# **DATA STRUCTURES LABORATORY - CODES**

Course Code: CIL36

Credits: 0:0:1

Contact Hours: 15P

Course Coordinator: Mrs. Akshatha G C

## **1. Array operations - Menu driven program in C**

**Problem:**

Design, develop and implement a menu driven program in C for array operations: create, display, insert, delete, exit. Support functions for each operation.

**Code:**

#include <stdio.h>  
#include <stdlib.h>  
  
void create\_array(int \*\*arr, int \*n) {  
 int i;  
 printf("Enter number of elements: ");  
 scanf("%d", n);  
 \*arr = (int\*)malloc((\*n) \* sizeof(int));  
 for(i=0;i<\*n;i++) {  
 printf("Element %d: ", i);  
 scanf("%d", &(\*arr)[i]);  
 }  
}  
  
void display\_array(int \*arr, int n) {  
 int i;  
 if(n==0) { printf("Array is empty.\n"); return; }  
 printf("Array elements:\n");  
 for(i=0;i<n;i++) printf("%d ", arr[i]);  
 printf("\n");  
}  
  
void insert\_element(int \*\*arr, int \*n) {  
 int pos, elem, i;  
 printf("Enter position to insert (0 to %d): ", \*n);  
 scanf("%d", &pos);  
 if(pos<0 || pos>\*n) { printf("Invalid position.\n"); return; }  
 printf("Enter element to insert: ");  
 scanf("%d", &elem);  
 \*arr = (int\*)realloc(\*arr, (\*n+1)\*sizeof(int));  
 for(i=\*n;i>pos;i--) (\*arr)[i] = (\*arr)[i-1];  
 (\*arr)[pos] = elem;  
 (\*n)++;  
 printf("Inserted.\n");  
}  
  
void delete\_element(int \*\*arr, int \*n) {  
 int pos, i;  
 if(\*n==0) { printf("Array is empty.\n"); return; }  
 printf("Enter position to delete (0 to %d): ", \*n - 1);  
 scanf("%d", &pos);  
 if(pos<0 || pos>=\*n) { printf("Invalid position.\n"); return; }  
 for(i=pos;i<\*n-1;i++) (\*arr)[i] = (\*arr)[i+1];  
 (\*n)--;  
 \*arr = (int\*)realloc(\*arr, (\*n)\*sizeof(int));  
 printf("Deleted.\n");  
}  
  
int main() {  
 int \*arr = NULL, n = 0;  
 int choice;  
 while(1) {  
 printf("\\n--- Array Menu ---\\n");  
 printf("1. Create array\\n2. Display array\\n3. Insert element\\n4. Delete element\\n5. Exit\\n");  
 printf("Enter choice: ");  
 if(scanf("%d", &choice)!=1) break;  
 switch(choice) {  
 case 1: create\_array(&arr, &n); break;  
 case 2: display\_array(arr, n); break;  
 case 3: insert\_element(&arr, &n); break;  
 case 4: delete\_element(&arr, &n); break;  
 case 5: free(arr); exit(0);  
 default: printf("Invalid choice.\\n");  
 }  
 }  
 return 0;  
}

## **2. Employee structure with dynamic allocation**

**Problem:**

Define EMPLOYEE structure with Emp\_name, Emp\_id, Dept\_name, Salary. Read and display N employees. Write function to find total salary of a specified department. Use pointer to structure and dynamic allocation.

**Code:**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
typedef struct {  
 char name[50];  
 int emp\_id;  
 char dept[30];  
 double salary;  
} Employee;  
  
double total\_salary\_by\_dept(Employee \*emps, int n, const char \*dept) {  
 double total = 0.0;  
 for(int i=0;i<n;i++) {  
 if(strcmp(emps[i].dept, dept)==0) total += emps[i].salary;  
 }  
 return total;  
}  
  
int main() {  
 int n;  
 printf("Enter number of employees: ");  
 scanf("%d", &n);  
 Employee \*emps = (Employee\*)malloc(n \* sizeof(Employee));  
 for(int i=0;i<n;i++) {  
 printf("Employee %d name: ", i+1); scanf("%s", emps[i].name);  
 printf("Employee %d id: ", i+1); scanf("%d", &emps[i].emp\_id);  
 printf("Employee %d dept: ", i+1); scanf("%s", emps[i].dept);  
 printf("Employee %d salary: ", i+1); scanf("%lf", &emps[i].salary);  
 }  
 printf("\\nEmployees entered:\\n");  
 for(int i=0;i<n;i++) {  
 printf("%s\t%d\t%s\t%.2lf\\n", emps[i].name, emps[i].emp\_id, emps[i].dept, emps[i].salary);  
 }  
 char query[30];  
 printf("\\nEnter department to compute total salary: ");  
 scanf("%s", query);  
 double total = total\_salary\_by\_dept(emps, n, query);  
 printf("Total salary for department %s = %.2lf\\n", query, total);  
 free(emps);  
 return 0;  
}

## **3. Stack of Integers (Array implementation)**

**Problem:**

Implement stack with push, pop, check palindrome using stack, demonstrate overflow/underflow, display status, exit.

**Code:**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#define MAX 100  
  
int stack\_arr[MAX];  
int top = -1;  
  
int is\_empty() { return top == -1; }  
int is\_full() { return top == MAX-1; }  
  
void push(int x) {  
 if(is\_full()) { printf("Stack Overflow\\n"); return; }  
 stack\_arr[++top] = x;  
 printf("Pushed %d\\n", x);  
}  
  
int pop() {  
 if(is\_empty()) { printf("Stack Underflow\\n"); return -1; }  
 return stack\_arr[top--];  
}  
  
void display() {  
 if(is\_empty()) { printf("Stack empty\\n"); return; }  
 printf("Stack (top to bottom): ");  
 for(int i=top;i>=0;i--) printf("%d ", stack\_arr[i]);  
 printf("\\n");  
}  
  
int is\_palindrome\_using\_stack(int \*arr, int n) {  
 // push into stack and compare by popping  
 int i;  
 int temp[MAX];  
 int t = -1;  
 for(i=0;i<n;i++) temp[++t] = arr[i];  
 for(i=0;i<n;i++) {  
 if(arr[i] != temp[t--]) return 0;  
 }  
 return 1;  
}  
  
int main() {  
 int choice, val;  
 int arr\_for\_pal[20], npal=0;  
 while(1) {  
 printf("\\n--- Stack Menu ---\\n1 Push\\n2 Pop\\n3 Display\\n4 Check Palindrome (enter sequence)\\n5 Exit\\nChoice: ");  
 scanf("%d", &choice);  
 switch(choice) {  
 case 1: printf("Value to push: "); scanf("%d", &val); push(val); break;  
 case 2: val = pop(); if(val!=-1) printf("Popped %d\\n", val); break;  
 case 3: display(); break;  
 case 4:  
 printf("Enter number of elements (<=20): "); scanf("%d", &npal);  
 for(int i=0;i<npal;i++) { printf("Element %d: ", i); scanf("%d", &arr\_for\_pal[i]); }  
 if(is\_palindrome\_using\_stack(arr\_for\_pal, npal)) printf("Palindrome\\n"); else printf("Not palindrome\\n");  
 break;  
 case 5: exit(0);  
 default: printf("Invalid choice\\n");  
 }  
 }  
 return 0;  
}

## **4. Infix to Postfix (using stack)**

**Problem:**

Convert a valid parenthesized infix expression (single character operands, + - \* /) to postfix using stack.

**Code:**

#include <stdio.h>  
#include <string.h>  
#include <ctype.h>  
  
#define MAX 100  
  
char stack[MAX];  
int top4 = -1;  
  
void push4(char c) { if(top4 < MAX-1) stack[++top4] = c; }  
char pop4() { if(top4>=0) return stack[top4--]; return '\0'; }  
char peek4() { if(top4>=0) return stack[top4]; return '\0'; }  
  
int prec(char c) {  
 if(c=='+'||c=='-') return 1;  
 if(c=='\*'||c=='/') return 2;  
 return 0;  
}  
  
void infix\_to\_postfix(const char \*infix, char \*postfix) {  
 int j=0;  
 for(int i=0;i<strlen(infix);i++) {  
 char c = infix[i];  
 if(isalnum(c)) postfix[j++] = c;  
 else if(c=='(') push4(c);  
 else if(c==')') {  
 while(top4!=-1 && peek4()!='(') postfix[j++] = pop4();  
 pop4(); // remove '('  
 } else { // operator  
 while(top4!=-1 && prec(peek4()) >= prec(c)) postfix[j++] = pop4();  
 push4(c);  
 }  
 }  
 while(top4!=-1) postfix[j++] = pop4();  
 postfix[j] = '\0';  
}  
  
int main() {  
 char infix[100], postfix[100];  
 printf("Enter infix expression: ");  
 scanf("%s", infix);  
 infix\_to\_postfix(infix, postfix);  
 printf("Postfix: %s\\n", postfix);  
 return 0;  
}

## **5. Evaluate Postfix expression (using stack)**

**Problem:**

Evaluate a valid postfix expression with non-negative single digit operands and operators + - \* /.

**Code:**

#include <stdio.h>  
#include <ctype.h>  
#include <stdlib.h>  
  
#define MAX 100  
  
int stack5[MAX], top5=-1;  
void push5(int v){ if(top5<MAX-1) stack5[++top5]=v; }  
int pop5(){ if(top5>=0) return stack5[top5--]; return 0; }  
  
int evaluate\_postfix(const char \*exp) {  
 for(int i=0; exp[i]; i++) {  
 char c = exp[i];  
 if(isdigit(c)) push5(c - '0');  
 else {  
 int b = pop5(), a = pop5(), res=0;  
 switch(c) {  
 case '+': res = a + b; break;  
 case '-': res = a - b; break;  
 case '\*': res = a \* b; break;  
 case '/': res = b!=0 ? a / b : 0; break;  
 }  
 push5(res);  
 }  
 }  
 return pop5();  
}  
  
int main() {  
 char exp[100];  
 printf("Enter postfix expression (no spaces): ");  
 scanf("%s", exp);  
 int result = evaluate\_postfix(exp);  
 printf("Result = %d\\n", result);  
 return 0;  
}

## **6. Recursive functions: Binary Search and Tower of Hanoi**

**Problem:**

Write recursive functions for Binary Search and Tower of Hanoi and demonstrate their use.

**Code:**

#include <stdio.h>  
  
int binary\_search\_recursive(int arr[], int l, int r, int x) {  
 if(r < l) return -1;  
 int mid = l + (r-l)/2;  
 if(arr[mid] == x) return mid;  
 if(arr[mid] > x) return binary\_search\_recursive(arr, l, mid-1, x);  
 return binary\_search\_recursive(arr, mid+1, r, x);  
}  
  
void tower\_of\_hanoi(int n, char from, char to, char aux) {  
 if(n==1) { printf("Move disk 1 from %c to %c\\n", from, to); return; }  
 tower\_of\_hanoi(n-1, from, aux, to);  
 printf("Move disk %d from %c to %c\\n", n, from, to);  
 tower\_of\_hanoi(n-1, aux, to, from);  
}  
  
int main() {  
 int arr[] = {2,4,6,8,10,12};  
 int n = sizeof(arr)/sizeof(arr[0]);  
 int x = 8;  
 int idx = binary\_search\_recursive(arr, 0, n-1, x);  
 if(idx!=-1) printf("Found %d at index %d\\n", x, idx); else printf("Not found\\n");  
   
 printf("\\nTower of Hanoi for 3 disks:\\n");  
 tower\_of\_hanoi(3, 'A', 'C', 'B');  
 return 0;  
}

## **7. Call center phone system simulation (Queue)**

**Problem:**

Simulate call center holding calls and servicing them based on arrival time. Options to add and remove calls.

**Code:**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
typedef struct {  
 int call\_id;  
 char caller[50];  
} Call;  
  
typedef struct Node {  
 Call data;  
 struct Node \*next;  
} Node;  
  
Node \*front = NULL, \*rear = NULL;  
  
void enqueue(Call c) {  
 Node \*temp = (Node\*)malloc(sizeof(Node));  
 temp->data = c; temp->next = NULL;  
 if(rear==NULL) front = rear = temp;  
 else { rear->next = temp; rear = temp; }  
 printf("Call %d enqueued\\n", c.call\_id);  
}  
  
Call dequeue() {  
 Call empty = {-1, ""};  
 if(front==NULL) { printf("Queue empty\\n"); return empty; }  
 Node \*temp = front;  
 Call c = temp->data;  
 front = front->next;  
 if(front==NULL) rear = NULL;  
 free(temp);  
 printf("Call %d dequeued\\n", c.call\_id);  
 return c;  
}  
  
void display\_queue() {  
 Node \*p = front;  
 if(!p) { printf("No calls\\n"); return; }  
 printf("Queue: ");  
 while(p) { printf("[%d:%s] ", p->data.call\_id, p->data.caller); p = p->next; }  
 printf("\\n");  
}  
  
int main() {  
 int choice;  
 int next\_id = 1;  
 while(1) {  
 printf("\\n1 Add Call\\n2 Serve Call\\n3 Display Queue\\n4 Exit\\nChoice: ");  
 scanf("%d", &choice);  
 if(choice==1) {  
 Call c; c.call\_id = next\_id++;  
 printf("Caller name: "); scanf("%s", c.caller);  
 enqueue(c);  
 } else if(choice==2) {  
 dequeue();  
 } else if(choice==3) {  
 display\_queue();  
 } else break;  
 }  
 return 0;  
}

## **8. Circular Queue of integers (array of structures)**

**Problem:**

Simulate circular queue using array of structures. Start and end identified by empty array element.

**Code:**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
#define MAXQ 5  
  
typedef struct {  
 int val;  
 int used; // 0 empty, 1 occupied  
} Elem;  
  
Elem q[MAXQ];  
  
int is\_full() {  
 for(int i=0;i<MAXQ;i++) if(!q[i].used) return 0;  
 return 1;  
}  
  
int is\_empty() {  
 for(int i=0;i<MAXQ;i++) if(q[i].used) return 0;  
 return 1;  
}  
  
void enqueue(int v) {  
 if(is\_full()) { printf("Queue full\\n"); return; }  
 for(int i=0;i<MAXQ;i++) if(!q[i].used) { q[i].val=v; q[i].used=1; printf("Enqueued %d at %d\\n", v, i); break; }  
}  
  
void dequeue() {  
 if(is\_empty()) { printf("Queue empty\\n"); return; }  
 // find front (earliest occupied index)  
 int idx = -1;  
 for(int i=0;i<MAXQ;i++) if(q[i].used) { idx = i; break; }  
 if(idx!=-1) { printf("Dequeued %d from %d\\n", q[idx].val, idx); q[idx].used=0; }  
}  
  
void display() {  
 printf("Queue state: ");  
 for(int i=0;i<MAXQ;i++) {  
 if(q[i].used) printf("[%d] ", q[i].val); else printf("[ ] ");  
 }  
 printf("\\n");  
}  
  
int main() {  
 for(int i=0;i<MAXQ;i++) q[i].used=0;  
 int choice, v;  
 while(1) {  
 printf("\\n1 Enqueue\\n2 Dequeue\\n3 Display\\n4 Exit\\nChoice: ");  
 scanf("%d", &choice);  
 if(choice==1) { printf("Value: "); scanf("%d", &v); enqueue(v); }  
 else if(choice==2) dequeue();  
 else if(choice==3) display();  
 else break;  
 }  
 return 0;  
}

## **9. Singly linked list maintaining names in alphabetical order**

**Problem:**

Create singly linked list of names in alphabetical order. Insert and delete specified name.

**Code:**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
typedef struct Node {  
 char name[50];  
 struct Node \*next;  
} Node;  
  
Node\* insert\_sorted(Node \*head, const char \*name) {  
 Node \*newn = (Node\*)malloc(sizeof(Node));  
 strcpy(newn->name, name);  
 newn->next = NULL;  
 if(!head || strcmp(name, head->name) < 0) {  
 newn->next = head; return newn;  
 }  
 Node \*p = head;  
 while(p->next && strcmp(p->next->name, name) < 0) p = p->next;  
 newn->next = p->next; p->next = newn;  
 return head;  
}  
  
Node\* delete\_name(Node \*head, const char \*name) {  
 if(!head) return head;  
 if(strcmp(head->name, name)==0) { Node \*t=head->next; free(head); return t; }  
 Node \*p = head;  
 while(p->next && strcmp(p->next->name, name)!=0) p = p->next;  
 if(p->next) { Node \*t = p->next; p->next = t->next; free(t); printf("Deleted %s\\n", name); }  
 else printf("Name not found\\n");  
 return head;  
}  
  
void display(Node \*head) {  
 Node \*p = head; if(!p) { printf("Empty\\n"); return; }  
 while(p) { printf("%s -> ", p->name); p = p->next; } printf("NULL\\n");  
}  
  
int main() {  
 Node \*head = NULL;  
 int choice; char nm[50];  
 while(1) {  
 printf("\\n1 Insert\\n2 Delete\\n3 Display\\n4 Exit\\nChoice: ");  
 scanf("%d", &choice);  
 if(choice==1) { printf("Name: "); scanf("%s", nm); head = insert\_sorted(head, nm); }  
 else if(choice==2) { printf("Name to delete: "); scanf("%s", nm); head = delete\_name(head, nm); }  
 else if(choice==3) display(head);  
 else break;  
 }  
 return 0;  
}

## **10. Stack using linked implementation**

**Problem:**

Maintain stack of integers using linked list implementation.

**Code:**

#include <stdio.h>  
#include <stdlib.h>  
  
typedef struct Node {  
 int data;  
 struct Node \*next;  
} Node;  
  
Node \*top10 = NULL;  
  
void push10(int v) {  
 Node \*n = (Node\*)malloc(sizeof(Node));  
 n->data = v; n->next = top10; top10 = n;  
 printf("Pushed %d\\n", v);  
}  
  
int pop10() {  
 if(!top10) { printf("Underflow\\n"); return -1; }  
 Node \*t = top10; int v = t->data; top10 = t->next; free(t); return v;  
}  
  
void display10() {  
 Node \*p = top10;  
 if(!p) { printf("Empty\\n"); return; }  
 printf("Stack: ");  
 while(p) { printf("%d ", p->data); p = p->next; }  
 printf("\\n");  
}  
  
int main() {  
 int choice, v;  
 while(1) {  
 printf("\\n1 Push\\n2 Pop\\n3 Display\\n4 Exit\\nChoice: ");  
 scanf("%d", &choice);  
 if(choice==1) { printf("Value: "); scanf("%d", &v); push10(v); }  
 else if(choice==2) { v = pop10(); if(v!=-1) printf("Popped %d\\n", v); }  
 else if(choice==3) display10();  
 else break;  
 }  
 return 0;  
}

## **11. Doubly linked list operations**

**Problem:**

Support: a) Insert a new node to the left of the node whose key value is read as input. b) Delete a node with given data, else show error.

**Code:**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
typedef struct DNode {  
 int data;  
 struct DNode \*prev, \*next;  
} DNode;  
  
DNode\* create\_node(int v) {  
 DNode \*n = (DNode\*)malloc(sizeof(DNode));  
 n->data=v; n->prev=n->next=NULL; return n;  
}  
  
void insert\_left\_of(DNode \*\*head, int key, int val) {  
 DNode \*p = \*head;  
 while(p && p->data != key) p = p->next;  
 if(!p) { printf("Key %d not found\\n", key); return; }  
 DNode \*n = create\_node(val);  
 n->next = p; n->prev = p->prev;  
 if(p->prev) p->prev->next = n; else \*head = n;  
 p->prev = n;  
 printf("Inserted %d left of %d\\n", val, key);  
}  
  
void delete\_node(DNode \*\*head, int val) {  
 DNode \*p = \*head;  
 while(p && p->data != val) p = p->next;  
 if(!p) { printf("Value %d not found\\n", val); return; }  
 if(p->prev) p->prev->next = p->next; else \*head = p->next;  
 if(p->next) p->next->prev = p->prev;  
 free(p); printf("Deleted %d\\n", val);  
}  
  
void display\_dlist(DNode \*head) {  
 DNode \*p = head; if(!p) { printf("Empty\\n"); return; }  
 while(p) { printf("%d <-> ", p->data); p = p->next; } printf("NULL\\n");  
}  
  
int main() {  
 DNode \*head=NULL;  
 int choice, key, val;  
 while(1) {  
 printf("\\n1 Insert left of key\\n2 Delete value\\n3 Display\\n4 Exit\\nChoice: ");  
 scanf("%d", &choice);  
 if(choice==1) { printf("Key: "); scanf("%d", &key); printf("Value to insert: "); scanf("%d", &val); insert\_left\_of(&head, key, val); }  
 else if(choice==2) { printf("Value to delete: "); scanf("%d", &val); delete\_node(&head, val); }  
 else if(choice==3) display\_dlist(head);  
 else break;  
 }  
 return 0;  
}

## **12. Binary tree construction and traversals**

**Problem:**

Construct a binary tree of integers and traverse using inorder, preorder, postorder.

**Code:**

#include <stdio.h>  
#include <stdlib.h>  
  
typedef struct Node {  
 int data;  
 struct Node \*left, \*right;  
} Node;  
  
Node\* newnode(int v) {  
 Node \*n = (Node\*)malloc(sizeof(Node));  
 n->data = v; n->left = n->right = NULL; return n;  
}  
  
void inorder(Node \*root) { if(!root) return; inorder(root->left); printf("%d ", root->data); inorder(root->right); }  
void preorder(Node \*root) { if(!root) return; printf("%d ", root->data); preorder(root->left); preorder(root->right); }  
void postorder(Node \*root) { if(!root) return; postorder(root->left); postorder(root->right); printf("%d ", root->data); }  
  
int main() {  
 // Example tree creation  
 Node \*root = newnode(1);  
 root->left = newnode(2); root->right = newnode(3);  
 root->left->left = newnode(4); root->left->right = newnode(5);  
  
 printf("Inorder: "); inorder(root); printf("\\n");  
 printf("Preorder: "); preorder(root); printf("\\n");  
 printf("Postorder: "); postorder(root); printf("\\n");  
 return 0;  
}