PROGRAMMING CONCEPTS

Solid programming foundation

1. Variables
2. Control Structures
3. Data Structures
4. Syntax
5. Tools
6. READ ON MOTIVATION!

Variables

What is a Variable?

Variables are names given to computer memory locations in order to store data in a program. This data can be known or unknown based on the assignment of value to the variables.

Variables can also be considered as 'containers' which are used to hold more than one value. Their sole purpose is to label and store data in the memory and then this Variable can be accessed throughout the program whenever needed.

Generally, the variable name is used as a reference to access the stored value.

Creating Variables

The process of creating a variable is also known as Declaration of Variable. Every programming language has its own way of declaring a variable. Declaration of a variable does 2 things:

1. It tells the compiler what the variable name is.
2. It specifies what type of data the variable will hold.

In general, the declaration consists of 3 parts – the variable type, variable label/name, and semi-colon to end the declaration.**Syntax** – data_type variable_name = value ;

For example: –

int speed; (Declaring a variable)

float sum = 0; (Declaring and initialising a variable sum to 0)

string my_variable = "Nipun Arora";

There are a few conventions which are needed to be followed while declaring Variables –

1. Variable names must begin with a letter, underscore, non-number character. Each language has its own conventions.
2. Few programming languages like PHP, Python, Perl, etc. do not require to specify data type at the start.
3. Do not use a comma with numbers.
4. Once a data type is defined for the variable, then only that type of data can be stored in it. For example, if the variable is declared as Int, then it can only store integer values.
5. A variable name once defined can only be used once in the program. You cannot define it again to store other types of value.
6. Built-in Data types are frequently used to declare the data type of the Variable.

Control Structures: -

Control structures allow you to change the flow of execution of program instructions.

This gives us a clue, but we will explain it a little easier. Control structures are a set of lines of code that can control a decision to be taken by the program, for example, your to read a particular book you find some text that says "if you want to know what happened to X character sees the Page X of the book, if you want to know the full story of X position X go to the page of the book. "In this situation the reader takes the" decision "they should do, though programming, the program should make that decision based on certain" rules "that the programmer wrote. In many programming decisions are made, if satisfied that sees this line of code but go to these other lines of code, for example:

```
if (price <500) {

RemoveProduct ();

} else {

ContinueProducto ();

}
```

in this example we use variables as you can see, tells us that if the variable price (the value where points) is less than 500 then programming sequence jumps at a given location code (in this case a method, a set of lines of code we can send for) RemoveProduct (); but is greater than 500 then will go to another location code to run another method called ContinueProduct (); this is a control structure, in specific, is a control structure called if – else.

There are other control structures, such as while.

```
while (age <18) {

displayMessage ("You are still a child");
age = age + 1;

}
```

in this example, while the condition is met, age variable is less than 18, the same code between the braces is executed once not satisfy the condition then proceed to the next line of code after this control structure (after the stopcock).

Other examples of control structures would, do – while

```
do {

displayMessage ("You still is a child");
age = age + 1;

} while (age <18);
```

for control structure,

```
for (age = 0; age <18; age ++) {

      displayMessage ("My age is" + age);

}
```

where a cycle of code is from 0 (age = 0) to 18 (age <18) with an increment of 1 (age ++) according to the conditions set and a message is displayed on each repetition, also this structure switch / Control case

```
 switch (age) {

      case 5:
            displayMessage ("I am a child");
            break;
      case 14:
            displayMessage ("I'm a teenager");
            break;
      case 18:
            displayMessage ("I'm an adult");
            break;
      case 60:
            displayMessage ("I am an old man");
            break;
}
```

were based on the value of the variable age is showing a message or another.

Data Structures: -

A data structure is a way of organizing a set of basic data in order to facilitate handling.

Let’s explain a little more depth, suppose you need to keep information from contacts in your address book in a program, what would you do? I probably would happen to create some variables to store this information, something like

```
contact1 = "Nipun Arora (123456789)";
contact2 = "Jasmine (123456789)";
contact3 = "Jenifer (123456789)";
contact4 = "Kimwoobin (123456789)";
contact5 = "Jimin (123456789)";
```

Perfect! Right? sorry this has two drawbacks:

- Amount of code

It’s okay if they are 10 or maybe 20 contacts / variables but how about your book has over 500 contacts?, or if you want to save contacts but are not records of students from a university with thousands of records! This is not right or for the programmer who has to write or for the program.

- Flexibility

If we add another contact, we cannot do it without necessarily modifying the source code, which must always avoid this simply something serious nuisance.

Solution.

The use of data structures greatly facilitates this task. In java there is a data structure called **List**, looks like this.

```
List contacts = new ArrayList()
```

Forgetting some technical issues, a List allows us to store a set of data within a single variable, we can add contacts in the following manner.

```
contacts.add("Nipun Arora Insa (123456789)");
```

You may be saying that looks the same way that using variables and true, with a couple of differences, using this data structure you are only creating one single variable unlike the thousands who would have to be created using variables, also is more flexible the Using these data structures because we can easily insert or delete records without touching anything in your code.

In short, a data structure allows us to organize information optimally and avoid creating countless variables.

There are other control structures as hash maps, which store information in the form key -> value, but the work completed to finish today, I think you've learned what a data structure.

Syntax: -

Syntax refers to the rules that define the structure of a language. Syntax in computer programming means the rules that control the structure of the symbols, punctuation, and words of a programming language.

If the syntax of a language is not followed, the code will not be understood by a compiler or interpreter.

Compilers convert programming languages like Java or C++ into binary code that computers can understand. If the syntax is incorrect, the code will not compile.

Interpreters execute programming languages such as JavaScript or Python at runtime. The incorrect syntax will cause the code to fail.

That's why it is crucial that a programmer pays close attention to a language's syntax. No programmer likes to get a syntax error.

Tools: -

There are hundreds or maybe thousands of tools for programmers, but certainly the most important are the IDEs, I leave a list of the best IDEs for most programming languages.

- Java – Netbeans, Eclipse
- C# – Visual Studio, SharpDevelop
- Objective-C – Xcode
- Delphi – RAD Studio
- Object Pascal – Delphi Lazarus
- C, C ++ – Visual Studio , Vim
- PHP – Eclipse, Netbeans, Nusphere PHPed
- Python – Eclipse, IDLE
- Perl – Padre
- Ruby – TextMate
- Visual Basic – Visual Studio

I know we are missing several but since this is an introduction to basic concepts of programming and not know what language you're about to learn these leave you alone.

Object oriented Programming

Object Oriented Programming, often referred to as **OOP**, is a programming paradigm that was created to deal with the growing complexity of large software systems. Programmers found out very early on that as applications grew in complexity and size, they became very difficult to maintain. One small change at any point in the program would trigger a ripple effect of errors due to dependencies throughout the entire program.

Programmers needed a way to create containers for data that could be changed and manipulated without affecting the entire program. They needed a way to section off areas of code that performed certain procedures so that their programs could become the interaction of many small parts, as opposed to one massive blob of dependency.

In simple words the object-oriented programming is a different approach to programming. It has been created with a view to increase programmer's productivity by overcoming the weaknesses found in procedural programming approach. Over the years many object-oriented programming languages such as C++, Java have come up and are becoming quite popular in the market. The major need for developing such languages was to manage the ever-increasing size and complexity of programs.

- **Class:**
  A class is a user-defined data type. It consists of data members and member functions, which can be accessed and used by creating an instance of that class. It represents the set of properties or methods that are common to all objects of one type. A class is like a blueprint for an object.
  For Example: Consider the Class of Cars. There may be many cars with different names and brands but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range, etc. So here, Car is the class, and wheels, speed limits, mileage are their properties.

- **Object:**
  It is a basic unit of Object-Oriented Programming and represents the real-life entities. An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.
  An object has an identity, state, and behavior. Each object contains data and code to manipulate the data. Objects can interact without having to know details of each other's data or code, it is sufficient to know the type of message accepted and type of response returned by the objects.
  For example, "Dog" is a real-life Object, which has some characteristics like color, Breed, Bark, Sleep, and Eats.

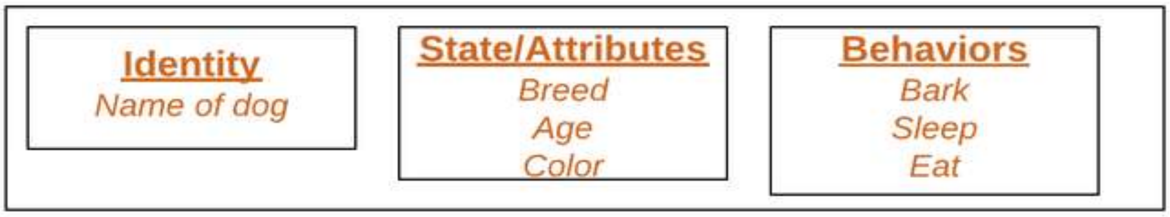


Fig 1: table

The four basics of object-oriented programming

Object-oriented programming has four basic concepts: encapsulation, abstraction, inheritance, and polymorphism. Even if these concepts seem incredibly complex, understanding the general framework of how they work will help you understand the basics of an OOP computer program. Below, we outline these four basic principles and what they entail:

Encapsulation
Abstraction
Inheritance
Polymorphism

1. Encapsulation

The word, “encapsulate,” means to enclose something. Just like a pill "encapsulates" or contains the medication inside of its coating, the principle of encapsulation works in a similar way in OOP: by forming a protective barrier around the information contained within a class from the rest of the code.

In OOP, we encapsulate by binding the data and functions which operate on that data into a single unit, the class. By doing so, we can hide private details of a class from the outside world and only expose functionality that is important for interfacing with it. When a class does not allow calling code access to its private data directly, we say that it is well encapsulated.

In simple words Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Another way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.

Wrapping up of data and functions into a single unit is called as data encapsulation. Encapsulation is the most basic concept of OOP. It is the way of combining both data and the functions that operate on that data under a single unit. The only way to access the data is provided by the functions (that are combined along with the data). These functions are considered as member functions in C++. It is not possible to access the data directly. If you want to reach the data item in an object, you call a member function in the object. It will read the data item and return the value to you. The data is hidden, so it is considered as safe and far away from accidental alternation. Data and its functions are said to be encapsulated into a single entity

Example: *Elaborating on the person class example from earlier, we might have private data in the class, such as "socialSecurityNumber," that should not be exposed to other objects in the program. By encapsulating this data member as a private variable in the class, outside code would not have direct access to it, and it would remain safe within that person’s object.*

If a method is written in the person class to perform, say, a bank transaction called "bankTransaction()," that function could then access the "socialSecurityNumber" variable as necessary. The person’s private data would be well encapsulated in such a class.

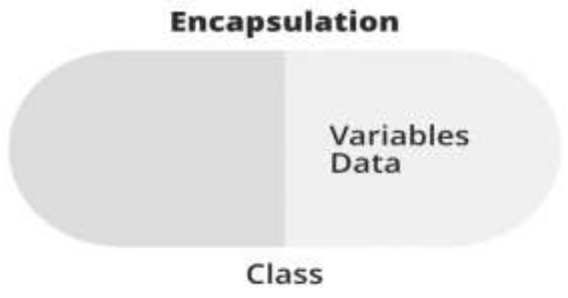


Fig 2: Encapsulation

2. Abstraction

Often, it’s easier to reason and design a program when you can separate the interface of a class from its implementation and focus on the interface. This is akin to treating a system as a “black box,” where it’s not important to understand the gory inner workings in order to reap the benefits of using it.

This process is called “abstraction” in OOP, because we are abstracting away the gory implementation details of a class and only presenting a clean and easy-to-use interface via the class’ member functions. Carefully used, abstraction helps isolate the impact of changes made to the code, so that if something goes wrong, the change will only affect the implementation details of a class and not the outside code.

Abstraction refers to the act of representing essential features without including the background details. To understand this concept more clearly, take an example of ‘switch board’. You only press particular switches as per your requirement. You need not know the internal working of these switches. What is happening inside is hidden from us. This is abstraction, where you only know the essential things to operate on switch board without knowing the background details of switch board

Example: *Think of a stereo system as an object with a complex logic board on the inside. It has buttons on the outside to allow for interaction with the object. When you press any of the buttons, you're not thinking about what happens on the inside because you can't see it. Even though you can't see the logic board completing these functions as a result of pressing a button, it's still performing them.*, albeit hidden to you.

This is the concept of abstraction, which is incredibly useful in all areas of engineering and also applied to great effect in object-oriented programming.

Example: *In OOP, we might have a class defined to represent the human body. One might define some functions as part of its publicly facing interface such as “walk()” or “eatFood().” Calling code could call these functions and remain completely oblivious to the complex inner workings of the human body and its necessary functions to perform the act of walking or eating. These details are completely hidden in the implementation of the walk() and eatFood() body functions and are, therefore, us abstracted away from the end user. In these cases, it’s not important for calling code to understand how the brain coordinates walking or how the stomach manages digesting the food, but rather simply that a human walked or ate.*

3. Inheritance

Object-oriented languages that support classes almost always support the notion of “inheritance.” Classes can be organized into hierarchies, where a class might have one or more parent or child classes. If a class has a parent class, we say it is derived or inherited from the parent class and it represents an “IS-A” type relationship. That is to say, the child class “IS-A” type of the parent class.

Therefore, if a class inherits from another class, it automatically obtains a lot of the same functionality and properties from that class and can be extended to contain separate code and data. A nice feature of inheritance is that it often leads to good code reuse since a parent class’ functions don’t need to be re-defined in any of its child classes.

Consider two classes: one being the superclass—or parent—and the other being the subclass—or child. The child class will inherit the properties of the parent class, possibly modifying or extending its behaviour. Programmers applying the technique of inheritance arrange these classes into what is called an "IS-A" type of relationship.

In simple words it is a process by which object of one class inherit the properties of objects of another class. It is the capability to define a new class in terms of an existing class. An existing class is known as a base class and the new class is known as derived class. Number of examples can be given on this aspect. For example, a motor cycle is a class in itself. It is also a member of two wheelers class. Two wheelers class in turn is a member of automotive class as shown in the automotive is an example of base class and two wheelers is its derived class. In simple words, we can say a motor cycle is a two wheeler automotive. C++ supports such hierarchical classification of classes. The main benefit from inheritance is that we can build a generic base class, and obtain a new class by adding some new features to an existing class and so on. Every new class defined in that way consists of features of both the classes. Inheritance allows existing classes to be adapted to new application without the need for modification.

Example: *For instance, in the animal world, an insect could be represented by an Insect superclass. All insects share similar properties, such as having six legs and an exoskeleton. Subclasses might be defined for grasshoppers and ants. Because they inherit or are derived from the Insect class, they automatically share all insect properties.*

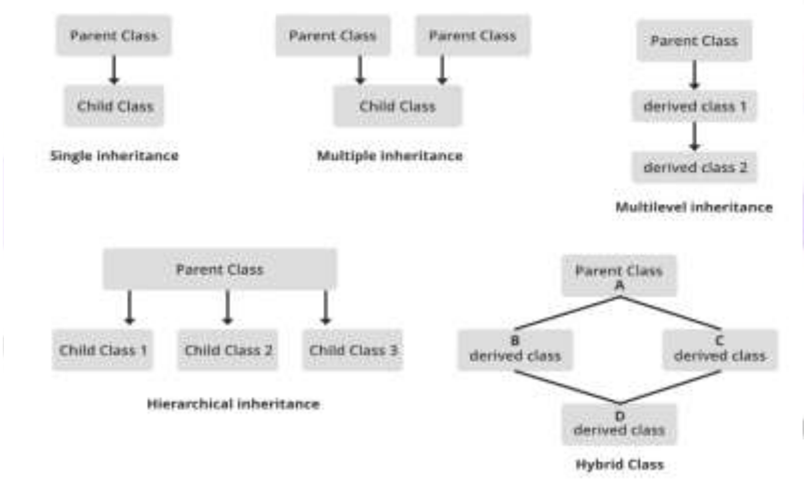


Fig 3: Inheritance

4. Polymorphism

In OOP, polymorphism allows for the uniform treatment of classes in a hierarchy. Therefore, calling code only needs to be written to handle objects from the root of the hierarchy, and any object instantiated by any child class in the hierarchy will be handled in the same way.

Because derived objects share the same interface as their parents, the calling code can call any function in that class' interface. At run-time, the appropriate function will be called depending on the type of object passed leading to possibly different behaviors.

Polymorphism is a key to the power of OOP. It is the concept that supports the capability of data to be processed in more than one form. For example, an operation may exhibit different behavior in different instances. The behavior depends upon the types of data used in the operation. For example let us consider

Example: *Suppose we have a class called, "Animal" and two child classes, "Cat," and "Dog." If the Animal class has a method to make a noise, called, "makeNoise," then, we can override the "makeNoise" function that is inherited by the sub-classes, "Cat" and "Dog," to be "meow" and "bark," respectively. Another function can, then, be written that accepts any Animal object as a parameter and invokes its "makeNoise" member function. The noise will be different: either a "meow" or a "bark" depending on the type of animal object that was actually passed to the function.*

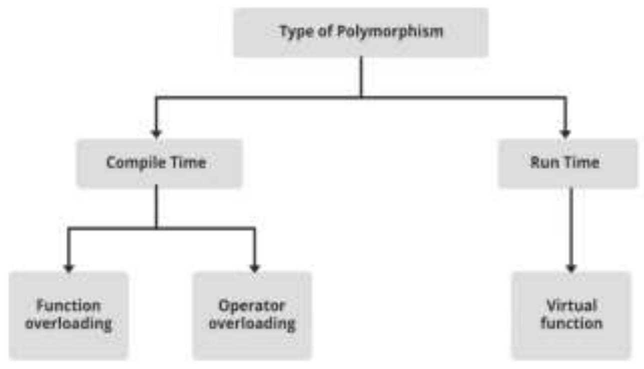


Fig 4: Polymorphism

Benefits of OOP

- OOP models complex things as reproducible, simple structures
- Reusable, OOP objects can be used across programs
- Allows for class-specific behavior through polymorphism
- Easier to debug, classes often contain all applicable information to them
- Secure, protects information through encapsulation

<u>Data Structures & Algorithms</u>

Data Structure is a way of collecting and organising data in such a way that we can perform operations on these data in an effective way. Data Structures is about rendering data elements in terms of some relationship, for better organization and storage. For example, we have some data which has, player's **name** "Virat" and **age** 26. Here "Virat" is of **String** data type and 26 is of **integer** data type.

We can organize this data as a record like **Player** record, which will have both player's name and age in it. Now we can collect and store player's records in a file or database as a data structure. **For example**: "Dhoni" 30, "Gambhir" 31, "Sehwag" 33

As applications are getting complex and data rich, there are three common problems that applications face now-a-days.

- **Data Search** − Consider an inventory of 1 million(10^6) items of a store. If the application is to search an item, it has to search an item in 1 million(10^6) items every time slowing down the search. As data grows, search will become slower.
- **Processor speed** − Processor speed although being very high, falls limited if the data grows to billion records.
- **Multiple requests** − As thousands of users can search data simultaneously on a web server, even the fast server fails while searching the data.

If you are aware of Object-Oriented programming concepts, then a class also does the same thing, it collects different type of data under one single entity. The only difference being data structures provides for techniques to access and manipulate data efficiently.

In simple language, Data Structures are structures programmed to store ordered data, so that various operations can be performed on it easily. It represents the knowledge of data to be organized in memory. It should be designed and implemented in such a way that it reduces the complexity and increases the efficiency.

Need for Data Structure :

- It gives different level of organization data.
- It tells how data can be stored and accessed in its elementary level.
- Provide operation on group of data, such as adding an item, looking up highest priority item.
- Provide a means to manage huge amount of data efficiently.
- Provide fast searching and sorting of data.

Goals of Data Structure:

Data structure basically implements two complementary goals.

Correctness:

Data structure is designed such that it operates correctly for all kinds of input, which is based on the domain of interest. In other words, correctness forms the primary goal of data structure, which always depends on the specific problems that the data structure is intended to solve.

Efficiency:

Data structure also needs to be efficient. It should process the data at high speed without utilizing much of the computer resources such as memory space. In a real time state, the efficiency of a data structure is an important factor that determines the success and failure of the process.

Features of Data Structure:

Some of the important features of data structures are:

Robustness:

Generally, all computer programmers wish to produce software that generates correct output for every possible input provided to it, as well as execute efficiently on all hardware platforms. This kind of robust software must be able to manage both valid and invalid inputs.

Adaptability:

Developing software projects such as word processors, Web browsers and Internet search engine involves large software systems that work or execute correctly and efficiently for many years. Moreover, software evolves due to ever changing market conditions or due to emerging technologies.

Reusability: Reusability and adaptability go hand-in-hand. It is a known fact that the programmer requires many resources for developing any software, which makes it an expensive enterprise. However, if the software is developed in a reusable and adaptable way, then it can be implemented in most of the future applications. Thus, by implementing quality data structures, it is possible to develop reusable software, which tends to be cost effective and time saving.

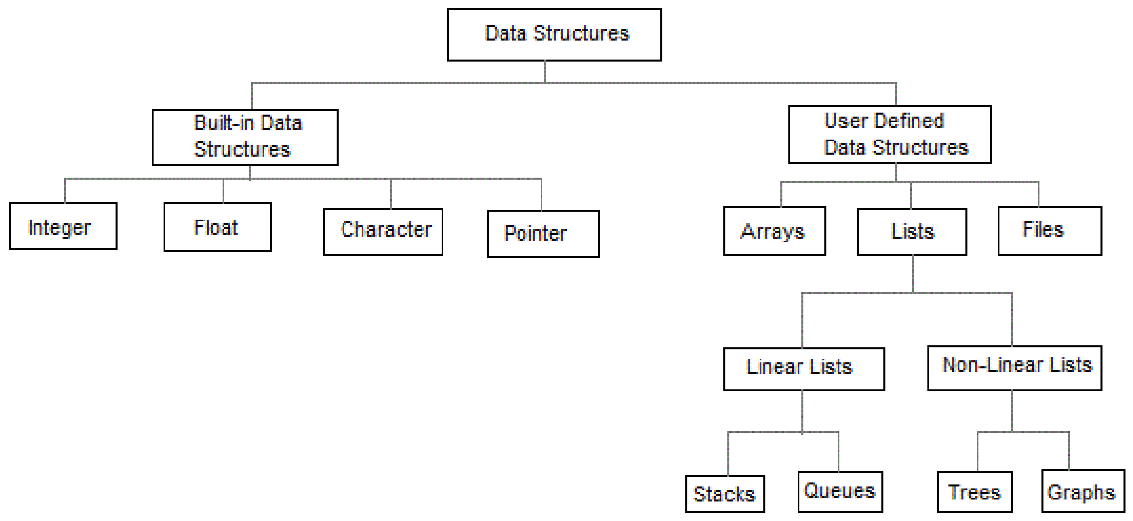
Basic types of Data Structures

As we have discussed above, anything that can store data can be called as a data structure, hence Integer, Float, Boolean, Char etc, all are data structures. They are known as **Primitive Data Structures**.

Then we also have some complex Data Structures, which are used to store large and connected data. Some examples of **Abstract Data Structure** are:

- Linked List
- Tree
- Graph
- Stack, Queue etc.

All these data structures allow us to perform different operations on data. We select these data structures based on which type of operation is required. We will look into these data structures in more details in our later lessons.



INTRODUCTION TO DATA STRUCTURES

The study of data structures helps to understand the basic concepts involved in organizing and storing data as well as the relationship among the data sets. This in turn helps to determine the way information is stored, retrieved and modified in a computer's memory.

Data structure is a branch of computer science. The study of data structure helps you to understand how data is organized and how data flow is managed to increase efficiency of any process or program. Data structure is the structural representation of logical relationship between data elements. This means that a data structure organizes data items based on the relationship between the data elements. Example: A house can be identified by the house name, location, number of floors and so on. These structured set of variables depend on each other to identify the exact house. Similarly, data structure is a structured set of variables that are linked to each other, which forms the basic component of a system

The data structures can also be classified on the basis of the following characteristics:

Characteristic	Description
Linear	In Linear data structures, the data items are arranged in a linear sequence. Example: **Array**
Non-Linear	In Non-Linear data structures, the data items are not in sequence. Example: **Tree**, **Graph**
Homogeneous	In homogeneous data structures, all the elements are of same type. Example: **Array**
Non-Homogeneous	In Non-Homogeneous data structure, the elements may or may not be of the same type. Example: **Structures**
Static	Static data structures are those whose sizes and structures associated memory locations are fixed, at compile time. Example: **Array**
Dynamic	Dynamic structures are those which expands or shrinks depending upon the program need and its execution. Also, their associated memory locations changes. Example: **Linked List created using pointers**

What is an Algorithm?

An algorithm is a finite set of instructions or logic, written in order, to accomplish a certain predefined task. Algorithm is not the complete code or program, it is just the core logic(solution) of a problem, which can be expressed either as an informal high-level description as **pseudocode** or using a **flowchart**. Every Algorithm must satisfy the following properties:

1. **Input**- There should be 0 or more inputs supplied externally to the algorithm.
2. **Output**- There should be at least 1 output obtained.
3. **Definiteness**- Every step of the algorithm should be clear and well defined.
4. **Finiteness**- The algorithm should have finite number of steps.
5. **Correctness**- Every step of the algorithm must generate a correct output.

An algorithm is said to be efficient and fast, if it takes less time to execute and consumes less memory space. The performance of an algorithm is measured on the basis of following properties:

1. Time Complexity
2. Space Complexity

Time Complexity

Time Complexity is a way to represent the amount of time required by the program to run till its completion. It's generally a good practice to try to keep the time required minimum, so that our algorithm completes its execution in the minimum time possible.

Space Complexity

It's the amount of memory space required by the algorithm, during the course of its execution. Space complexity must be taken seriously for multi-user systems and in situations where limited memory is available.

An algorithm generally requires space for following components:

- **Instruction Space:** It's the space required to store the executable version of the program. This space is fixed but varies depending upon the number of lines of code in the program.
- **Data Space:** It's the space required to store all the constants and variables (including temporary variables) value.
- **Environment Space:** It's the space required to store the environment information needed to resume the suspended function.

Applications of Data Structure and Algorithms

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

From the data structure point of view, following are some important categories of algorithms −

- **Search** − Algorithm to search an item in a data structure.
- **Sort** − Algorithm to sort items in a certain order.
- **Insert** − Algorithm to insert item in a data structure.
- **Update** − Algorithm to update an existing item in a data structure.
- **Delete** − Algorithm to delete an existing item from a data structure.