**W9 -** PRACTICE

*Dynamic Memory Allocation*

## *At the end of this practice, you should be able to…*

* Identify **memory leaks**
* **Allocate** and **free** **dynamic arrays** (1D & 2D)
* Allocate memory for **array of structures**
* **Resize array** as new data comes in

­

## *How do we structure exercises?*

We organize this practice into 4 parts:

|  |  |
| --- | --- |
| ANALYSE | **Understand** existing codes, find the **bugs** or **complete** missing gaps |
| MANIPULATE | Ensure you can **apply the theory** with some basic challenges |
| CREATE | **Express your creativity** with more complex challenges |

## *Are you lost?*

You can read the following documentation to be ready for this practice

Structs

<https://www.w3schools.com/c/c_structs.php>

**ANALYSE**

**EX 1 (Fix Buggy Code)**

Look at the bellow code.

int main() {

int\* arr = **malloc**(10 \* sizeof(int));

for (int i = 0; i < 10; i++) {

arr[i] = i \* 2;

}

printf("Done\n");

return 0;

}

**Q1 –** Explain the **memory leak**

no free(arr)

**Q2 –** Fix the code

int main() {

int\* arr = **malloc**(10 \* sizeof(int));

for (int i = 0; i < 10; i++) {

arr[i] = i \* 2;

}

printf("Done\n");

free(arr);

return 0;

}

**EX 2 (Fix Buggy Code)**

Look at the bellow code.

int main() {

char\* str = **malloc**(50);

strcpy(str, "Hello");

str = **malloc**(100);

strcpy(str, "World");

**free**(str);

return 0;

}

**Q1 –** Explain the **memory leak**

free str only frees the second allocation of 100 bytes

**Q2 –** Fix the code

int main() {

char\* str = **malloc**(50);

strcpy(str, "Hello");

free(str);

str = **malloc**(100);

strcpy(str, "World");

**free**(str);

return 0;

}

**EX 3 (Fix Buggy Code)**

Look at the bellow code.

typedef struct {

char\* name;

int age;

} Person;

int main() {

Person\* p = **malloc**(sizeof(Person));

p->name = **malloc**(100);

strcpy(p->name, "Alice");

**free**(p); // name is not freed!

return 0;

}

**Q1 –** Explain the **memory leak**

we must free inner members first before freeing the entire struct. If we free the struct then the members of said struct is “orphaned”

**Q2 –** Fix the code

typedef struct {

char\* name;

int age;

} Person;

int main() {

Person\* p = **malloc**(sizeof(Person));

p->name = **malloc**(100);

strcpy(p->name, "Alice");

free(p->name);

**free**(p); // name is not freed!

return 0;

}

**MANIPULATE**

**EX 1 (*The Survey Tool – Array allocation*)**

🎯 We want to build a **survey tool**.

The number of people who will take the survey is not known in advance.

* The program should ask how many participants will respond
* Then collect and store each person’s rating (between 1 and 10)
* At the end, it prints all the ratings entered

**✅ Example**

How many ratings will you enter? 4

Enter rating #1: 8

Enter rating #2: 6

Enter rating #3: 10

Enter rating #4: 7

Ratings entered: 8 6 10 7

**🔒 Constraints**

* You **must** use **malloc**() and **free**().
* Validate that the memory allocation was successful (i.e., check for NULL).
* Use a loop to read and display the elements.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main(){

int numRatings;

printf("How many ratings will you enter? ");

scanf("%d", &numRatings);

int\* ratings = malloc(numRatings \* sizeof(int));

if(ratings != NULL){

for(int i = 0; i<numRatings;i++){

printf("Enter rating #%d: ", i+1);

scanf("%d", ratings+i);

}

printf("Ratings entered: ");

for(int i = 0; i<numRatings;i++){

printf("%d ", \*ratings+i);

}

free(ratings);

} else {

printf("Failed to allocate memory");

}

return 0;

}

**EX 2 (*The Classroom Seating Chart – Array 2D allocation*)**

💡 **BEFORE START !!**

To allocate dynamically an array 2D:

* First allocate an **array of row pointers**
* Then **allocate each row** individually.

int rows = 3, cols = 4;

// Allocation of the row pointers

int\*\* array = **malloc**(rows \* sizeof(int\*));

// Allocation of each rows

for (int i = 0; i < rows; i++) {

array[i] = **malloc**(cols \* sizeof(int));

}

// Deallocation

for (int i = 0; i < rows; i++) {

**free**(array[i]);

}

**free**(array);

🎯 **Objective**

You're helping a school create a **digital classroom seating plan**

* Each student has a unique ID number.
* The class can have any number of rows and columns depending on the room.
* Your program must allow the school to enter the seating chart and display it cleanly.

**✅ Example**

Enter number of rows: 2

Enter number of columns: 3

Enter student ID at row 0, column 0: 101

Enter student ID at row 0, column 1: 102

Enter student ID at row 0, column 2: 103

Enter student ID at row 1, column 0: 201

Enter student ID at row 1, column 1: 202

Enter student ID at row 1, column 2: 203

Seating Chart:

101 102 103

201 202 203

**🔒 Constraints**

* You **must** use **malloc**() and **free**().
* Validate that the memory allocation was successful (i.e., check for NULL).

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main(){

int rows, cols;

printf("Enter number of rows: ");

scanf("%d", &rows);

printf("Enter number of columns: ");

scanf("%d", &cols);

// Allocation of the row pointers

int\*\* array = malloc(rows \* sizeof(int\*));

if(array == NULL) {

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

return 1;

}

// Allocation of each rows

for (int i = 0; i < rows; i++) {

array[i] = malloc(cols \* sizeof(int));

if(array[i] == NULL) {

printf("Memory allocation failed for row %d!\n", i);

for (int j = 0; j < i; j++) {

free(array[j]);

}

free(array);

return 1;

}

}

for(int i = 0; i<rows;i++){

for(int j = 0; j < cols; j++){

printf("Enter student id at row %d, column %d: ", i,j);

scanf("%d",&array[i][j]);

}

}

printf("Seating Chart\n");

for(int i = 0; i<rows;i++){

for(int j = 0; j < cols; j++){

printf("%d ", array[i][j]);

}

printf("\n");

}

// Deallocation

for (int i = 0; i < rows; i++) {

free(array[i]);

}

free(array);

}

**EX 3 (*The Employees – Array of structures*)**

💡 **BEFORE START !!**

To allocate dynamically an array of structures:

typedef struct {

char name[50];

int age;

} Student;

int n;

scanf("%d", &n);

// Allocation

struct Student\* students = **malloc**(n \* sizeof(Student));

// Deallocation

**free**(students);

🎯 **Objective**

1. Define a **struct Employee** with:

char name[50];

int salary;

1. Dynamically **allocate memory for 2 employees** using malloc.
2. Ask the **user to input** the name and salary for these 2 employees.
3. Use **realloc** to expand the array to **store 5 employees**.
4. Ask the **user to input** the remaining 3 employees.
5. Print the name and salary of all 5 employees.
6. **Free** the allocated memory.

**🔒 Constraints**

* You **must** use **malloc**() and **free**().
* Validate that the memory allocation was successful (i.e., check for NULL).

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main() {

typedef struct {

char name[50];

int salary;

} Employee;

// Allocation

Employee\* employees = malloc(2 \* sizeof(Employee));

if(employees == NULL){

printf("Malloc failed\n");

return 1;

}

for(int i = 0; i<2; i++){

printf("Enter the name for employee #%d: ", i+1);

scanf("%49s", employees[i].name);

printf("Enter salary for employee %d: ", i + 1);

scanf("%d", &employees[i].salary);

}

Employee\* temp = realloc(employees, 5 \* sizeof(Employee));

if (temp == NULL) {

printf("Realloc failed\n");

return 1;

}

for (int i = 2; i < 5; i++) {

printf("Enter name for employee %d: ", i + 1);

scanf("%49s", employees[i].name);

printf("Enter salary for employee %d: ", i + 1);

scanf("%d", &employees[i].salary);

}

printf("\nEmployee List:\n");

for (int i = 0; i < 5; i++) {

printf("%d. Name: %s, Salary: %d\n", i + 1, employees[i].name, employees[i].salary);

}

// Deallocation

free(employees);

return 0;

}

**EX 4 (*Growing a Dynamic List*)**

Our goal is to **build a list** that can store integers and **automatically resize itself** once its capacity is exceeded:

Initial list capacity: 2

Adding item: 10

Adding item: 20

Adding item: 30

--> List is full, resizing to 4

Adding item: 30

Adding item: 40

Adding item: 50

--> List is full, resizing to 6

Adding item: 50

Final list contents (5 items):

10 20 30 40 50

**Q1 – Define the structure**

First define the structure to handle a dynamic list:

typedef struct {

int\* items; // The list of items

int size; // How many items are currently in the list of items

int capacity; // What is the space allocated for the list of items

} List;

**Q2 – Understand the start code**

Then read and understand the start code:

int main() {

  // Create a list and start with enough space for 2 items  
  List myList;

myList.size = 0;  
  myList.capacity = 2;  
  myList.data = **malloc**(myList.size \* sizeof(int));  
  
  // Find out if memory allocation was successful  
  if (myList.data == NULL) {  
    printf("Memory allocation failed");  
    return 1; // Exit the program with an error code  
  }

  // Add 5 items to the list  
  for (int i = 0; i < 5; i++) {  
    **addToList(&myList, i + 1);**  }  
  
  // Display the contents of the list

  printf("Final list content:");  
  for (int j = 0; j < myList.size; j++) {  
    printf("%d ", myList.data[j]);  
  }  
  
  // Free the memory when it is no longer needed  
  **free**(myList.data);  
  myList.data = NULL;  
  
  return 0;  
}

What is the goal of the **size member** in the List structure?

Tracks number of occupied elements in growing list

What is the goal of the **capacity member** in the List structure?

Sets the limit of total elements in list

**Q3 – Implement the addToList function**

Finally write a function addToList

void addToList(List\* theList, int item){

if(theList->size==theList->capacity){

int\* newData = realloc(theList->data, theList->capacity+2 \* sizeof(int));

if (newData == NULL) {

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

return;

}

printf("--> List is full, resizing to %d\n", theList->capacity+2);

theList->data = newData;

theList->capacity = theList->capacity+2;

}

theList->data[theList->size] = item;

theList->size++;

}

#include <stdio.h>

#include <stdlib.h>

typedef struct {

int\* data; // The list of items

int size; // How many items are currently in the list of items

int capacity; // What is the space allocated for the list of items

} List;

void addToList(List\* theList, int item){

if(theList->size==theList->capacity){

int\* newData = realloc(theList->data, theList->capacity+2 \* sizeof(int));

if (newData == NULL) {

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

return;

}

printf("--> List is full, resizing to %d\n", theList->capacity+2);

theList->data = newData;

theList->capacity = theList->capacity+2;

}

theList->data[theList->size] = item;

theList->size++;

}

int main() {

// Create a list and start with enough space for 2 items

List myList;

myList.size=0;

myList.capacity=2;

myList.data = malloc(myList.size \* sizeof(int));

// Find out if memory allocation was successful

if(myList.data == NULL) {

printf("Memory allocation failed");

return 1;// Exit the program with an error code

}

// Add 5 items to the list

for(int i =0; i < 5; i++) {

addToList(&myList, i +1);

}

// Display the contents of the list

printf("Final list content:");

for(int j =0; j < myList.size; j++) {

printf("%d ", myList.data[j]);

}

// Free the memory when it is no longer needed

free(myList.data);

myList.data = NULL;

return 0;

}

is this problem stupid why did it say items then data

The function:

* Adds an integer to the end of the list.
* If the list is full, it uses realloc to increase the capacity by 2 items.

**💡 TIP : ACCESS TO STRUCTURE VIA ARROW OPERATOR :**

You can use [an arrow operator to access your structure members](https://accuweb.cloud/resource/articles/arrow-operator-in-c) from a pointer on structure

\*(myStructure).name // NORMAL SYNTAX

myStructure->name // ARROW OPERATOR SYNTAX

**Q4 – Test it !**

Make sure your code works properly. Try different use cases.

*Congrats, you have made your first dynamic data structure !*

**CREATE**

**PROBLEM (*Inventory system)***



You are building **an inventory system** for a restaurant, a bookstore, or any theme you like.

Each item has a **name**, a **quantity**, and a **value** (or **any other properties** you want to track).

The number of items in the inventory is **not fixed** — your program must be able to grow the list dynamically as new items are added.

Your program shall provide the following features:

1. Add item

2. View inventory

3. Search item // You can specify how to search (by name, id..)

4. Exit

**🔒 Constraints**

* You **must** use **malloc**() and **free**().
* Validate that the memory allocation was successful (i.e., check for NULL).

// #include <stdio.h>

// #include <stdlib.h>

// typedef struct {

// int\* data; // The list of items

// int size; // How many items are currently in the list of items

// int capacity; // What is the space allocated for the list of items

// } List;

// void addToList(List\* theList, int item){

// if(theList->size==theList->capacity){

// int\* newData = realloc(theList->data, theList->capacity+2 \* sizeof(int));

// if (newData == NULL) {

// printf("Memory reallocation failed!\n");

// return;

// }

// printf("--> List is full, resizing to %d\n", theList->capacity+2);

// theList->data = newData;

// theList->capacity = theList->capacity+2;

// }

// theList->data[theList->size] = item;

// theList->size++;

// }

// int main() {

// // Create a list and start with enough space for 2 items

// List myList;

// myList.size=0;

// myList.capacity=2;

// myList.data = malloc(myList.size \* sizeof(int));

// // Find out if memory allocation was successful

// if(myList.data == NULL) {

// printf("Memory allocation failed");

// return 1;// Exit the program with an error code

// }

// // Add 5 items to the list

// for(int i =0; i < 5; i++) {

// addToList(&myList, i +1);

// }

// // Display the contents of the list

// printf("Final list content:");

// for(int j =0; j < myList.size; j++) {

// printf("%d ", myList.data[j]);

// }

// // Free the memory when it is no longer needed

// free(myList.data);

// myList.data = NULL;

// return 0;

// }

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define NAME\_LENGTH 50

#define INITIAL\_CAPACITY 2

typedef struct {

int id;

char name[NAME\_LENGTH];

int quantity;

double unit\_price;

} InventoryItem;

typedef struct {

InventoryItem\* items;

int size;

int capacity;

int next\_id;

} InventorySystem;

void initializeInventory(InventorySystem\* inv) {

inv->size = 0;

inv->capacity = INITIAL\_CAPACITY;

inv->next\_id = 1;

inv->items = malloc(inv->capacity \* sizeof(InventoryItem));

if (inv->items == NULL) {

printf("Malloc Failed\n");

exit(1);

}

}

void freeInventory(InventorySystem\* inv) {

free(inv->items);

inv->items = NULL;

inv->size = 0;

inv->capacity = 0;

}

void resizeInventory(InventorySystem\* inv) {

int new\_capacity = inv->capacity \* 2;

InventoryItem\* new\_items = realloc(inv->items, new\_capacity \* sizeof(InventoryItem));

if (new\_items == NULL) {

printf("Failed to expand inventory\n");

return;

}

printf("--> Inventory capacity expanded from %d to %d\n", inv->capacity, new\_capacity);

inv->items = new\_items;

inv->capacity = new\_capacity;

}

void addItem(InventorySystem\* inv) {

if (inv->size == inv->capacity) {

resizeInventory(inv);

}

InventoryItem new\_item;

new\_item.id = inv->next\_id++;

printf("Enter item name: ");

scanf("%49s", new\_item.name);

printf("Enter quantity: ");

scanf("%d", &new\_item.quantity);

printf("Enter unit price: ");

scanf("%lf", &new\_item.unit\_price);

inv->items[inv->size++] = new\_item;

printf("Item added successfully (ID: %d)\n", new\_item.id);

}

void displayInventory(const InventorySystem\* inv) {

if (inv->size == 0) {

printf("Inventory is empty.\n");

return;

}

printf("\nCurrent Inventory (%d items):\n", inv->size);

printf("ID\tName\t\tQuantity\tUnit Price\tTotal Value\n");

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

double total\_value = 0;

for (int i = 0; i < inv->size; i++) {

const InventoryItem\* item = &inv->items[i];

double item\_value = item->quantity \* item->unit\_price;

total\_value += item\_value;

printf("%d\t%-10s\t%8d\t$%8.2f\t$%8.2f\n",

item->id, item->name, item->quantity, item->unit\_price, item\_value);

}

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

printf("Total inventory value: $%.2f\n\n", total\_value);

}

void searchItem(const InventorySystem\* inv) {

if (inv->size == 0) {

printf("Inventory is empty.\n");

return;

}

printf("Search by:\n");

printf("1. ID\n");

printf("2. Name\n");

printf("Enter choice: ");

int choice;

scanf("%d", &choice);

if (choice == 1) {

int id;

printf("Enter item ID: ");

scanf("%d", &id);

for (int i = 0; i < inv->size; i++) {

if (inv->items[i].id == id) {

const InventoryItem\* item = &inv->items[i];

printf("Found item:\n");

printf("ID: %d\nName: %s\nQuantity: %d\nUnit Price: $%.2f\n",

item->id, item->name, item->quantity, item->unit\_price);

return;

}

}

printf("Item with ID %d not found.\n", id);

}

else if (choice == 2) {

char name[NAME\_LENGTH];

printf("Enter item name: ");

scanf("%49s", name);

printf("Matching items:\n");

int found = 0;

for (int i = 0; i < inv->size; i++) {

if (strcasecmp(inv->items[i].name, name) == 0) {

const InventoryItem\* item = &inv->items[i];

printf("ID: %d, Name: %s, Quantity: %d, Price: $%.2f\n",

item->id, item->name, item->quantity, item->unit\_price);

found = 1;

}

}

if (!found) {

printf("No items named '%s' found.\n", name);

}

}

else {

printf("Invalid choice.\n");

}

}

int main() {

InventorySystem inventory;

initializeInventory(&inventory);

int choice;

do {

printf("\nRestaurant Inventory System\n");

printf("1. Add item\n");

printf("2. View inventory\n");

printf("3. Search item\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

addItem(&inventory);

break;

case 2:

displayInventory(&inventory);

break;

case 3:

searchItem(&inventory);

break;

case 4:

printf("Exiting...\n");

break;

default:

printf("Invalid choice. Try again.\n");

}

} while (choice != 4);

freeInventory(&inventory);

return 0;

}