i)
{
int temp = a[i];
a[i] = a[min];
a[min] = temp;
}
}
}
void printArray(int a[], int n)
{
for (int i = 0; i < n; i++)
```

```
{
printf("%d ", a[i]);
}
}
int main()
{
int a[] = {64, 25, 12, 22, 11, 90};
int n = sizeof(a) / sizeof(a[0]);
printf("Array before sorting:\n");
printArray(a, n);
SelectionSort(a, n);
printf("\nArray after sorting:\n");
printArray(a, n);
return 0;
}
```

Output:

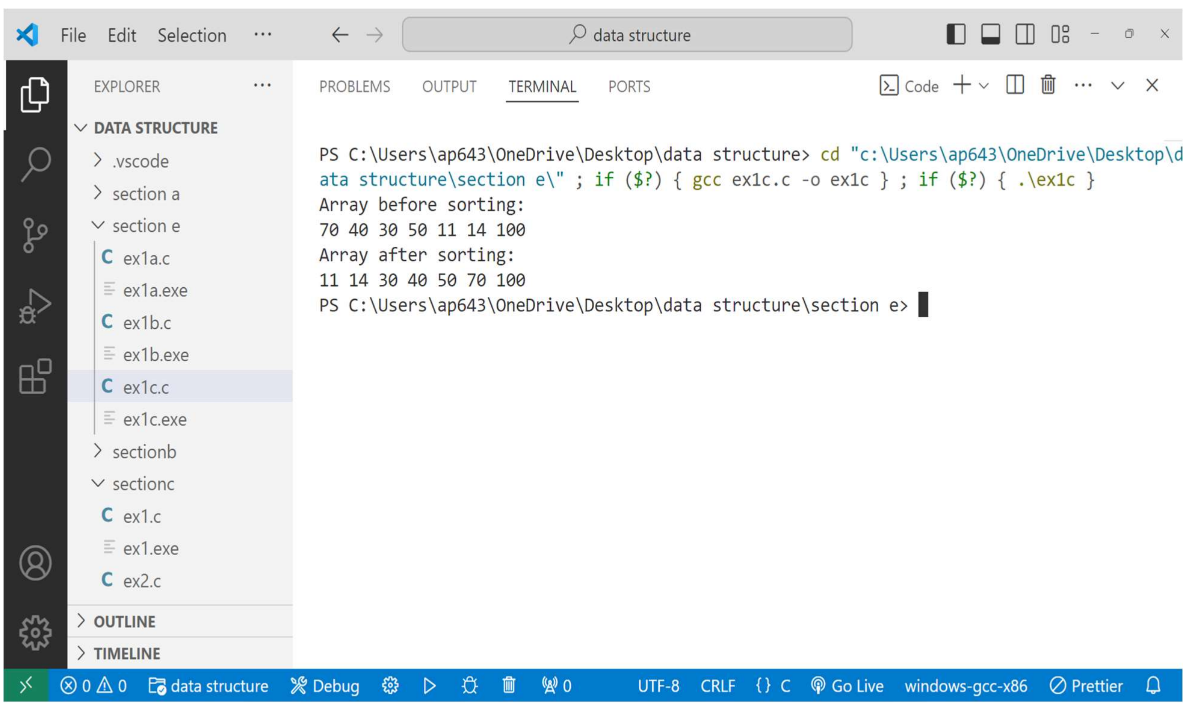


Solution(c):

```
#include <stdio.h>
void insertionSort(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;
}
}
void printArray(int a[], int n)
{
for (int i = 0; i < n; i++)
{
printf("%d ", a[i]);
}
}
```

```
int main()
{
int a[] = {70,40,30,50,11,14,100};
int n = sizeof(a) / sizeof(a[0]);
printf("Array before sorting:\n");
printArray(a, n);
insertionSort(a, n);
printf("\nArray after sorting:\n");
printArray(a, n);
return 0;}
```

Output:

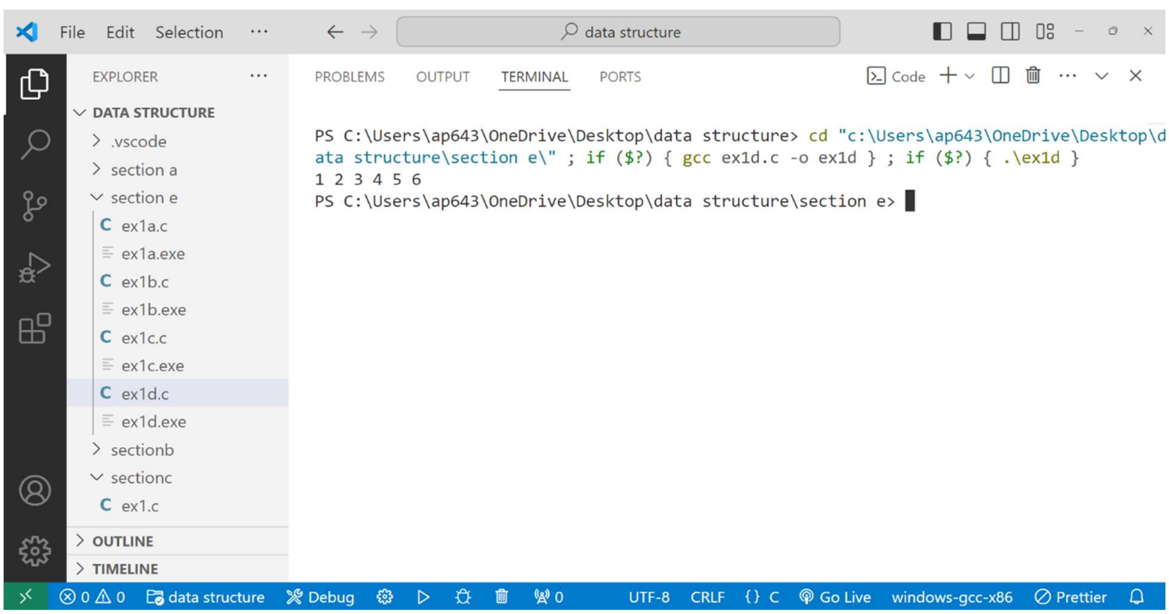


Solution (d):

```
#include <stdio.h>
void swap(int *a, int *b){
int temp = *a;
*a = *b;
*b = temp;
}
int Partition(int a[], int low, int high){
int pivot = a[low];
int start = low;
int end = high;
while (start < end){
while (a[start] <= pivot)
start++;
while (a[end] > pivot)
end--;
if (start < end){
swap(&a[start], &a[end]);
}
}
swap(&a[low], &a[end]);
return end;
}
void Quicksort(int a[], int low, int high)
{
if (low < high){
```

```
int pivot = Partition(a, low, high);
Quicksort(a, low, pivot - 1);
Quicksort(a, pivot + 1, high);
}
}
int main(){
int a[] = {5, 2, 4, 6, 1, 3};
int n = sizeof(a) / sizeof(a[0]);
Quicksort(a, 0, n - 1);
for (int i = 0; i < n; i++)
printf("%d ", a[i]);
return 0;}
```

Output:



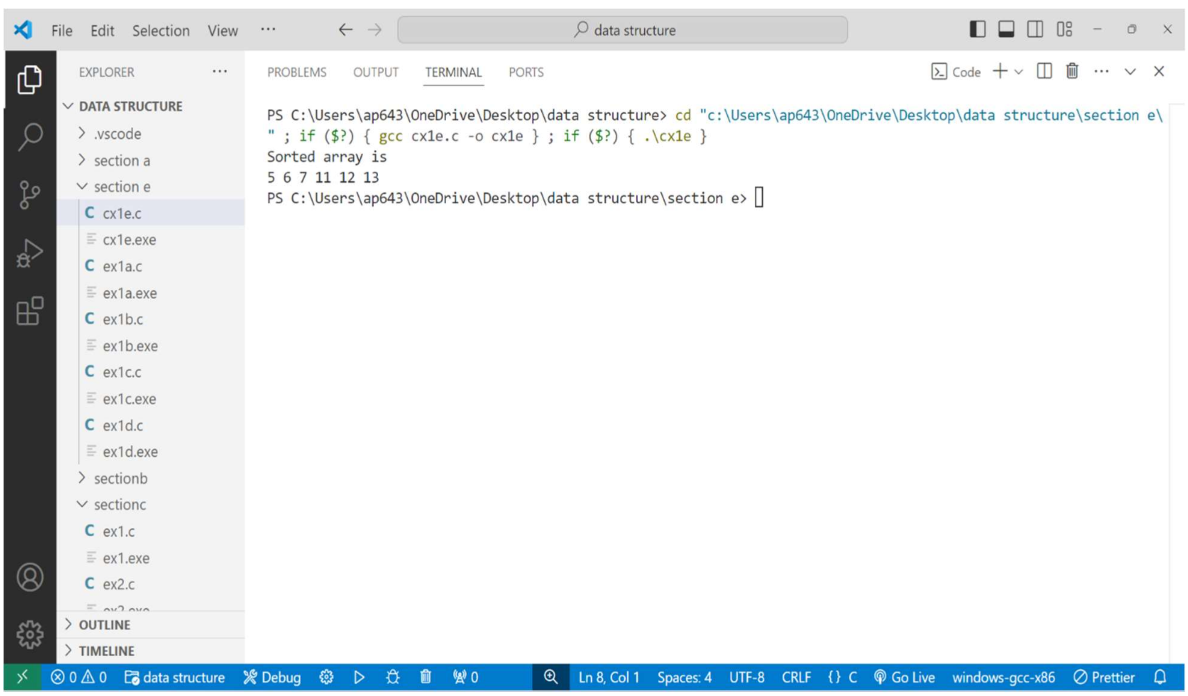
Solution (e):

```
#include <stdio.h>
void Merge(int arr[], int beg, int mid, int end)
{
int i, j, k;
int n1 = mid - beg + 1;
int n2 = end - mid;
int L[n1], R[n2];
for (i = 0; i < n1; i++)
L[i] = arr[beg + i];
for (j = 0; j < n2; j++)
R[j] = arr[mid + 1 + j];
i = 0;
j = 0;
k = beg;
while (i < n1 && j < n2)
{
if (L[i] <= R[j])
{
arr[k] = L[i];
i++;
}
else
{
arr[k] = R[j];
j++;
```

```
}
k++;
}
while (i < n1)
{
arr[k] = L[i];
i++;
k++;
}
while (j < n2)
{
arr[k] = R[j];
j++;
k++;
}
}
void MergeSort(int arr[], int beg, int end)
{
if (beg < end)
{
int mid = (beg + end) / 2;
MergeSort(arr, beg, mid);
MergeSort(arr, mid + 1, end);
Merge(arr, beg, mid, end);
}
}
```

```
int main()

{

int arr[] = {12, 11, 13, 5, 6, 7};

int n = sizeof(arr) / sizeof(arr[0]);

MergeSort(arr, 0, n - 1);

printf("Sorted array is \n");

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

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

return 0;}
```

Output:



Experiment No.: 2

Program Description:

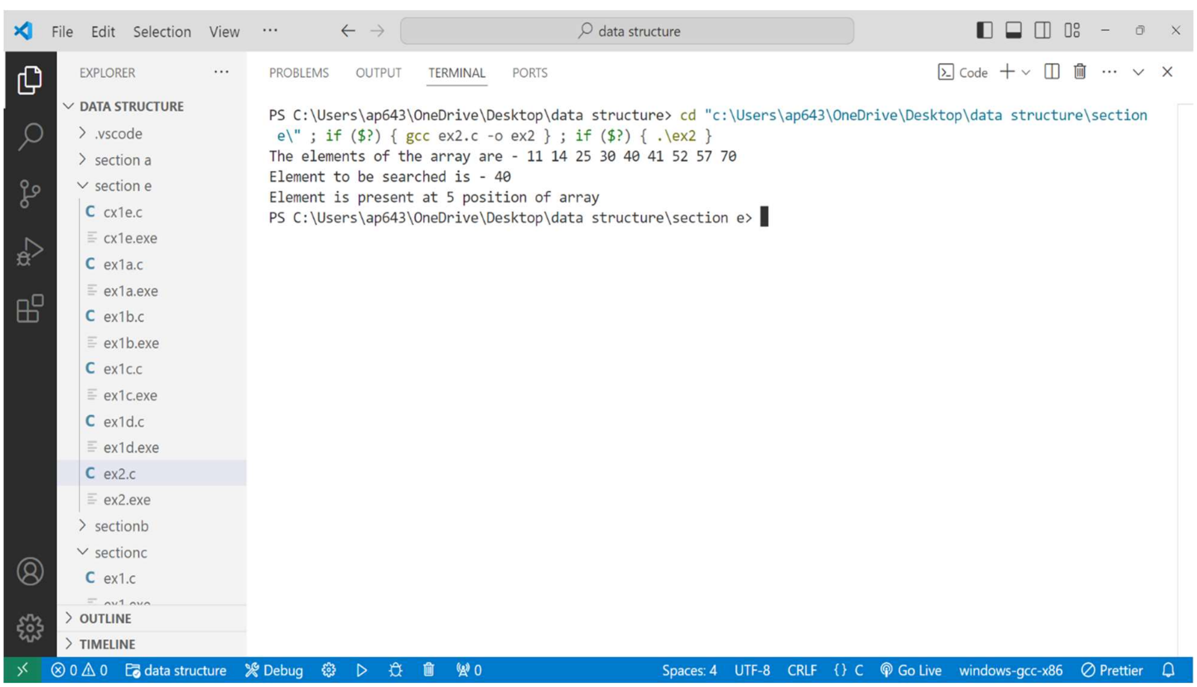
Implementation of Binary Search on a list of numbers stored in an Array.

Solution:

```
#include <stdio.h>
int binarySearch(int a[], int beg, int end, int val)
{
int mid;
if(end >= beg)
{       mid = (beg + end)/2;
if(a[mid] == val)
{
return mid+1;
}
else if(a[mid] < val)
{
return binarySearch(a, mid+1, end, val);
}
else
{
return binarySearch(a, beg, mid-1, val);
}
}
return -1;
}
int main() {
```

```
int a[] = {11, 14, 25, 30, 40, 41, 52, 57, 70};
int val = 40;
int n = sizeof(a) / sizeof(a[0]);
int res = binarySearch(a, 0, n-1, val);
printf("The elements of the array are - ");
for (int i = 0; i < n; i++)
printf("%d ", a[i]);
printf("\nElement to be searched is - %d", val);
if (res == -1)
printf("\nElement is not present in the array");
else
printf("\nElement is present at %d position of array", res);
return 0;
}
```

Output:



Experiment No.: 3

Program Description:

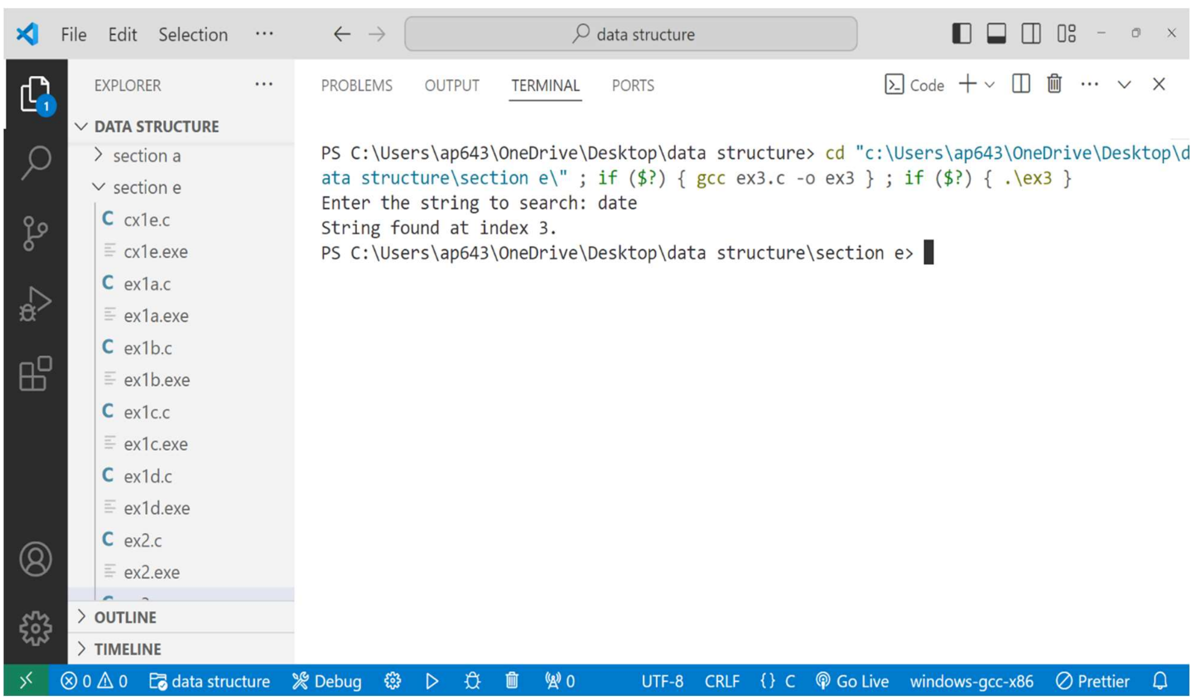
Implementation of Binary Search on a list of strings stored in an Array

Solution:

```
#include <stdio.h>
#include <string.h>
int binarySearch(char *arr[], int size, const char *target) {
int left = 0, right = size - 1;
while (left <= right) {
int mid = left + (right - left) / 2;
int cmp = strcmp(arr[mid], target);
if (cmp == 0) {
return mid;
} else if (cmp < 0) {
left = mid + 1;
} else {
right = mid - 1;
}
}
return -1;
}
int main() {
char *arr[] = {"apple", "banana", "cherry", "date", "fig", "grape"};
int size = sizeof(arr) / sizeof(arr[0]);
char target[50];
printf("Enter the string to search: ");
```

```
scanf("%s", target);

int result = binarySearch(arr, size, target);

if (result != -1) {

printf("String found at index %d.\n", result);

} else {

printf("String not found.\n");

}

return 0;

}
```

Output:



Experiment No.: 3

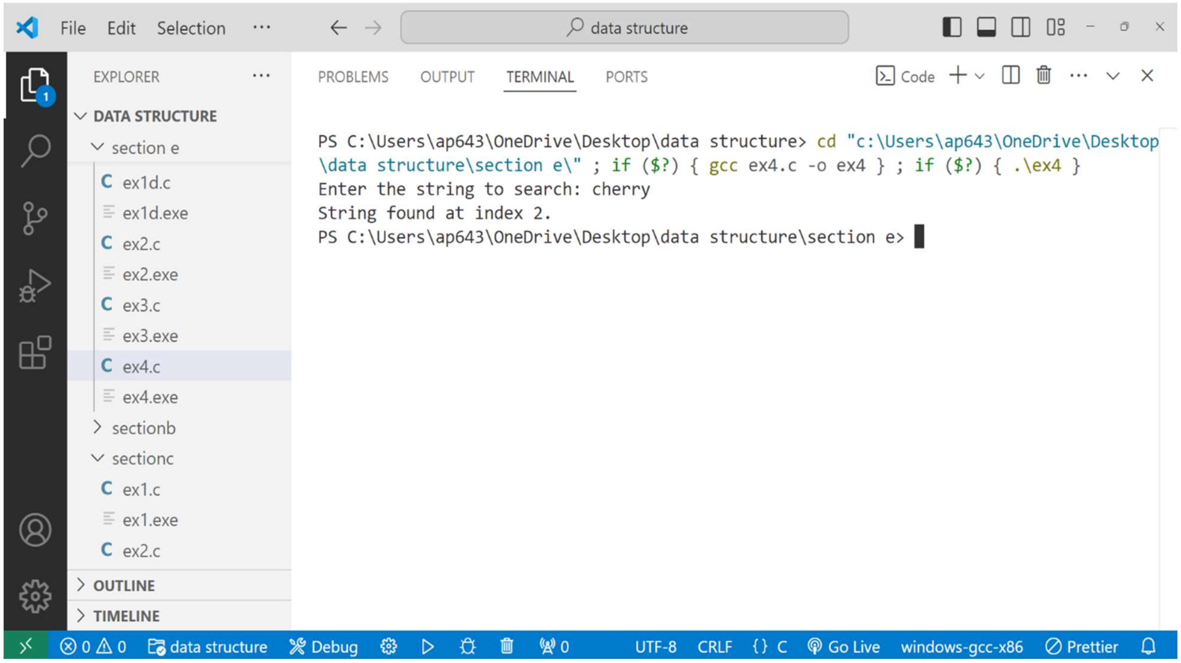
Program Description:

Implementation of Linear Search on a list of strings stored in an Array

Solution:

```
#include <stdio.h>
#include <string.h>
int linearSearch(char *arr[], int size, const char *target) {
for (int i = 0; i < size; i++) {
if (strcmp(arr[i], target) == 0) {
return i;}
}
return -1;}
int main() {
char *arr[] = {"apple", "banana", "cherry", "date", "fig", "grape"};
int size = sizeof(arr) / sizeof(arr[0]);
char target[50];
printf("Enter the string to search: ");
scanf("%s", target);
int result = linearSearch(arr, size, target);
if (result != -1) {
printf("String found at index %d.\n", result);
} else {
printf("String not found.\n");
}
return 0;
}
```

Output:


File Edit Selection ···
data structure
EXPLORER
DATA STRUCTURE
section e
ex1d.c
ex1d.exe
ex2.c
ex2.exe
ex3.c
ex3.exe
ex4.c
ex4.exe
sectionb
sectionc
ex1.c
ex1.exe
ex2.c
OUTLINE
TIMELINE
PROBLEMS OUTPUT TERMINAL PORTS
Code
PS C:\Users\ap643\OneDrive\Desktop\data structure> cd "c:\Users\ap643\OneDrive\Desktop\data structure\section e\" ; if ($?) { gcc ex4.c -o ex4 } ; if ($?) { .\ex4 }
Enter the string to search: cherry
String found at index 2.
PS C:\Users\ap643\OneDrive\Desktop\data structure\section e>
0 0 data structure Debug 0 UTF-8 CRLF {} C Go Live windows-gcc-x86 Prettier