BEYOND THE SYLLABUS – STRUCTURES

A Mailing List Example: To illustrate how structures and arrays of structures are used, Write a C program to develop a simple mailing list program that uses an array of structures to hold the address information.

- Usually, the members of a structure are logically related.

✓ AIM

To write a C program that maintains a simple mailing list using **arrays of structures**. The program should allow the user to:

1. **Enter** a new address record
2. **Delete** an existing record
3. **Display (list)** all stored records
4. Exit the application

The program must make use of **structures**, **arrays**, **functions**, and **menu-driven interaction**.

✓ ALGORITHM

Step 1:

Start the program and define the structure addr containing name, street, city, state, and zip.

Step 2:

Declare an array addr_list[MAX] to store up to 100 records.

Step 3:

Call the function **init_list()** to initialize all structure entries by setting name[0] = '\0'.

Step 4:

Use a **menu-driven loop** to repeatedly display the options:

1. Enter a name
2. Delete a name
3. List the file
4. Quit

Step 5:

Based on the menu choice:

Case 1: Enter a Name

1. Call find_free() to locate the first empty slot.
2. If slot is available:
    - Input name, street, city, state, and zip.
    - Store them in the structure array.
3. If no slot is free, display "List Full".

Case 2: Delete a Name

1. Ask the user to enter the record number (index).
2. If valid, set name[0] = '\0' to mark the entry as deleted.

Case 3: List the File

1. Traverse the array from index 0 to MAX.
2. For every element whose name is not empty:
    - Display name, street, city, state, and zip.

Case 4: Quit

Exit the program.

Step 6:
Continue displaying the menu until the user chooses to quit.
Step 7:
Stop the program.

✅ C Program — Mailing List Example

```
/* Simple mailing list example - array of structures */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX 100

struct addr {
   char name[30];
   char street[40];
   char city[20];
   char state[3];
   unsigned long int zip;
} addr_list[MAX];

void init_list(void);
void enter(void);
void delete_rec(void);
void list(void);
int menu_select(void);
int find_free(void);

/* Remove trailing newline from fgets */
void strip_newline(char *s) {
   s[strcspn(s, "\n")] = '\0';
}

int main(void)
{
   int choice;
   init_list(); /* initialize the structure array */

   for(;;) {
      choice = menu_select();
      switch(choice) {
         case 1: enter(); break;
         case 2: delete_rec(); break;
         case 3: list(); break;
         case 4: exit(0);
      }
   }
```

```
  return 0;
}

/* Initialize the list. */
void init_list(void)
{
  for(int t = 0; t < MAX; ++t)
    addr_list[t].name[0] = '\0';
}

/* Get a menu selection. */
int menu_select(void)
{
  char s[80];
  int c;

  printf("1. Enter a name\n");
  printf("2. Delete a name\n");
  printf("3. List the file\n");
  printf("4. Quit\n");

  do {
    printf("\nEnter your choice: ");
    fgets(s, sizeof(s), stdin);
    c = atoi(s);
  } while(c < 1 || c > 4);

  return c;
}

/* Input addresses into the list. */
void enter(void)
{
  int slot;
  char s[80];

  slot = find_free();
  if(slot == -1) {
    printf("\nList Full\n");
    return;
  }

  printf("Enter name: ");
  fgets(addr_list[slot].name, sizeof(addr_list[slot].name), stdin);
  strip_newline(addr_list[slot].name);

  printf("Enter street: ");
  fgets(addr_list[slot].street, sizeof(addr_list[slot].street), stdin);
  strip_newline(addr_list[slot].street);
```

```
    printf("Enter city: ");
    fgets(addr_list[slot].city, sizeof(addr_list[slot].city), stdin);
    strip_newline(addr_list[slot].city);

    printf("Enter state (2 letters): ");
    fgets(addr_list[slot].state, sizeof(addr_list[slot].state), stdin);
    strip_newline(addr_list[slot].state);

    printf("Enter zip: ");
    fgets(s, sizeof(s), stdin);
    addr_list[slot].zip = strtoul(s, NULL, 10);
}

/* Find an unused structure. */
int find_free(void)
{
    for(int t = 0; t < MAX; ++t)
        if(addr_list[t].name[0] == '\0')
            return t;

    return -1; /* no slots free */
}

/* Delete an address. */
void delete_rec(void)
{
    char s[80];
    int slot;

    printf("Enter record #: ");
    fgets(s, sizeof(s), stdin);
    slot = atoi(s);

    if(slot >= 0 && slot < MAX)
        addr_list[slot].name[0] = '\0';
    else
        printf("Invalid record number\n");
}

/* Display the list on the screen. */
void list(void)
{
    for(int t = 0; t < MAX; ++t) {
        if(addr_list[t].name[0]) {
            printf("%s\n", addr_list[t].name);
            printf("%s\n", addr_list[t].street);
            printf("%s\n", addr_list[t].city);
            printf("%s\n", addr_list[t].state);
            printf("%lu\n\n", addr_list[t].zip);
        }
```

```
    }
  }
```

MENU-DRIVEN PROGRAM USING STRUCTURES + DYNAMIC MEMORY

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

/* Structure definition */
struct addr {
   char name[30];
   char street[40];
   char city[20];
   char state[3];
   unsigned long zip;
};

/* Global dynamic list */
struct addr *list = NULL;
int count = 0;     // number of records

/* Function prototypes */
void add_record();
void delete_record();
void list_records();
int menu();
void strip_newline(char *s);

int main() {
   int choice;

   for (;;) {
      choice = menu();
      switch(choice) {
         case 1: add_record(); break;
         case 2: delete_record(); break;
         case 3: list_records(); break;
         case 4:
            free(list);
            exit(0);
         default:
            printf("Invalid choice\n");
      }
   }
   return 0;
}

/* Remove trailing newline */
void strip_newline(char *s) {
```

```
    s[strcspn(s, "\n")] = '\0';
}

/* Menu */
int menu() {
    char s[10];
    int c;

    printf("\n----- MAILING LIST MENU -----\n");
    printf("1. Add a Record\n");
    printf("2. Delete a Record\n");
    printf("3. Display All Records\n");
    printf("4. Quit\n");
    printf("Enter choice: ");

    fgets(s, sizeof(s), stdin);
    c = atoi(s);
    return c;
}

/* Add a new record using dynamic memory (realloc) */
void add_record() {
    char temp[100];

    /* Increase memory for one more record */
    list = realloc(list, (count + 1) * sizeof(struct addr));

    if (!list) {
        printf("Memory allocation failed!\n");
        exit(1);
    }

    /* Input details */
    printf("Enter name: ");
    fgets(list[count].name, sizeof(list[count].name), stdin);
    strip_newline(list[count].name);

    printf("Enter street: ");
    fgets(list[count].street, sizeof(list[count].street), stdin);
    strip_newline(list[count].street);

    printf("Enter city: ");
    fgets(list[count].city, sizeof(list[count].city), stdin);
    strip_newline(list[count].city);

    printf("Enter state: ");
    fgets(list[count].state, sizeof(list[count].state), stdin);
    strip_newline(list[count].state);

    printf("Enter zip: ");
```

```
    fgets(temp, sizeof(temp), stdin);
    list[count].zip = strtoul(temp, NULL, 10);

    count++;
    printf("\nRecord added successfully!\n");
}

/* Delete a record */
void delete_record() {
    char s[10];
    int index;

    if (count == 0) {
        printf("No records to delete.\n");
        return;
    }

    printf("Enter record number to delete (0 - %d): ", count - 1);
    fgets(s, sizeof(s), stdin);
    index = atoi(s);

    if (index < 0 || index >= count) {
        printf("Invalid record number!\n");
        return;
    }

    /* Shift remaining records up */
    for (int i = index; i < count - 1; i++) {
        list[i] = list[i + 1];
    }

    /* Reduce memory */
    count--;
    list = realloc(list, count * sizeof(struct addr));

    printf("Record deleted successfully!\n");
}

/* Display all records */
void list_records() {
    if (count == 0) {
        printf("No records available.\n");
        return;
    }

    printf("\n----- MAILING LIST -----\n");

    for (int i = 0; i < count; i++) {
        printf("Record %d:\n", i);
        printf("Name   : %s\n", list[i].name);
```

```
        printf("Street : %s\n", list[i].street);
        printf("City   : %s\n", list[i].city);
        printf("State  : %s\n", list[i].state);
        printf("Zip    : %lu\n", list[i].zip);
        printf("----------------------------\n");
    }
}
```

☑ What This Improved Version Includes

✓ **Uses malloc + realloc for dynamic storage**
No wasted memory—grows and shrinks automatically.

✓ **Eliminates fixed array limit**
Stores unlimited records until memory is full.

✓ **Safe input fgets()**
No security risk like gets().

✓ **Automatic shifting after delete**
Maintains continuous record numbering.

✓ **Clean and modern C code**

✅ RESULT

The program was successfully executed. It enables the user to:

- Add new mailing list records
- Delete existing records
- Display all stored address entries

using **structures**, **arrays**, and a **menu-driven interface**.