**Lab on Structures in C**

**Objective**

To understand and implement structures in C, including their initialization, memory alignment, size optimization, and compiler-specific features.

**1. Definitions**

Structures in C are user-defined data types that group different variables together.

**Example:**

struct Student {

int id;

char name[50];

float marks;

};

**2. Creating an Instance of a Structure**

You can create an instance of a structure in two ways:

struct Student s1; // Standard Declaration

struct Student s2 = {101, "Alice", 89.5}; // Initialization

**3. Initializing Structure Instances in C89 and C99**

**C89 Style:**

struct Student s1 = {101, "Alice", 89.5};

**C99 Style:**

s1 = (struct Student){102, "Bob", 90.0};

**4. Offsets of the Fields in a Structure Instance**

The offsetof macro from <stddef.h> gives the offset of fields.

#include <stdio.h>

#include <stddef.h>

struct Example {

char a;

int b;

double c;

};

int main() {

printf("Offset of a: %lu\n", offsetof(struct Example, a));

printf("Offset of b: %lu\n", offsetof(struct Example, b));

printf("Offset of c: %lu\n", offsetof(struct Example, c));

return 0;

}

**5. Alignment Restrictions for Data Types**

Memory alignment depends on the architecture and compiler.

#include <stdio.h>

#include <stdalign.h>

int main() {

printf("Alignment of int: %zu\n", alignof(int));

printf("Alignment of double: %zu\n", alignof(double));

return 0;

}

**6. Size of a Structure & Dependence on Alignment Restrictions**

The size of a structure depends on alignment and padding.

#include <stdio.h>

struct Example {

char a;

int b;

double c;

};

int main() {

printf("Size of struct: %lu\n", sizeof(struct Example));

return 0;

}

**7. Optimizing Structure Size by Reordering Fields**

Rearranging fields can reduce padding and optimize memory usage.

struct NonOptimized {

char a;

int b;

char c;

};

struct Optimized {

char a;

char c;

int b;

};

**8. Packed and Unpacked Structures**

A packed structure removes extra padding.

struct \_\_attribute\_\_((packed)) PackedStruct {

char a;

int b;

char c;

};

**9. Support for Packed Structures in GCC and MS Compilers**

**GCC (Gnu Compiler Collection)**

struct \_\_attribute\_\_((packed)) MyStruct {

char a;

int b;

char c;

};

**MSVC (Microsoft Visual C++)**

#pragma pack(1)

struct MyStruct {

char a;

int b;

char c;

};

#pragma pack()