Chapter 1: Introduction

By Er. Kushal Ghimire

Review of Array, Structure, Union, Class, Pointer

Brief: Array, Structure, Union, Class, Pointer

Array = collection of homogeneous data in a contiguous memory.

String = an array of characters.

Structure = collection of heterogeneous data in a non contiguous memory.

Union = similar to structure except the memory allocation is of the highest size among structure member.

Class = blueprint or model of an object.

Pointer = a variable that stores memory address that it is pointing to instead of a value.

Store Single Value

- Variable
  - The value stored in a variable can be manipulated/changed.
- Constant
  - The value stored in a constant cannot be manipulated/changed.
- Macro Definition
  - Preprocessor Directive puts the value of a macro in every place of your program where you have used the macro.

```c
#include <stdio.h>
// Macro Definition
// N is a macro
#define N 5

int main()
{
    // Variables = x, y and z
    int x = 8;
    float y = 8.8;
    char z = 'K';

    // Constants = a, b and c
    const int a = 8;
    const float b = 8.8;
    const char c = 'K';

    return 0;
}
```

Store Multiple Value

Homogeneous Collection

- Array
- String

```c
#include <stdio.h>

int main()
{
    // Arrays
    int numbers[] = {11, 22, 33, 44, 55};
    float marks[] = {11.1, 22.2, 33.3, 44.4, 55.5};

    // Strings
    char name[] = "Kushal Ghimire";
    char my_name[] = {'K', 'u', 's', 'h', 'a', 'l'};

    return 0;
}
```

Store Multiple Value

Heterogeneous Collection

- Structure
- Union

```c
#include <stdio.h>

// Structure
struct student
{
    int roll_no;
    char name[100];
    float marks;
};

int main()
{
    // One Student
    struct student s1;

    //Forty Eight Student
    struct student all_students[48];

    return 0;
}
```

```c
#include <stdio.h>

// Union
union student
{
    int roll_no;
    char name[100];
    float marks;
};

int main()
{
    // One Student
    union student s1;

    //Forty Eight Student
    union student all_students[48];

    return 0;
}
```

Pointer

- points to the memory location.
- normal variable stores value, pointer variable stores memory location/address.
- can access value through memory location using memory dereferencing.

```c
#include <stdio.h>

int main()
{
    // Variable
    int n1 = 17;
    float f1 = 17.7;

    // Pointer
    int *p1 = &n1;
    float *p2 = &f1;

    // Dereferencing: *p1 *p2
    printf("\np1 = %i and p2 = %i", p1, p2);
    printf("\n*p1 = %d and *p2 = %f", *p1, *p2);

    return 0;
}
```

```
p1 = 1820629632 and p2 = 1820629636
*p1 = 17 and *p2 = 17.700001

...Program finished with exit code 0
```

Class

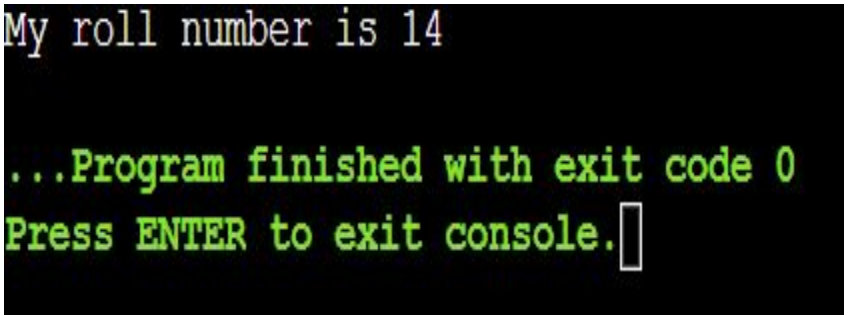
```cpp
#include <iostream>

class Student
{
    public:
    // data member
        int roll;
        char name[100];
        float marks;

    public:
    // methods
        void display_roll()
        {
            printf("My roll number is %d", this->roll);
        }
};

int main()
{
    Student s1, s2;
    s1.roll = 14;
    s1.display_roll();

    return 0;
}
```



Data Structure and its Classification

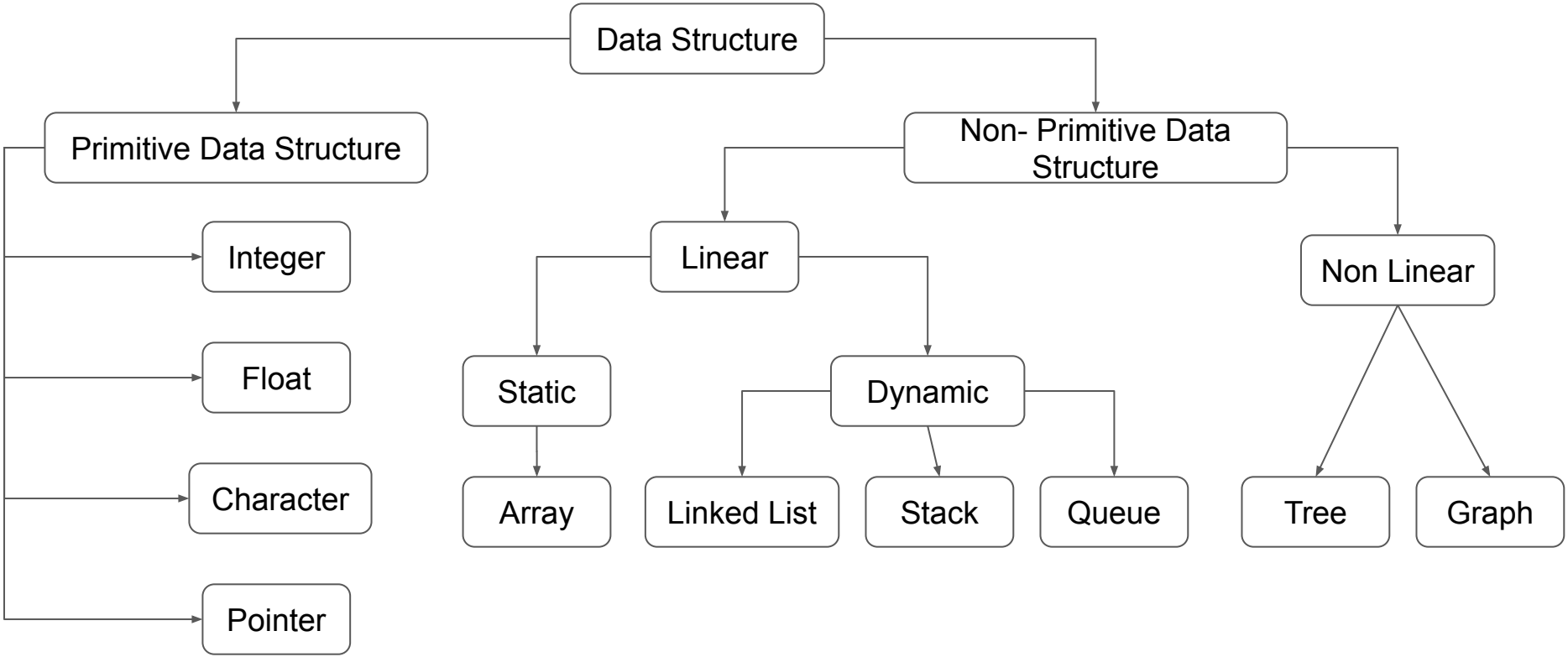
Data Structure

- The raw facts or figures are termed as data i.e a data is a single unit of value.
- Data structure is the structural representation of logical relationships between elements of data.
- Data structure affects the design of both the structural and functional aspects of a program.

Data structure + Algorithm = Program

- The representation of a particular data structure in the memory of a computer is called a storage structure i.e. a data structure should be represented in such a way that it utilizes maximum efficiency.
- The data structure can be represented in both main and auxiliary memory of the computer. A storage structure representation in auxiliary memory is often called a file structure.
- In short, Data structure is an agreement about:
    - How to store a collection of objects in memory ?
    - What operation can we perform on that data ?
    - What is the suitable algorithm for the desired operations ?
    - How time and space efficient those algorithms are ?

Classification of data structure



Classification of data structure

Primitive Data Structure

- These are the basic data structures and are directly operated upon by the machine instructions, which is in a primitive level.
- They are integers, floating point numbers, characters, string constants, pointers etc.

Non-Primitive Data Structure

- It is a more sophisticated data structure emphasizing on structuring of a group of homogeneous (same type) or heterogeneous (different type) data items.
- Array, list, files, linked list, trees and graphs fall in this category.

Abstract Data Type

Abstract Data Type [ADT]

- ADTs are like user defined data types which defines operations on values using functions without specifying what is there inside the function and how the operations are performed.
- An abstract data type is a data declaration packaged together with the operations that are meaningful on the data type.
- In other words, we can encapsulate the data and the operation on data and we hide them from user.
- **There are multiple ways to implement an ADT.**
- Example: STACK ADT has the following operations:
  - initialize() : initialize an empty array.
  - push() : insert an element into the stack.
  - pop() : delete an element from the stack.
  - peek() : display the top most element in the stack.
  - isEmpty() : checks if stack is empty.
  - isFull() : checks if stack is full.

Examples of ADT

1. Linear ADTs:
    a. Stack (last-in, first-out) ADT
    b. Queue (first-in, first-out) ADT
    c. Lists ADTs
        i. Arrays
        ii. Linked List
        iii. Circular List
        iv. Doubly Linked List

2. Non Linear ADTs:
    a. Tree
        i. Binary Search Tree [BST]
    b. Heap
    c. Graph
        i. Directed
        ii. Undirected
    d. Hash table