SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

COURSE NAME : 23ITT101- PROBLEM SOLVING & C PROGRAMMING

I YEAR /I SEMESTER

Unit V – STRUCTURE AND UNION

Topic : Programs

Structure

```
// student details
#include <stdio.h>
#include <string.h>
// Defining a structure
struct Student
 {
    int id;
    char name[50];
    float marks;
};
int main() {
    // Declaring a structure variable
    struct Student student1;
    // Initializing the structure members
    student1.id = 101;
    strcpy(student1.name, "John Doe");
   student1.marks = 85.5;
```

```
 // Displaying the structure data
    printf("Student Details:\n");
    printf("ID: %d\n", student1.id);
    printf("Name: %s\n", student1.name);
   printf("Marks:%.2f\n",student1.marks);
    return 0;
}
```

Output:

Student Details:

ID: 101

Name: John Doe

Marks: 85.50

```
//Linked List
#include <stdio.h>
#include <stdlib.h>
// Definition for singly-linked list node.
struct ListNode {
    int val;
    struct ListNode* next;
};
// Function to insert a new node at the end
void insert(struct ListNode** head, int value) {
    struct ListNode* new_node = (struct ListNode*)malloc(sizeof(struct ListNode));
    new_node->val = value;
    new_node->next = NULL;
    if (*head == NULL) {
        *head = new_node;
        return;
    }
```

```
struct ListNode* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = new_node;
}
// Function to print the linked list
void printList(struct ListNode* head) {
    struct ListNode* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->val);
        temp = temp->next;
    }
    printf("NULL\n");
}
```

```
// Example of usage
int main() {
    struct ListNode* head = NULL;
    insert(&head, 1);
    insert(&head, 2);
    insert(&head, 3);
    printList(head);
    return 0;
}
```

```
//stack
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
// Definition for stack structure
struct Stack {
    int arr[MAX];
    int top;
};
// Initialize stack
void initStack(struct Stack* stack)
{    stack->top = -1;}
// Push operation
void push(struct Stack* stack, int value) {
    if (stack->top >= MAX - 1) {
        printf("Stack overflow\n");
        return;
    }
    stack->arr[++(stack->top)] = value;
}
```

```
// Pop operation
int pop(struct Stack* stack) {
    if (stack->top == -1) {
        printf("Stack underflow\n");
        return -1; // Error code
    }
    return stack->arr[(stack->top)--];
}
// Peek operation
int peek(struct Stack* stack) {
    if (stack->top == -1) {
        printf("Stack is empty\n");
        return -1; // Error code
    }
    return stack->arr[stack->top];
}
```

```
// Example of usage
int main() {
    struct Stack stack;
    initStack(&stack);

    push(&stack, 10);
    push(&stack, 20);
    push(&stack, 30);

    printf("Top of the stack: %d\n", peek(&stack));

    printf("Popped value: %d\n", pop(&stack));
    printf("Top of the stack after pop: %d\n", peek(&stack));

    return 0;
}
```

```c
//Binary tree
#include <stdio.h>
#include <stdlib.h>

// Definition for a binary tree node
struct TreeNode {
    int val;
    struct TreeNode* left;
    struct TreeNode* right;
};

// Function to create a new node
struct TreeNode* createNode(int value) {
    struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    newNode->val = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}
```

```c
// Inorder traversal (left, root, right)
void inorderTraversal(struct TreeNode* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->val);
        inorderTraversal(root->right);
    }
}
```

```c
// Example of usage
int main() {
    struct TreeNode* root = createNode(10);
    root->left = createNode(5);
    root->right = createNode(15);
    root->left->left = createNode(3);
    root->left->right = createNode(7);

    printf("Inorder traversal: ");
    inorderTraversal(root);  // Output: 3 5 7 10 15
    printf("\n");

    return 0;
}
```

```
// Queue
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
// Queue structure using two stacks
struct Queue {
    int stack1[MAX];
    int stack2[MAX];
    int top1;
    int top2;
};
// Initialize queue
void initQueue(struct Queue* queue) {
    queue->top1 = -1;
    queue->top2 = -1;
}
// Enqueue operation
void enqueue(struct Queue* queue, int value) {
    if (queue->top1 >= MAX - 1) {
        printf("Queue overflow\n");
return;    }
 queue->stack1[++(queue->top1)]          =
value;
}
// Dequeue operation
int dequeue(struct Queue* queue) {
    if (queue->top2 == -1) {
        if (queue->top1 == -1) {
            printf("Queue underflow\n");
            return -1; // Error code
  }
 while (queue->top1 != -1) {
            queue->stack2[++(queue->top2)]
= queue->stack1[(queue->top1)--];
        }
    }
 return queue->stack2[(queue->top2)--];
}
// Example of usage
int main() {
    struct Queue queue;
    initQueue(&queue);

    enqueue(&queue, 10);
    enqueue(&queue, 20);
    enqueue(&queue, 30);

    printf("Dequeued      value:      %d\n",
dequeue(&queue));  // Output: 10
    printf("Dequeued      value:      %d\n",
dequeue(&queue));  // Output: 20

    return 0;
}
```

```
// Hash map
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10
// Define a structure for key-value pairs
struct KeyValue {
    int key;
    int value;
    struct KeyValue* next;
};
// Define the hash map structure
struct HashMap {
    struct KeyValue* table[SIZE];
};
// Hash function to get the index for a key
int hash(int key) {
    return key % SIZE;
}
```

```
// Function to insert a key-value pair into the hash map
void put(struct HashMap* map, int key, int value) {
    int index = hash(key);
    struct KeyValue* new_pair = (struct KeyValue*)malloc(sizeof(struct KeyValue));
    new_pair->key = key;
    new_pair->value = value;
    new_pair->next = map->table[index];
    map->table[index] = new_pair;
}
// Function to retrieve a value by key
int get(struct HashMap* map, int key) {
    int index = hash(key);
    struct KeyValue* pair = map->table[index];
    while (pair != NULL) {
        if (pair->key == key)
{           return pair->value;      }
        pair = pair->next;     }
```

```
return -1;  // Return -1 if the key doesn't exist
}
// Example of usage
int main() {
    struct HashMap map = {0};
// Initialize hash map with nulls
    put(&map, 1, 100);
    put(&map, 2, 200);
    put(&map, 11, 300);
// This will collide with key 1 due to the hash function
    printf("Value for key 1: %d\n", get(&map, 1));
// Output: 100
    printf("Value for key 2: %d\n", get(&map, 2));
// Output: 200
    printf("Value for key 11: %d\n", get(&map, 11));  // Output: 300
        return 0;   }
```

Union

```
//Union example
#include <stdio.h>
#include <string.h>
// Defining a union
union Data {
    int i;
    float f;
    char str[20];
};
int main() {
    // Declaring a union variable
    union Data data;
// Assigning and displaying integer value
    data.i = 10;
    printf("Integer: %d\n", data.i);
    // Assigning and displaying float value
    data.f = 22.5;
    printf("Float: %.2f\n", data.f);

    // Assigning and displaying string value
    strcpy(data.str, "Hello, Union");
    printf("String: %s\n", data.str);

    // Observing memory sharing
    printf("\nMemory Sharing:\n");
    printf("Integer: %d (corrupted)\n", data.i);
// Data is overwritten
    printf("Float: %.2f (corrupted)\n", data.f);
// Data is overwritten
    printf("String: %s\n", data.str);

    return 0;
}
```

Output:

Integer: 10

Float: 22.50

String: Hello, Union

Memory Sharing:

Integer: 1162627392 (corrupted)

Float: 151071683796574752 4608.00 (corrupted)

String: Hello, Union

Union

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

// Defining a union
union Value {
    int intVal;
    float floatVal;
    char charVal;
};

int main() {
    // Declaring a union variable
    union Value val;

    // Assigning an integer
    val.intVal = 42;
    printf("Integer Value: %d\n", val.intVal);

    // Assigning a float
    val.floatVal = 3.14;
    printf("Float Value: %.2f\n", val.floatVal);

    // Assigning a character
    val.charVal = 'A';
    printf("Character Value: %c\n", val.charVal);

    // Observing memory overlap
    printf("\nMemory Sharing Observation:\n");
    printf("Integer Value (corrupted): %d\n", val.intVal);
    printf("Float Value (corrupted): %.2f\n", val.floatVal);

    return 0;
}
```

Output:

Integer Value: 42

Float Value: 3.14

Character Value: A

Memory Sharing Observation:

Integer Value (corrupted): 1094795585

Float Value (corrupted): 4.600602

Union

```
#include <stdio.h>

// Defining a union
union Data {
    int i;
    float f;
    char bytes[4];
};

int main() {
    // Declaring and initializing the union
    union Data data;
    data.i = 0x41424344;
 // Hexadecimal representation for ASCII 'ABCD'
 // Interpreting the same memory as integer, float, and bytes
    printf("Interpreted as Integer: %d\n", data.i);
    printf("Interpreted as Float: %f\n", data.f);
    printf("Interpreted as Bytes: %c %c %c %c\n", data.bytes[0], data.bytes[1], data.bytes[2], data.bytes[3]);
    return 0;
}
```

Output:

Interpreted as Integer: 1094861636

Interpreted as Float: 12.347846

Interpreted as Bytes: D C B A

Union

```
//Union for Tagged Data Type (Discriminated Union)
#include <stdio.h>
#include <string.h>

// Defining a union
union Value {
    int i;
    float f;
    char str[20];
};

// Defining a structure with a tag and union
struct Data {
    int type;  // 1 for int, 2 for float, 3 for string
    union Value value;
};
```

```
int main() {
    struct Data data;
    // Example 1: Storing an integer
    data.type = 1;
    data.value.i = 42;
    printf("Integer: %d\n", data.value.i);
    // Example 2: Storing a float
    data.type = 2;
    data.value.f = 3.1415;
    printf("Float: %.4f\n", data.value.f);
    // Example 3: Storing a string
    data.type = 3;
    strcpy(data.value.str, "Hello, Union!");
    printf("String: %s\n", data.value.str);
    return 0;
}
```

Output:

Integer: 42

Float: 3.1415

String: Hello, Union!


Thank
you