Structure and Dynamic Memory Management assignments

1. Refer the question 1 solved in “Structure and function”. Extend the above program to read a number of records from the user as a single command line argument (each record is delimited by a semicolon and record fields are delimited by comma) and store in an array of structures.
Sample input and output are given below.
Input: “user1,90;user21,100, userABC,56,userX,40”;
Output:
No. of records: 4
Record 1:
Name:user1, Percentage:90
Record 2:
Name:user21, Percentage:100
Record 3:
Name:userABC, Percentage:56
Record 4:
Name:userX, Percentage:40
Implement all required functions and call them to get the desired output.
Check for memory leak

Sol:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


struct student {

   char name[50];

   int percentage;

};

int read_records(char *input, struct student **students) {

   int count = 0;

   char *record, *field;

   *students = (struct student *)malloc(10 * sizeof(struct student));

   if (*students == NULL) {

      printf("Memory allocation failed!\n");
```

```
        return -1;
    }
     record = strtok(input, ";");
    while (record != NULL) {
        field = strtok(record, ",");
        int field_count = 0;

         while (field != NULL) {
            if (field_count == 0) {
                strcpy((*students)[count].name, field);
            } else if (field_count == 1) {
                (*students)[count].percentage = atoi(field);
            }
            field = strtok(NULL, ",");
            field_count++;
        }
        count++;
        record = strtok(NULL, ";");
    }

    return count;
}

void print_records(struct student *students, int count) {
    printf("No. of records: %d\n", count);
    for (int i = 0; i < count; i++) {
        printf("Record %d: Name: %s, Percentage: %d\n", i + 1, students[i].name,
students[i].percentage);
    }
}
void free_records(struct student *students) {
```

```
    free(students);
}
int main(int argc, char *argv[]) {
    if (argc < 2) {
        printf("Please provide input records as a command-line argument.\n");
        return 1;
    }
 struct student *students = NULL;
    int count = read_records(argv[1], &students);

    if (count == -1) {
        return 1;
    }

    print_records(students, count);
    free_records(students);
    return 0;
}
```

Output:

```
user57@trainux01:~/Batch17OCT2024/struc$ ./a.out user57,90
No. of records: 1
Record 1: Name: user57, Percentage: 90
```

2. 2a. Extend Q1. Above and add 3 functions below.
//to search for a name and to replace it with a user defined name, return replaced string
char*search_update(char *searchstr, char *replacestr);
//search and delete the record with given name or percentage value, return SUCCESS on successful delete else FAILURE
int delete_record(char *searchstr, int percent);
//search for name and if found create a copy of the record in newstudent and return SUCCESS, else FAILURE
int copy(char *name, struct student **newstudent);
OR

2b. Refer the code in “structure_dynamic” and implement the functions below.
name_ret free_person()
name_ret update_person()

sol:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


struct student {

   char name[50];

   int percentage;

};

#define SUCCESS 1

#define FAILURE 0


int read_records(char *input, struct student **students) {

   int count = 0;

   char *record, *field;

   *students = (struct student *)malloc(10 * sizeof(struct student)); // Assuming at least 10 records for now

   if (*students == NULL) {

      printf("Memory allocation failed!\n");

      return -1;

   }


   record = strtok(input, ";");

   while (record != NULL) {

      field = strtok(record, ",");

      int field_count = 0;


      // Process the fields (name and percentage)

      while (field != NULL) {
```

```
            if (field_count == 0) {
                strcpy((*students)[count].name, field);
            } else if (field_count == 1) {
                (*students)[count].percentage = atoi(field);
            }
            field = strtok(NULL, ",");
            field_count++;
        }

        count++;
        record = strtok(NULL, ";");
    }

    return count;
}

char* search_update(char *searchstr, char *replacestr, struct student *students, int count) {
    for (int i = 0; i < count; i++) {
        if (strcmp(students[i].name, searchstr) == 0) {
            strcpy(students[i].name, replacestr); // Replace the name
            return students[i].name;
        }
    }
    return NULL; // Not found
}

int delete_record(char *searchstr, int percent, struct student *students, int *count) {
    for (int i = 0; i < *count; i++) {
        if (strcmp(students[i].name, searchstr) == 0 || students[i].percentage == percent) {
            // Shift remaining records to delete the current one
            for (int j = i; j < *count - 1; j++) {
```

```
            students[j] = students[j + 1];
        }
        (*count)--;
        return SUCCESS;
      }
    }
    return FAILURE;
}

int copy(char *name, struct student *students, int count, struct student **newstudent) {
    for (int i = 0; i < count; i++) {
      if (strcmp(students[i].name, name) == 0) {
        // Allocate memory for the new student record
        *newstudent = (struct student *)malloc(sizeof(struct student));
        if (*newstudent == NULL) {
            return FAILURE;
        }
        strcpy((*newstudent)->name, students[i].name);
        (*newstudent)->percentage = students[i].percentage;
        return SUCCESS;
      }
    }
    return FAILURE;
}

void print_records(struct student *students, int count) {
    printf("No. of records: %d\n", count);
    for (int i = 0; i < count; i++) {
      printf("Record %d: Name: %s, Percentage: %d\n", i + 1, students[i].name,
students[i].percentage);
    }
```

```
}

void free_records(struct student *students) {
  free(students);
}

int main(int argc, char *argv[]) {
  if (argc < 2) {
    printf("Please provide input records as a command-line argument.\n");
    return 1;
  }

  struct student *students = NULL;
  int count = read_records(argv[1], &students);

  if (count == -1) {
    return 1;
  }

  print_records(students, count);
  printf("\nUpdating 'userABC' to 'userNew'.\n");
  char *updated_name = search_update("userABC", "userNew", students, count);
  if (updated_name != NULL) {
    printf("Updated name: %s\n", updated_name);
  } else {
    printf("Name not found for update.\n");
  }

  printf("\nDeleting record with name 'user21'.\n");
  if (delete_record("user21", -1, students, &count) == SUCCESS) {
```

```
        printf("Record 'user21' deleted successfully.\n");
    } else {
        printf("Record 'user21' not found.\n");
    }
   struct student *new_student = NULL;
    printf("\nCopying record of 'userX'.\n");
    if (copy("userX", students, count, &new_student) == SUCCESS) {
        printf("Copied student: Name: %s, Percentage: %d\n", new_student->name, new_student-
>percentage);
        free(new_student); // Don't forget to free the copied student
    } else {
        printf("Record 'userX' not found for copying.\n");
    }
    print_records(students, count);
    free_records(students);

    return 0;
}
```

Output:

No. of records: 4

Record 1: Name: user1, Percentage: 90

Record 2: Name: user21, Percentage: 100

Record 3: Name: userABC, Percentage: 56

Record 4: Name: userX, Percentage: 40

Updating 'userABC' to 'userNew'.

Updated name: userNew

Deleting record with name 'user21'.

Record 'user21' deleted successfully.

Copying record of 'userX'.

Copied student: Name: userX, Percentage: 40

No. of records: 3

Record 1: Name: user1, Percentage: 90

Record 2: Name: userNew, Percentage: 56

Record 3: Name: userX, Percentage: 40