

C1-S3 PRACTICE

ALLOCATE MEMORY DYNAMICALLY

- At the end of this practice, you should be able to...
 - ✓ Manipulate pointers
 - ✓ Understand and implement swapping by **value** and by **reference**.
 - ✓ Understand the difference between **stack** and **heap** memory.
 - ✓ Perform dynamic memory allocation and deallocation using **new** and **delete**.

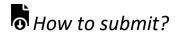
Assuming your file is named: exercise.cpp:

- ✓ Open a terminal at your file location
- ✓ Compile your Program using the following command

g++ -o exercise exercise.cpp

✓ Run Your Program using the following command

./exercise



- ✓ Make a report PDF containing the screenshot of your program code and output for each exercise.
- ✓ Submit your final source code report PDF to Microsoft team and turn it in.

Are you lost?

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

✓ C++ Memory management

EXERCISE 1

Learning objectives:

- Understand how pointers work in C++
- Practice the concepts of passing arguments by value, by reference, and by pointer in C++

In this exercise, you need to implement different methods of swapping two integer values in C++

- Swapping by values
- Swapping by references
- Swapping by pointers
- Q1 Run the swapByValue() function to swap a and b by value.
 - Was the swap successful? YES / NO
 - Explain why:
- Q2 Implement the swapByReference() function to swap a and b by reference.
 - Was the swap successful? YES / NO
 - Explain why:
- Q3 Implement the swapByPointer() function to swap a and b by reference.
 - Was the swap successful? YES / NO
 - Explain why:
- Q4 Look at the log related to A and B address after each swap operation
 - Do the addresses change after swapping? YES / NO
 - Explain why:

Q5 - (FOR FUN)

In the swapping algorithm, we need **temp** as a temporary variable. Can we swap two values without using the **temp** variable (only a & b are needed)?

EXERCISE 2

Learning objectives:

- ✓ Understand dynamic memory allocation in C++.
- ✓ Practice using structures to store related data.
- ✓ Implement input validation and basic operations on array elements.

In this exercise, you will need to manage a list of items to sell.

Each item has an id, a name and a price:

```
struct Item
{
    int id;
    std::string name;
    double price;
};
```

You challenge is to manage this list of items dynamically in memory, to adjust the array size at runtime based on user input, optimizing memory usage without needing to reserve a fixed amount of space.

- Q1 In the *main*, input the number of items (N) to manage. Ensure that the number entered is positive.
- Q2 In the main, allocate an array of Items dynamically.
- Q3 In the main, prompt user to enter the information for each of the N items
- Q4 Draw a memory map showing the state of the stack and heap after entering the N item information
- **Q5** Use <u>sizeof(type_name)</u> to compute the size of the items array in memory
- **Q6** Print each element of the Items array.
- Q6 Free the allocated memory for the Items array stored in the heap

EXERCISE 3

This exercise is based on your last practice exercise on students and scores. You can review it before.

Learning objectives:

- ✓ Create an array of student struct with a given size at runtime
- ✓ Parse and store student data in a dynamic array and store it in the heap.
- ✓ Compute the average score for each student.

You challenge is to manage this **list of students dynamically** in memory, but also the **list of student scores.**

- **Q1** Write code to prompt the user to enter the number of students. Ensure that the number entered is positive.
- **Q2** Implement code to dynamically allocate an array of Student structs based on the number of students entered by the user.

```
struct Student {
   int id;
   std::string name;
   int* scores;
   int numScores;
};
```

- Q3 Write Code to Populate Student Data:
 - ✓ For each student, write code to:
 - ✓ Prompt the user to enter the student's ID and name.
 - ✓ Prompt the user to enter the number of scores for the student (numScores), ensuring the number is positive.
- Q4 Write a code Allocate an array of scores for each student with a size of numScores
- **Q5** Write code to display the ID, name, and average score of each student in a formatted table.
 - ✓ Ensure the average score is displayed with two decimal places.
 - You should display the below output:

Student ID Name Average Score			
1	1	Alice	87.40
1	2	Bob	78.60
1	3	Charlie	90.40
1	4	David	70.00
	5	Eve	67.20

Q6 – Write Code to Free Allocated Memory: