DSA - All 12 Practicals: Full Code + Line-by-Line Explanation (Easy Language)

Author: ChatGPT — explanations written in simple, viva-friendly language.

Program 1: Right-angle triangle with numbers

Code:

```
#include <stdio.h>

int main() {
    int n, i, j;
    printf("Enter number of rows: ");
    scanf("%d", &n);
    for(i=1; i<=n; i++) {
        for(j=1; j<=i; j++) printf("%d ", j);
        printf("\n");
    }
    return 0;
}
```

Line-by-line explanation:

1. #include <stdio.h> - includes standard input/output functions (printf, scanf).
2. int main() { - program execution starts here.
3. int n, i, j; - declare variables: n = number of rows, i and j are loop counters.
4. printf("Enter number of rows: "); - asks the user to input how many rows to print.
5. scanf("%d", &n); - reads the integer entered by the user into n.
6. for(i=1; i<=n; i++) { - outer loop: repeats for each row from 1 to n.
7. for(j=1; j<=i; j++) printf("%d ", j); - inner loop: prints numbers 1..i for the current row.
8. printf("\n"); - prints newline to move to the next row after the inner loop finishes.
9. } - end of outer loop.
10. return 0; - indicates successful program termination.

Program 2: Diamond shape with numbers

Code:

```
#include <stdio.h>

int main() {
    int n, i, j, space;
    printf("Enter number of rows: ");
    scanf("%d", &n);
    for(i=1;i<=n;i++){
        for(space=i;space<n;space++) printf(" ");
        for(j=1;j<=2*i-1;j++) printf("%d", j);
        printf("\n");
    }
    for(i=n-1;i>=1;i--){
        for(space=n;space>i;space--) printf(" ");
        for(j=1;j<=2*i-1;j++) printf("%d", j);
        printf("\n");
    }
    return 0;
}
```

Line-by-line explanation:

1. #include <stdio.h> - include I/O functions.
2. int main() { - start.
3. int n, i, j, space; - n: rows; i,j: counters; space: for leading spaces.
4. printf/scanf to read n.
5. for(i=1;i<=n;i++){ - loop for top half of diamond.
6. for(space=i;space<n;space++) printf(" "); - print spaces to center pyramid for current row.
7. for(j=1;j<=2*i-1;j++) printf("%d", j); - print numbers forming the row (width 2*i-1).
8. printf("\n"); - newline.
9. } - end top half.
10. for(i=n-1;i>=1;i--){ - loop for bottom half in reverse.
11. for(space=n;space>i;space--) printf(" "); - leading spaces for bottom rows.
12. for(j=1;j<=2*i-1;j++) printf("%d", j); - numbers for bottom row.
13. printf("\n"); - newline.
14. }
15. return 0; - finish.

Program 3: Pyramid with * and Alphabets

Code:

```
#include <stdio.h>

int main(){
    int n,i,j,space;
    printf("Enter number of rows: ");
    scanf("%d",&n);

    printf("\nPyramid with *\n");
    for(i=1;i<=n;i++){
        for(space=i;space<n;space++) printf(" ");
        for(j=1;j<=2*i-1;j++) printf("*");
        printf("\n");
    }

    printf("\nPyramid with alphabets\n");
    char ch='A';
    for(i=1;i<=n;i++){
        for(space=i;space<n;space++) printf(" ");
        for(j=1;j<=2*i-1;j++) printf("%c", ch);
        printf("\n");
        ch++;
    }
    return 0;
}
```

Line-by-line explanation:

1. #include <stdio.h> - I/O functions.
2. int main(){ - start.
3. int n,i,j,space; - declare variables.
4. printf/scanf to read n.
5. Print header for star pyramid.
6. for(i=1;i<=n;i++){ - loop rows for star pyramid.
7. for(space=i;space<n;space++) printf(" "); - leading spaces.
8. for(j=1;j<=2*i-1;j++) printf("*"); - print stars for width 2*i-1.
9. printf("\n"); - newline.
10. }
11. Print header for alphabet pyramid.
12. char ch='A'; - start with 'A'.
13. for(i=1;i<=n;i++){ - each row uses same letter ch.
14. for(space...) ... - leading spaces.
15. for(j=1;j<=2*i-1;j++) printf("%c", ch); - print letter ch repeated.
16. printf("\n"); ch++; - newline and next letter for next row.
17. }
18. return 0; - finish.

Program 4: Binary search for name

Code:

```
#include <stdio.h>
#include <string.h>

int main(){
    char names[5][20] = {"Amit","Bhavesh","Chetan","Deepak","Esha"};
    char key[20];
    int low=0,high=4,mid,found=0;
    printf("Enter name to search: ");
    scanf("%s",key);
    while(low<=high){
        mid=(low+high)/2;
        int cmp=strcmp(names[mid],key);
        if(cmp==0){found=1;break;}
        else if(cmp<0) low=mid+1;
        else high=mid-1;
    }
    if(found) printf("Name found!\n");
    else printf("Name not found.\n");
    return 0;
}
```

Line-by-line explanation:

1. #include <stdio.h> and #include <string.h> - for I/O and strcmp().
2. char names[5][20] = {...}; - a sorted list of 5 names stored as strings.
3. char key[20]; - buffer for input name.
4. int low=0, high=4, mid, found=0; - initialize binary search indices and found flag.
5. scanf to read key.
6. while(low<=high){ mid=(low+high)/2; int cmp=strcmp(names[mid],key); ... } - standard binary searc
   - cmp == 0 => found
   - cmp < 0 => names[mid] < key => search right half (low = mid+1)
   - cmp > 0 => search left half (high = mid-1)
7. if(found) print found else not found.
8. return 0;

Program 5: Bubble sort and Selection sort

Code:

```
#include <stdio.h>

void bubble(int a[],int n){
    for(int i=0;i<n-1;i++)
        for(int j=0;j<n-i-1;j++)
            if(a[j]>a[j+1]){
                int t=a[j];a[j]=a[j+1];a[j+1]=t;
            }
}

void selection(int a[],int n){
    for(int i=0;i<n-1;i++){
        int min=i;
        for(int j=i+1;j<n;j++)
            if(a[j]<a[min]) min=j;
        int t=a[i];a[i]=a[min];a[min]=t;
    }
}

int main(){
    int a[5]={5,3,2,4,1};
    bubble(a,5);
    printf("Bubble sorted: ");
    for(int i=0;i<5;i++) printf("%d ",a[i]);
    printf("\n");

    int b[5]={5,3,2,4,1};
    selection(b,5);
    printf("Selection sorted: ");
    for(int i=0;i<5;i++) printf("%d ",b[i]);
    return 0;
}
```

Line-by-line explanation:

1. #include <stdio.h> - I/O.
2. bubble function: outer loop for passes, inner loop compares adjacent elements and swaps if a[j] >
3. selection function: for each i, find index min of smallest element in remaining array and swap wi
4. main: shows arrays, calls both sorts, and prints sorted results.

Program 6: Student database using array of structures

Code (full code as in your file):

```
#include <stdio.h>
#include <string.h>

struct student{
    int roll;
    char name[20],dept[20],subject[20];
    int marks;
} s[10];
int n=0;
void add(){
    printf("Enter roll, name, dept, subject, marks: ");
    scanf("%d%s%s%s%d",&s[n].roll,s[n].name,s[n].dept,s[n].subject,&s[n].marks);
    n++;
}

void display(){
    for(int i=0;i<n;i++)
        printf("%d %s %s %s %d\n",s[i].roll,s[i].name,s[i].dept,s[i].subject,s[i].marks);
}

void search(){
    int r;printf("Enter roll to search: ");scanf("%d",&r);
    for(int i=0;i<n;i++)
        if(s[i].roll==r){
            printf("Found: %s %s %s %d\n",s[i].name,s[i].dept,s[i].subject,s[i].marks);
            return;
        }
    printf("Not found\n");
}

void delete(){
    int r;printf("Enter roll to delete: ");scanf("%d",&r);
    for(int i=0;i<n;i++)
        if(s[i].roll==r){
            for(int j=i;j<n-1;j++) s[j]=s[j+1];
            n--;printf("Deleted\n");return;
        }
    printf("Not found\n");
}

int main(){
    int ch;
    do{
        printf("\n1.Add 2.Delete 3.Search 4.Display 5.Exit: ");
        scanf("%d",&ch);
        switch(ch){
            case 1:add();break;
            case 2:delete();break;
            case 3:search();break;
            case 4:display();break;
        }
    }while(ch!=5);
    return 0;
}
```

Line-by-line explanation:

1. #include <stdio.h>, #include <string.h> - I/O and string functions.
2. struct student{...} s[10]; - defines a student record and array of 10 students.
3. int n=0; - keeps current count of records.
4. add(): prompts and reads roll, name, dept, subject, marks into s[n], then increments n.
5. display(): loops from 0 to n-1 and prints each student's details.
6. search(): asks roll number, looks for matching s[i].roll and prints details if found.
7. delete(): finds matching roll, shifts subsequent records left to overwrite, decrements n.

8. main(): shows a menu in loop allowing add, delete, search, display until user chooses Exit (5).

Program 7: Stack using array

Code:

```
#include <stdio.h>
#define SIZE 5
int stack[SIZE], top=-1;

void push(int val){
    if(top==SIZE-1) printf("Stack full\n");
    else stack[++top]=val;
}

void pop(){
    if(top==-1) printf("Stack empty\n");
    else printf("Popped %d\n",stack[top--]);
}

void display(){
    if(top==-1) printf("Stack empty\n");
    else{
        for(int i=top;i>=0;i--) printf("%d ",stack[i]);
        printf("\n");
    }
}

int main(){
    int ch,val;
    do{
        printf("\n1.Push 2.Pop 3.Display 4.Exit: ");
        scanf("%d",&ch);
        switch(ch){
            case 1: printf("Enter value: ");scanf("%d",&val);push(val);break;
            case 2: pop();break;
            case 3: display();break;
        }
    }while(ch!=4);
    return 0;
}
```

Line-by-line explanation:

1. #define SIZE 5 sets maximum stack size.
2. stack array and top index -1 means empty.
3. push: check overflow (top==SIZE-1), otherwise increment top and store value.
4. pop: check underflow (top==-1), otherwise print and decrement top.
5. display: print elements from top to 0.
6. main: menu-driven operations for push/pop/display.

Program 8: Decimal to binary using stack

Code:

```
#include <stdio.h>
#define SIZE 20
int stack[SIZE],top=-1;

void push(int val){ stack[++top]=val; }
int pop(){ return stack[top--]; }

int main(){
    int n; printf("Enter decimal: "); scanf("%d",&n);
    while(n>0){ push(n%2); n/=2; }
    printf("Binary: ");
    while(top!=-1) printf("%d",pop());
    return 0;
}
```

Line-by-line explanation:

1. read decimal n.
2. while n>0: push remainder n%2 (bit) and divide n by 2.
3. popping prints bits in correct order (most significant last popped first).

Program 9: Linked list insert at front, delete from end

Code:

```
#include <stdio.h>
#include <stdlib.h>

struct node{int data;struct node*next;};
struct node*head=NULL;

void insert_front(int val){
    struct node*newnode=malloc(sizeof(struct node));
    newnode->data=val;newnode->next=head;head=newnode;
}

void delete_end(){
    if(head==NULL) printf("List empty\n");
    else if(head->next==NULL){free(head);head=NULL;}
    else{
        struct node*temp=head;
        while(temp->next->next!=NULL) temp=temp->next;
        free(temp->next);temp->next=NULL;
    }
}

void display(){
    struct node*t=head;
    while(t){printf("%d ",t->data);t=t->next;}
    printf("\n");
}

int main(){
    insert_front(10);insert_front(20);insert_front(30);
    display();
    delete_end();display();
    return 0;
}
```

Line-by-line explanation:
1. struct node defines a node with data and next pointer.
2. head initialized to NULL (empty list).
3. insert_front: allocate node, set data, next points to current head, update head to new node.
4. delete_end: handle empty list, single-node list, or traverse to second-last node and remove last
5. display: iterate and print node data.

Program 10: Linked list insert at end, delete from front

Code:

```
#include <stdio.h>
#include <stdlib.h>

struct node{int data;struct node*next;};
struct node*head=NULL;

void insert_end(int val){
    struct node*newnode=malloc(sizeof(struct node));
    newnode->data=val;newnode->next=NULL;
    if(head==NULL) head=newnode;
    else{
        struct node*t=head;
        while(t->next) t=t->next;
        t->next=newnode;
    }
}

void delete_front(){
    if(head==NULL) printf("Empty\n");
    else{
        struct node*t=head;
        head=head->next;
        free(t);
    }
}

void display(){
    struct node*t=head;
    while(t){printf("%d ",t->data);t=t->next;}
    printf("\n");
}

int main(){
    insert_end(10);insert_end(20);insert_end(30);
    display();
    delete_front();display();
    return 0;
}
```

Line-by-line explanation:

1. insert_end: create new node with next=NULL; if list empty, new becomes head; else traverse to las
2. delete_front: remove head node and move head to next.
3. display: print list nodes.

Program 11: Stack using linked list

Code:

```
#include <stdio.h>
#include <stdlib.h>

struct node{int data;struct node*next;};
struct node*top=NULL;

void push(int val){
    struct node*newnode=malloc(sizeof(struct node));
    newnode->data=val;
    newnode->next=top;
    top=newnode;
}

void pop(){
    if(top==NULL) printf("Empty\n");
    else{
        struct node*t=top;
        printf("Popped %d\n",t->data);
        top=top->next;
        free(t);
    }
}

void display(){
    struct node*t=top;
    while(t){printf("%d ",t->data);t=t->next;}
    printf("\n");
}

int main(){
    push(10);push(20);push(30);
    display();
    pop();display();
    return 0;
}
```

Line-by-line explanation:
1. push: create new node and add at front (top) of the list.
2. pop: remove node at top and free memory.
3. display: print from top to bottom.

Program 12: Binary Search Tree traversals

Code:

```
#include <stdio.h>
#include <stdlib.h>

struct node{
    int data;
    struct node*left,*right;
};

struct node*create(int val){
    struct node*n=malloc(sizeof(struct node));
    n->data=val;n->left=n->right=NULL;
    return n;
}

struct node*insert(struct node*root,int val){
    if(root==NULL) return create(val);
    if(val<root->data) root->left=insert(root->left,val);
    else root->right=insert(root->right,val);
    return root;
}

void inorder(struct node*root){
    if(root){inorder(root->left);printf("%d ",root->data);inorder(root->right);}
}
void preorder(struct node*root){
    if(root){printf("%d ",root->data);preorder(root->left);preorder(root->right);}
}
void postorder(struct node*root){
    if(root){postorder(root->left);postorder(root->right);printf("%d ",root->data);}
}

int main(){
    struct node*root=NULL;
    root=insert(root,50);
    insert(root,30);insert(root,70);insert(root,20);insert(root,40);
    printf("Inorder: ");inorder(root);
    printf("\nPreorder: ");preorder(root);
    printf("\nPostorder: ");postorder(root);
    return 0;
}
```

Line-by-line explanation:

1. struct node: node structure with data and left/right pointers.
2. create: allocate a new node, set data and children NULL, return pointer.
3. insert: recursive insertion into BST (left if val < root->data else right).
4. inorder/preorder/postorder: recursive traversal functions (explain order in words).
5. main: build BST and print traversals.