Understanding Union vs Structure

Problem Statement: Write a program to define a union and structure to store employee information (name, employee ID, and salary). Demonstrate the difference in memory usage and behavior between a union and structure when storing the same set of data.

Assignment Tasks:

• Define a union and a struct for employee information.

• Initialize and display values stored in both the union and struct.

• Calculate and display the memory size occupied by each using size of().

CODE:

```
#include <stdio.h>
#include <string.h>
struct EmployeeStruct {
   char name[50];
   int employeeID;
   float salary;
};
union EmployeeUnion {
   char name[50];
   int employeeID;
   float salary;
};
int main() {
   // Initialize structure
   struct EmployeeStruct empStruct;
   strcpy(empStruct.name, "Saurabh kala");
   empStruct.employeeID = 102;
   empStruct.salary = 85000.50;
   // Initialize union
   union EmployeeUnion empUnion;
   strcpy(empUnion.name, "Saurabh kala");

   // Display structure information
   printf("Structure:\n");
   printf("Name: %s\n", empStruct.name);
   printf("Employee ID: %d\n", empStruct.employeeID);
   printf("Salary: %.2f\n", empStruct.salary);

   printf("\nUnion (after storing name):\n");
   printf("Name: %s\n", empUnion.name);

   empUnion.employeeID = 102;
   printf("Union (after storing employeeID):\n");
   printf("Employee ID: %d\n", empUnion.employeeID);
```

```
    empUnion.salary = 85000.50;
    printf("Union (after storing salary):\n");
    printf("Salary: %.2f\n", empUnion.salary);
    printf("\nMemory Usage:\n");
    printf("Size of structure: %zu bytes\n", sizeof(empStruct));
    printf("Size of union: %zu bytes\n", sizeof(empUnion));

    return 0;
}
```

OUTPUT

```
C:\Users\saura\union vs struc

Structure:
Name: Saurabh kala
Employee ID: 102
Salary: 85000.50

Union (after storing name):
Name: Saurabh kala
Union (after storing employeeID):
Employee ID: 102
Union (after storing salary):
Salary: 85000.50

Memory Usage:
Size of structure: 60 bytes
Size of union: 52 bytes

--------------------------------
Process exited after 0.1064 seconds with return value 0
Press any key to continue . . .
```

Dynamic Memory Allocation with malloc() and free()

Problem Statement: Write a program to dynamically allocate memory for an array of integers. Perform the following operations:
1. Input the number of elements (n).
2. Allocate memory dynamically using malloc().
3. Input n elements into the array.
4. Find the sum and average of the elements.
5. Release the memory using free().

Assignment Tasks:

- Use malloc() for dynamic memory allocation.
- Input values into the dynamically allocated array.
- Calculate sum and average.
- Use free() to release the allocated memory.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
   int n, i;
   int *arr;
   int sum = 0;
   float average;
   // Step 1: Input the number of elements
   printf("Enter the number of elements: ");
   scanf("%d", &n);
   // Step 2: Allocate memory dynamically using malloc()
   arr = (int*) malloc(n * sizeof(int));
   if (arr == NULL) {
      printf("Memory allocation failed.\n");
      return 1;
   }
   // Step 3: Input n elements into the array
   printf("Enter %d elements:\n", n);
   for (i = 0; i < n; i++) {
      printf("Element %d: ", i + 1);
      scanf("%d", &arr[i]);
      sum += arr[i];
   }
   // Step 4: Calculate sum and average
   average = (float)sum / n;
   printf("\nSum of elements: %d\n", sum);
   printf("Average of elements: %.2f\n", average);
   // Step 5: Release the memory using free()
   free(arr);
   return 0;
}
```

OUTPUT:

```
C:\Users\saura\malloc.exe

Enter the number of elements: 5
Enter 5 elements:
Element 1: 20
Element 2: 30
Element 3: 50
Element 4: 70
Element 5: 90

Sum of elements: 260
Average of elements: 52.00

--------------------------------
Process exited after 22.48 seconds with return value 0
Press any key to continue . . .
```