Dynamic Memory

<div style="border:1px solid black; padding:10px; background-color:#e8e8e0;">

Objectives

1. Dynamic Memory
2. Freeing Memory
3. Linked Lists Introduction
4. Sorting (Basic)
5. Exercise

</div>

1. Dynamic Memory

What is static memory allocation?
When we declare variables, we are preparing the variables that will be used. This way a compiler can know that the variable is an important part of the program. So, the compiler allocates spaces to all these variables during compilation of the program. **The variables allocated using static memory allocation will be stored in *STACK*.**

But wait, what if the input is not known during compilation? Say, you want to pass input in the command line arguments (During runtime). The size of the input is not known beforehand during compilation.

The need for dynamic memory!!
We suffer in terms of inefficient storage use and lack or excess of slots to enter data. This is when dynamic memory play an important role. This method allows us to create storage blocks or room for variables during run time. Now there is no wastage. **The variables allocated using dynamic memory allocation will be stored in *HEAP*.**

<div style="border:1px solid black; padding:10px; background-color:#fbe2a0;">

Tip!
Fine dining restaurant vs drive thru
Most of the times we reserve space at a fine dining restaurant by calling them up or reserving online. (The space is wasted if you do not show up even after reserving). The restaurant can also not accommodate more than its capacity. (Static memory)

Consider McDonalds drive-thru, you don't reserve space, you will somehow find a spot in the queue and just manage to grab your order. Here you are not reserving any seats beforehand, but still you can manage to get a meal. The restaurant can serve multiple customers even if they have not reserved. (Dynamic memory)

</div>

Dynamic memory allocation example.

```cpp
int main()
{
   //Dynamic memory allocation
   int *ptr1 = new int;
```

```
    int *ptr2 = new int[10];
}
```

2. Freeing Memory

Once a programmer allocates memory dynamically as shown in the previous section, it is the responsibility of the programmer to delete/free the memory which is created. If not deleted/freed, even though the variable/array is still not used, the memory will continue to be occupied. This causes a **memory leak.**

To delete a variable **ptr1** allocated dynamically

```
delete ptr1;
```

To delete an array ptr2 allocated dynamically

```
delete [] ptr2;
```

3. Linked List Introduction

A linked list is a data structure that stores a list, where each element in the list points to the next element.

Let's elaborate:

Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array. Following are the important terms to understand the concept of Linked List.
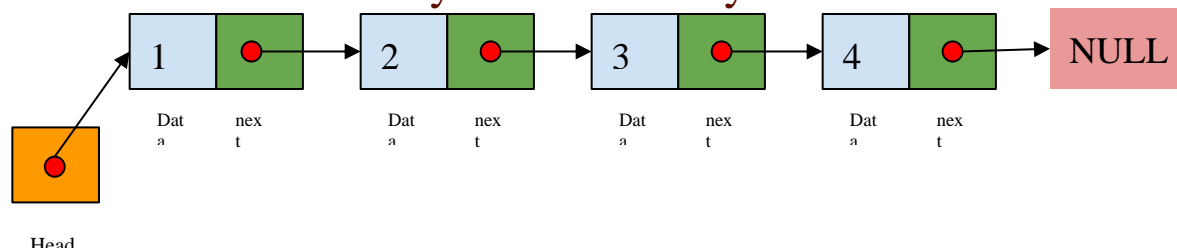
- **Node** – Each Node of a linked list can store data and a link.

- **Link** – Each Node of a linked list contains a link to the next Node (unit of Linked List), often called 'next' in code.

- **Linked List** – A Linked List contains a connection link from the first link called Head. Every Link after the head points to another Node (unit of Linked List). The last node points to NULL.

Dynamic Memory

In code, each node of the linked list will be represented by a class or struct. These will, at a minimum, contain two pieces of information - the data that node contains, and a pointer to the next node. Example code to create these is below:

```
class Node
{
public:
        int data;
        Node *next;
};
```

```
struct node
{
    int data;
    Node *next;
};
```

4. Sorting – Bubble Sort

Bubble sort is a simple sorting algorithm. It is based on the idea of repeatedly comparing pairs of adjacent elements and then swapping their positions if they exist in the wrong order.

Say, A[] is an unsorted array of n elements. We want it sorted in ascending order.

Pseudocode:

```
void bubble_sort(int A[], int n ) {

    int temp;
    for(int k = 0; k< n-1; k++) {

    // Last k elements are already in place
        for(int i = 0; i < n-k-1; i++) {

            if(A[i] > A[i+1] ) {
                temp = A[i];
                A[i] = A[i+1];
```
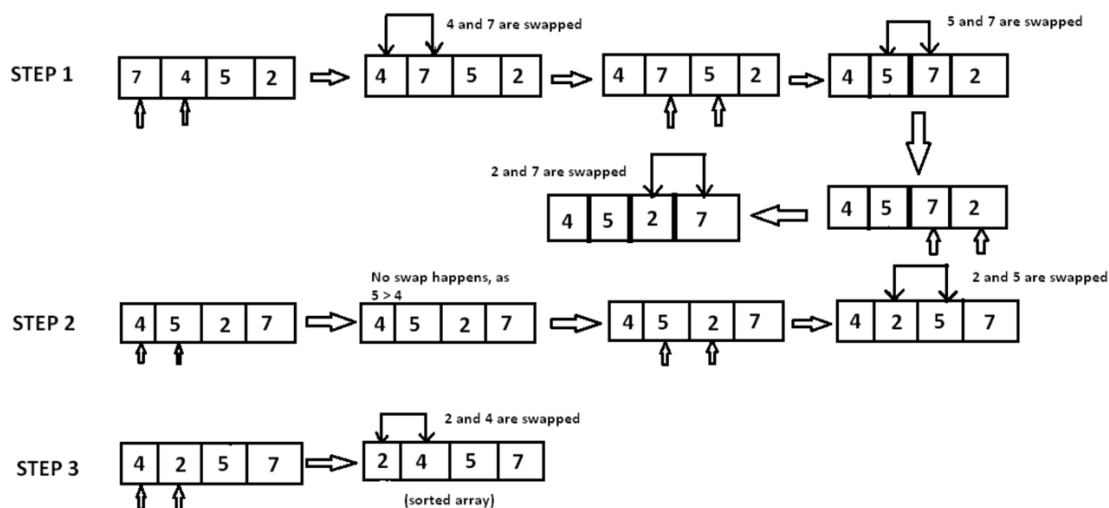
```
            A[i + 1] = temp;


          }
        }
    }
}
```

Example with A[] = {7,4,5,2}

5. Exercise

Open your exercise file and complete the TODOs in the C++ program which does the following:
1. It will read from a text file. Each line of the file contains a number. Provide the file name as a command line argument. Number of lines in the file is not known.
2. Create an array dynamically of a capacity (say 10) and store each number as you read from the file.
3. If you exhaust the array but yet to reach end of file dynamically resize/double the array and keep on adding.