**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**

After allocate memory(int\* arr) but doesn’t release it when it is no longer needed at the end.

**Q2 –** Fix the code

Add free(arr);

**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**

The original 50 bytes of allocated memory doesn’t get freed before str are pointing to another 100 bytes of allocated memory

**Q2 –** Fix the code

Add free(str) after strcpy(str, Hello);

**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**

The name in the struct doesn’t get freed before you free the struct p

**Q2 –** Fix the code

Add free(p->name); before free(p);

**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>

int main() {

int ratingAmount = 0;

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

scanf(" %d", &ratingAmount);

getchar();

int\* rating = malloc(ratingAmount \* sizeof(int));

if(rating == NULL) {

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

return 1;

}

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

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

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

getchar();

}

printf("Rating Entered : ");

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

printf("%d ", rating[i]);

}

printf("\n");

free(rating);

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>**

**int main() {**

**int rows = 3;**

**int columns = 2;**

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

**if(ptrSeat == NULL) {**

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

**return 1;**

**}**

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

**ptrSeat[i] = malloc(columns \* sizeof(int));**

**if(ptrSeat[i] == NULL){**

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

**return 1;**

**}**

**}**

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

**for(int j = 0; j < columns; j++) {**

**printf("Enter student's ID at row %d, and column %d : ", i, j);**

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

**}**

**}**

**printf("Seating chart : \n");**

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

**for(int j = 0; j < columns; j++) {**

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

**}**

**printf("\n");**

**}**

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

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

**}**

**free(ptrSeat);**

**return 0;**

**}**

**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 <string.h>**

**#include <stdlib.h>**

**typedef struct {**

**char name[50];**

**int salary;**

**} Employe;**

**void checkMemory(Employe\* employe){**

**if(employe == NULL) {**

**printf("Memory allocation faild !\n");**

**exit(1);**

**}**

**}**

**int main() {**

**int employeAmount = 2;**

**Employe \*employe = malloc(employeAmount \* sizeof(Employe));**

**checkMemory(employe);**

**for (int i = 0; i < employeAmount; i++){**

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

**fgets(employe[i].name, sizeof(employe[i].name), stdin);**

**employe[i].name[strcspn(employe[i].name, "\n")] = '\0';**

**printf("Enter employe's salary : ");**

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

**getchar();**

**}**

**int temp = employeAmount;**

**employeAmount = 5;**

**employe = realloc(employe, employeAmount \* sizeof(Employe));**

**checkMemory(employe);**

**for(int i = temp; i < employeAmount; i++){**

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

**fgets(employe[i].name, sizeof(employe[i].name), stdin);**

**employe[i].name[strcspn(employe[i].name, "\n")] = '\0';**

**printf("Enter employe's salary : ");**

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

**getchar();**

**}**

**for(int i = 0; i < employeAmount; i++){**

**printf("Employe %s salary %d\n", employe[i].name, employe[i].salary);**

**}**

**free(employe);**

**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?

The size is for tracking the amount of the items that is currently in the list

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

The capacity is the amount of the max item that your list can hold.

**Q3 – Implement the addToList function**

Finally write a function addToList

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

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 !*

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

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;

void expandCapacity(List\* theList, int expandAmount){

int newCompacity = theList->**capacity** + expandAmount;

int\* ptrtheList = realloc(theList->**items**, newCompacity \* sizeof(int));

if(ptrtheList == NULL) {

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

exit(1);

}

theList->**items** = ptrtheList;

theList->**capacity** = newCompacity;

}

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

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

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

expandCapacity(theList, 2);

}

printf("Adding item : %d\n", item);

theList->**items**[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.**items** = malloc(myList.**capacity** \* sizeof(int));

*// Find out if memory allocation was successful*

if (myList.**items** == 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.**items**[j]);

}

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

free(myList.**items**);

myList.**items** = NULL;

return 0;

}

**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>**

**#include <string.h>**

***// item struct***

**typedef struct {**

**char name[50];**

**int id;**

**int quantity;**

**int value;**

**} Item;**

***// Inventory struct***

**typedef struct {**

**Item\* item;**

**int size;**

**int capacity;**

**} Inventory ;**

***// extea function for help***

**void searchMenu() {**

***// let you choose to search between name or id***

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

**printf("1. Search by name\n"); *// if search by name, id == NULL***

**printf("2. Search by ID\n"); *// if search by id, name == NULL***

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

**}**

**void feat() {**

***// all the features for this exercise***

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

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

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

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

**printf("4. exit\n");**

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

**}**

**void printItem(Item item) {**

***// print item info***

**printf("Item name : %s, Item ID : %d, Value : %d, Quantity : %d\n",**

**item.name,**

**item.id,**

**item.value,**

**item.quantity**

**);**

**}**

**void expandCapacity(Inventory\* theInventory, int expanAmount) {**

***//expnan the inventory capacity if we are at the limit***

**int newCapacity = theInventory->capacity + expanAmount;**

**Item\* ptrInventory = realloc(theInventory->item, newCapacity \* sizeof(Item));**

**if(ptrInventory == NULL) {**

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

**exit(1);**

**}**

**theInventory->item = ptrInventory;**

**theInventory->capacity = newCapacity;**

**printf("Inventeory capacity full, expan to %d\n", newCapacity);**

**}**

***// all the features function***

**void viewInventory(Inventory\* theInventory) {**

***// view inventory***

**printf("The amount of item in the inventory : %d\n", theInventory->size);**

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

**printItem(theInventory->item[i]);**

**}**

**}**

**void searchItem(Inventory\* theInventory, const char\* nameSearch, int idSearch){**

***// search item by id or name***

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

**if(strcmp(nameSearch, theInventory->item[i].name) == 0 || idSearch == theInventory->item[i].id) {**

**printItem(theInventory->item[i]);**

**}**

**}**

**}**

**void addItem(Inventory\* theInventory){**

***// add item to the inventory***

**if(theInventory->size == theInventory->capacity) expandCapacity(theInventory, 2);**

**Item addItem;**

***// enter the item name that you are adding***

**printf("Enter item's name : ");**

**fgets(addItem.name, sizeof(addItem.name), stdin);**

**addItem.name[strcspn(addItem.name, "\n")] = '\0';**

***// initailize the item id to size + 1***

**addItem.id = theInventory->size + 1;**

***//enter the item's value***

**printf("Enter the item's value : ");**

**scanf(" %d", &addItem.value);**

**getchar();**

***// enter the item's quantity***

**printf("Enter the item quantity : ");**

**scanf(" %d", &addItem.quantity);**

**getchar();**

**theInventory->item[theInventory->size] = addItem;**

**theInventory->size++ ;**

**printf("Add item successfully!\n");**

**}**

**void searchFeat(Inventory\* theInventory) {**

***// search item***

**int choice = 0;**

**searchMenu();**

**printf("Enter your choice : ");**

**scanf(" %d", &choice);**

**getchar();**

**switch (choice) {**

**case 1 : *// search by name***

**char searchName[50];**

**printf("Enter your item's name : ");**

**fgets(searchName, sizeof(searchName), stdin);**

**searchName[strcspn(searchName, "\n")] = '\0';**

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

**if(strcmp(searchName, theInventory->item[i].name) == 0) {**

**printItem(theInventory->item[i]);**

**return;**

**}**

**}**

**printf("There's no item name : %s\n", searchName);**

**break;**

**case 2 : *// search by ID***

**int idSearch;**

**printf("Enter you item's ID : ");**

**scanf(" %d", &idSearch);**

**getchar();**

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

**if(idSearch == theInventory->item[i].id) {**

**printItem(theInventory->item[i]);**

**return;**

**}**

**}**

**printf("There's no item with ID : %d\n", idSearch);**

**break;**

**}**

**}**

**int main(){**

**Inventory inventory;**

**inventory.size = 0;**

**inventory.capacity = 10;**

**inventory.item = malloc(inventory.capacity \* sizeof(Item));**

**if(inventory.item == NULL) {**

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

**return 1;**

**}**

**int choice = 0;**

**while(choice != 4){**

**feat();**

**printf("Enter your choice : ");**

**scanf(" %d", &choice);**

**getchar();**

**switch (choice) {**

**case 1 : *// add item***

**addItem(&inventory);**

**break;**

**case 2 : *// Vieww inventory***

**viewInventory(&inventory);**

**break;**

**case 3 : *// Search Item***

**searchFeat(&inventory);**

**break;**

**}**

**}**

**free(inventory.item);**

**inventory.item = NULL;**

**inventory.size = 0;**

**inventory.capacity = 0;**

**return 0;**

**}**