CS214-Data Structure

Lecturer: Dr. Salwa Osama

Introduction

What is Data Structure?

- Data is the basic entity or fact that is used in calculation or manipulation process
- Data structure is a way of organizing data items by considering its relationship to each other.

Why Data Structure is important?

- The selection of **good data structure** will help the programmer to design more **efficient programs.**

- The efficiency of the program depends on two measurements
  1. Space complexity
  2. Time complexity

Why Data Structure is important?

- The time complexity can now be expressed as function of number of key operations performed.

- One solution may require more space but less time while the other requires less space but takes more time. Thus, we may have to sacrifice one at the cost of the other.

Why Data Structure is important?

- The space needed by a program:
  - Instruction space
  - Data space
  - Environment stack space:

  The data saved on the environment stack is:

  (a) Return address

  (b) Values of all lead variables and the values of formal parameters in the function being invoked.

Memory Management

- Memory allocation is a fundamental concept in programming, especially in languages like C where memory management is done explicitly by the programmer.
- There are two primary types of memory allocations :

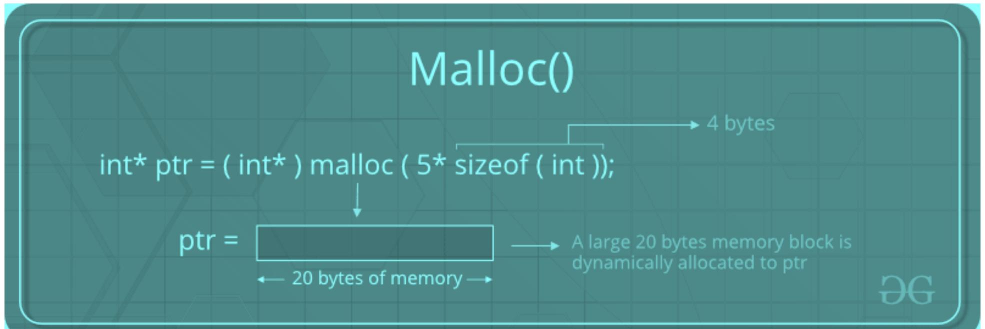
1. Static memory allocation or Compile time

   EX: float *a[5], f;*

2. Dynamic memory allocation or Run time

   EX: int* ptr = (int *) malloc (10 * sizeof (int));

Demo
Static Memory
Dynamic Memory


Malloc()
int* ptr = (int*) malloc (5* sizeof (int));
4 bytes
ptr =
20 bytes of memory
A large 20 bytes memory block is dynamically allocated to ptr

free(ptr);
To avoid memory leaks

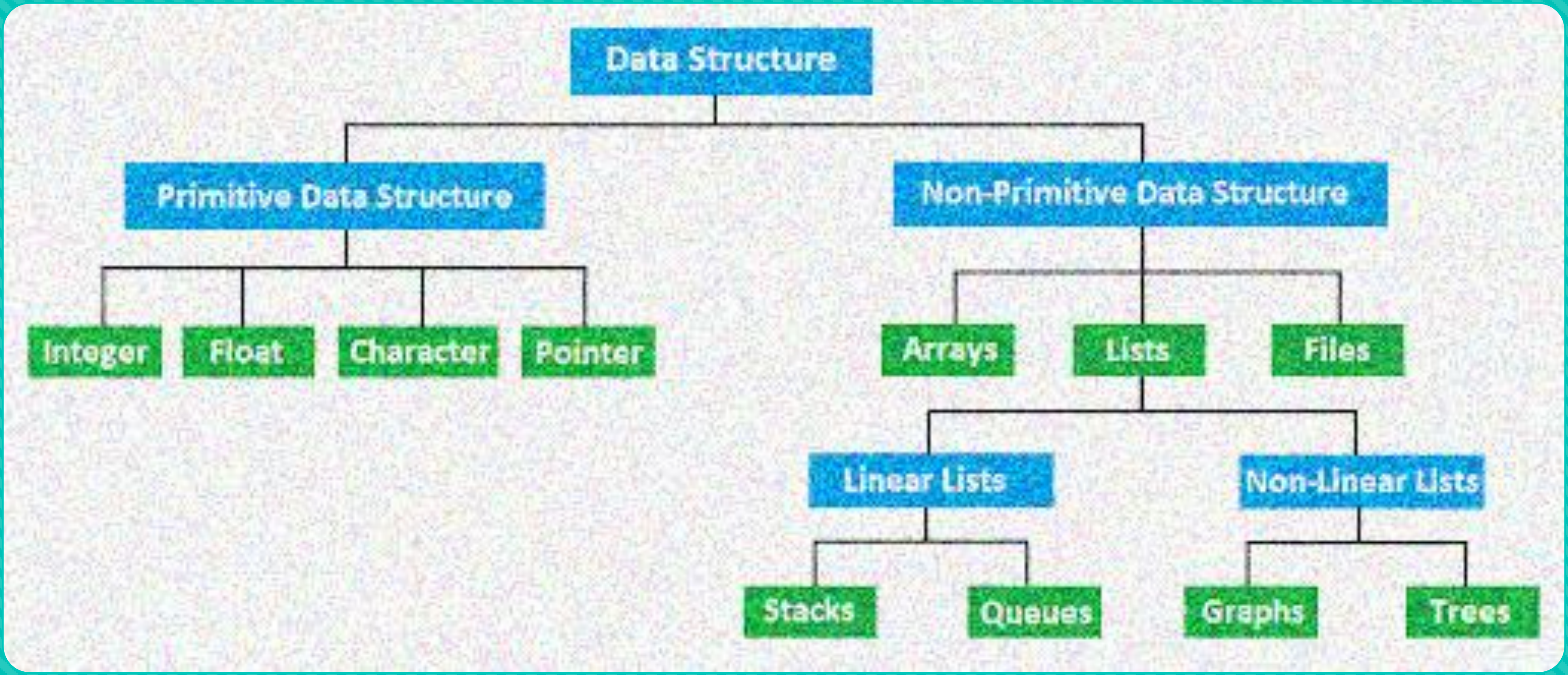
Memory Management

Demo
Primitive Type

EX:*struct Employee{*

```
        int Emp_Code;
        char Emp_Name[50];
        float Emp_Salary;
  }
  struct Employee *str_ptr = (struct Employee *) malloc(sizeof (struct Employee));
```

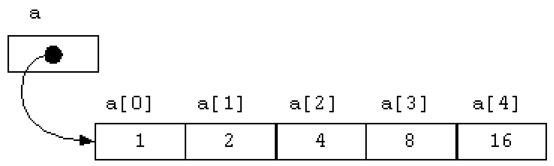
When this statement is executed, a contiguous block of memory of size 60 bytes will be allocated.



Have you ever tried to structure your data before?

Arrays

- **Array** is one of the most familiar non-primitive linear **data structure.**



- It is a collection of items –which are of the same type- stored contiguously in the memory.

- We used to deal with the Array **without thinking about its details.**

Arrays

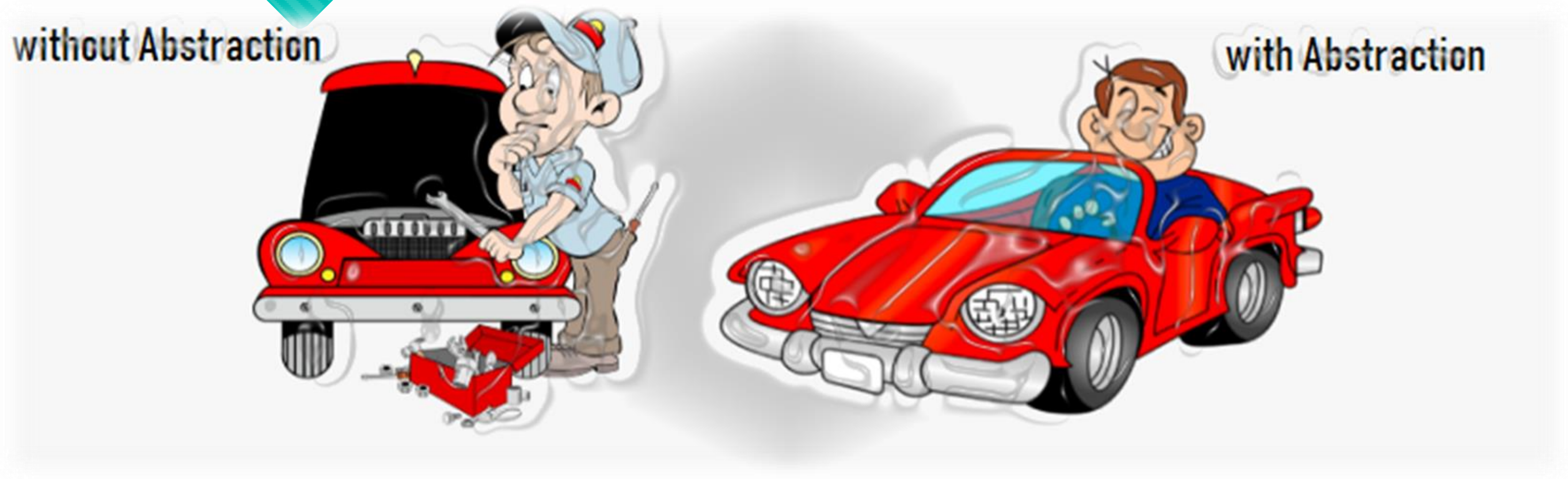
- The c statement: `int A[10];`

  means:
  - reserving a contiguous space in memory, so that:
    memory size = element size * number of elements
  - giving the starting address the name `A`
- The c statement : `A[3] = 27;`

  means:
  - calculates the location address:

    Loc address = A+3* sizeof(int)
  - stores 27 in that location.

Abstraction

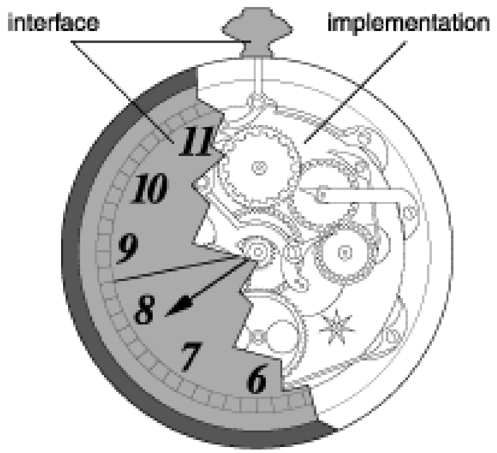


Focusing on essential features while hiding unnecessary implementation details.

This enables developers to work at higher levels of abstraction, improving system design, modularity, and ease of maintenance.

Abstract Data Type (ADT)

- **Data abstraction** allows us to use a *data structure without* considering how it is implemented.



- **Abstract Data Type (ADT)**

  Is an organized data

  object and a set of

  operations for manipulating it

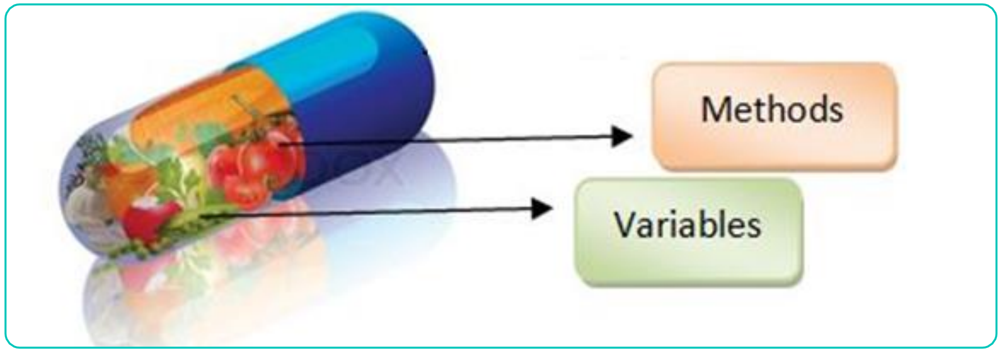
ADT = Organized data + Operations

Abstract Data Type (ADT)

- **ADT** is defined in terms of the operations that can be performed on instances of the type rather than by how those operations are actually implemented.
- In other words, an ADT defines the ***interface*** of the objects.
- The interface is considered a kind of **contract** between the implementers and the users of the ADT.

Information hiding (Encapsulation)

- The hiding of the data structure implementation inside the ADT is referred to as ***encapsulation*** *or* ***information hiding***.



Information hiding (Encapsulation)

- We refer to a program that uses an ADT as **user** and a program that specifies the ADT as an **implementation**
- You use the structure at the **"User Level"** without caring about the details at the **"Implementation Level".**
- The user level, does not change even if the implementation of the used structure is changed.

Why using ADT?

- Rather than having to understand the detailed implementation of the set operations, the user only has to study the interface at a much higher level so much time can be saved.
- The ADT can be used in a variety of different programs
- The implementation of the component can be changed without affecting some other component.

Any Questions