Basics of Data Structure

Aim

To equip the students with the basic skills of using Data Structures in programs

Instructional Objectives

Objectives of this chapter are:

- Describe Data Structures and its types
- Explain items included elementary data organisation
- Summarize the role of algorithms in programming
- Explain the procedure to calculate time and space complexities
- Explain the string processing with its functions
- Demonstrate memory allocation and address variable

Data Structures and its Types

Definition

Data Structures represents the logical and mathematical model of a particular organization of data.

The study of data structures includes the following steps:

(a) The logical (storage representation) and mathematical (operations performed) description of the structure required.
(b) Implementation of the data structure in the program.
(c) Quantitative analysis of the structure which includes the determination of the amount of memory to store the structure and the time required to process the structure.

GOALS OF DATA STRUCTURES

The goals of data structures can be designed to answer certain questions such as:

(a) Does the data structure do what it is supposed to do?

(b) Does the representation work according to the requirement specification of the task?

(c) Is there a proper description of the representation describing how to use it and how it works?

Desired Characteristics of a Data Structure:

Correctness: By correctness, we mean that a data structure is designed, to work correctly for all possible inputs that one might encounter.

Efficiency: Useful data structure and their operations also need to be efficient i.e., they should be fast and not use more of the computer's resources, such as memory space, than required.

Robustness: Every good programmer wants to produce software that is robust, which means that a program produces the correct output for all inputs

Adaptability: Software, needs to be able to evolve over time in response to changing conditions. Software should also be able to adapt to unexpected events. Thus, another important goal of quality software is that it be adaptable.

Reusability: Developing quality software can be expensive, and its cost can be reduced somewhat if the software is designed in a way that make sit easily reusable in future applications. Software reuse can be a significant cost-saving and timesaving technique.

Types of Data Structures

1. Primitive Data Structures:

They are directly operated upon by machine level instructions.

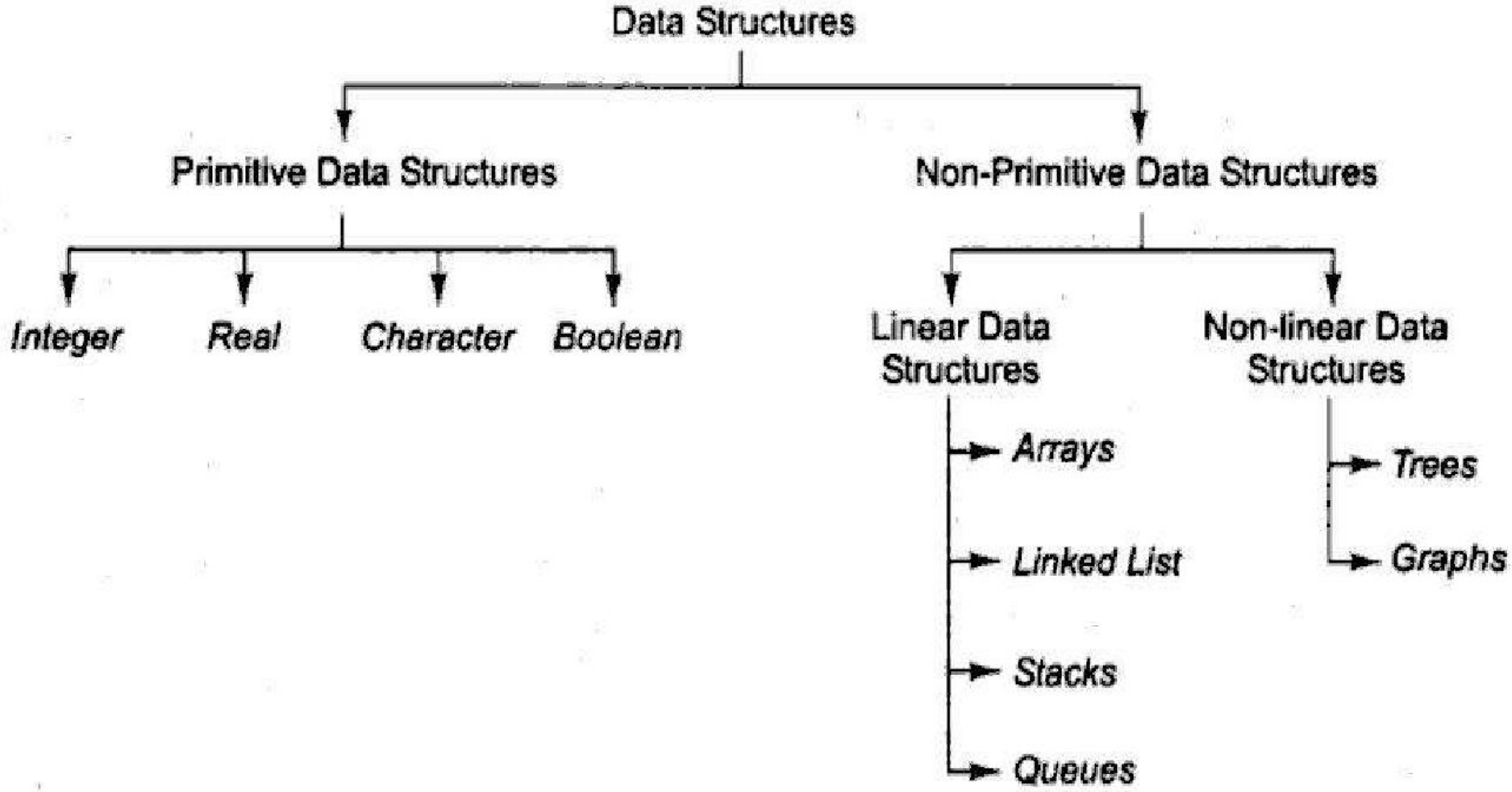
Examples: int, float double etc.

2. Non-primitive data structure:

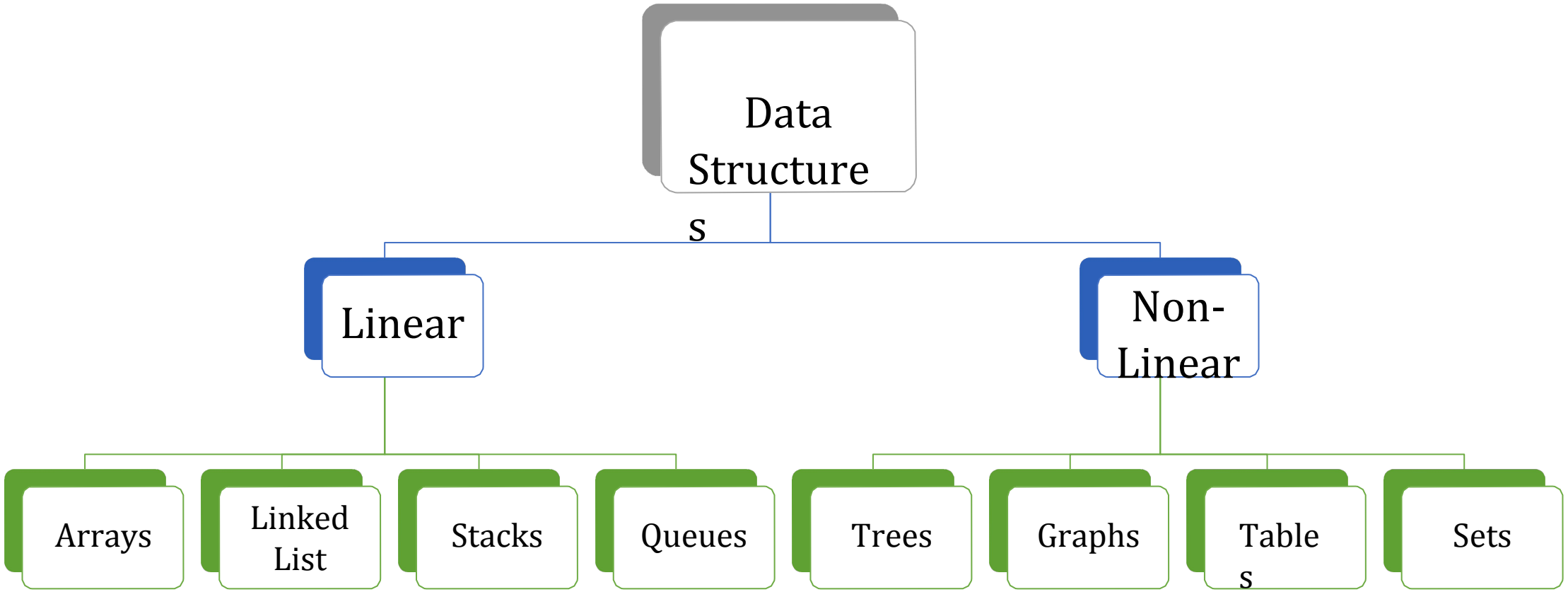
These data types are derived from primary data types

They are used to store group of values.

Examples: Arrays, Structures, Linked Lists, Queues etc.


Data Structures
Primitive Data Structures
Non-Primitive Data Structures
Integer
Real
Character
Boolean
Linear Data Structures
Non-linear Data Structures
Arrays
Linked List
Stacks
Queues
Trees
Graphs

Types of Data Structures



Quiz / Assessment

1) Which of the following data structure is linear type?

a) Graph
b) Trees
c) Binary tree
d) Stack

2) Which of the following data structure is non-linear type?

a) Strings
b) Lists
c) Stacks
d) Graph

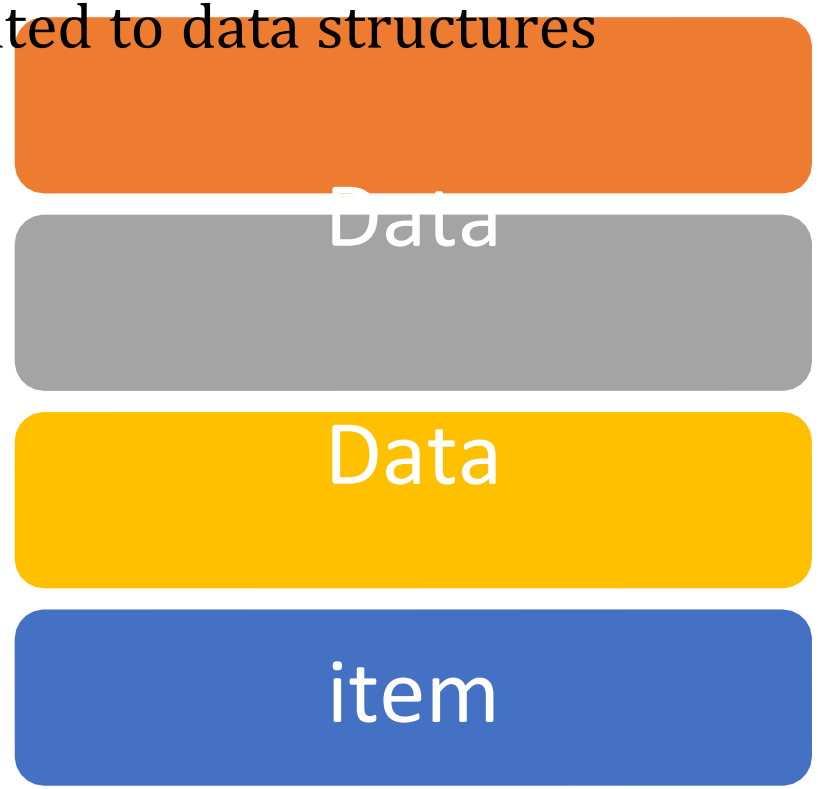
3) Which of the following data structure can't store the non-homogeneous data
elements?

a)Arrays
b) Records
c) Pointers
d) Stacks

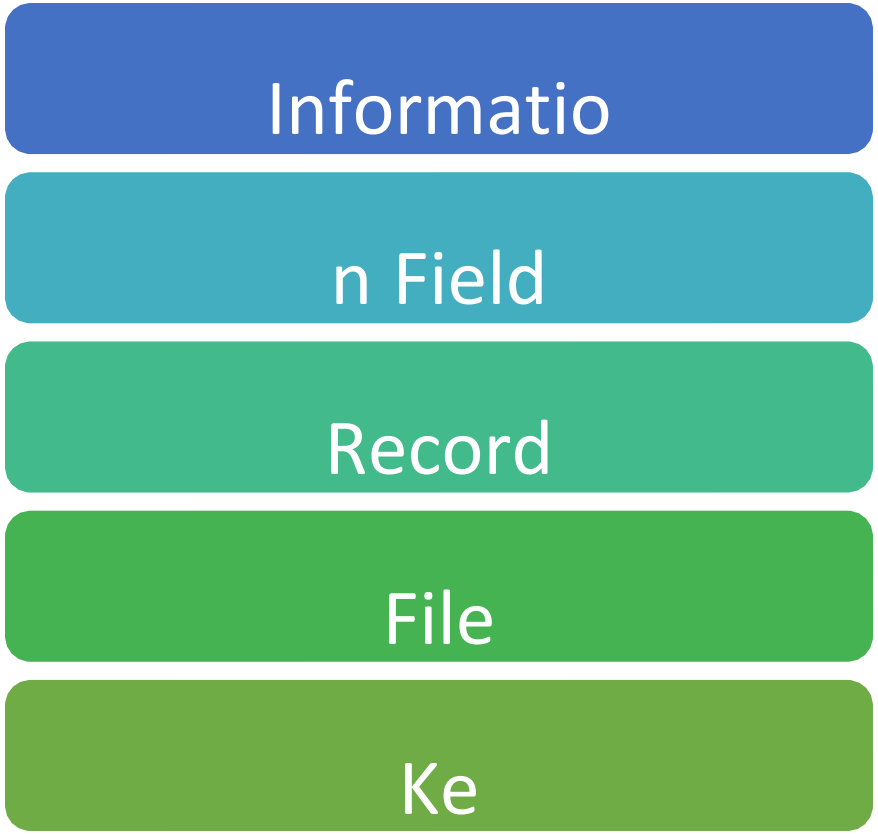
Items Included in Elementary Data Organisation

Items in Elementary Data Organization

Basic Terminologies related to data structures are:



Items in Elementary Data Organization (Contd.)



Items in Elementary Data Organization example

Attributes:	Name	Age	Sex	Social Security Number
Values:	ROHLAND, GAIL	34	F	134-24-5533

Items in Elementary Data Organization example

Suppose an automobile dealership maintains an inventory file where each record contains the following data:

Serial Number, Type, Year, Price, Accessories

The Serial Number field can serve as a primary key for the file, since each automobile has a unique serial number.

Data structure operations

- Data structure operations are the methods used to manipulate the data in a data structure.
- The most common data structure operations are:

1. Traversal
2. Searching
3. Inserting
4. Deleting
5. Sorting
6. Merging

Data structure operations

1. **Traversal:** Traversing the data structure means visiting each element of the data structure in order to perform some specific operation like searching or sorting.
2. **Searching:** The process of finding the location of an element within the data structure is called Searching. Finding the location of a record with a specific key value or finding location of all such records.
3. **Inserting:** Insertion can be defined as the process of adding the elements to the data structure at any location.

Data structure operations

4. Deleting: The process of removing an element from the data structure is called Deletion. We can delete an element from the data structure at any random location.

If we try to delete an element from an empty data structure then **underflow** occurs.

5. Sorting: The process of arranging the data structure in a specific order is known as Sorting. There are many algorithms that can be used to perform sorting,

6. Merging: Combining the records in two different sorted files into a single sorted file.

Example of Data structure operations

An organization contains a membership file in which each record contains the following data for a given member:

Name, Address, Telephone Number, Age, Sex

(a) Suppose the organization wants to announce a meeting through a mailing. Then one would traverse the file to obtain Name and Address for each member.

(b) Suppose one wants to find the names of all members living in a certain area. Again one would traverse the file to obtain the data.

(c) Suppose one wants to obtain Address for a given Name. Then one would search the file for the record containing Name.

(d) Suppose a new person joins the organization. Then one would insert his or her record into the file.

(e) Suppose a member dies. Then one would delete his or her record from the file.

(f) Suppose a member has moved and has a new address and telephone number. Given the name of the member, one would first need to search for the record in the file. Then one would perform the "update"—i.e., change items in the record with the new data.

(g) Suppose one wants to find the number of members 65 or older. Again one would traverse the file, counting such members.

Role of Algorithms in Programming

Algorithms – Role in Programming

An algorithm is a step by step process to solve a problem.

In programming, algorithms are implemented in the form of methods

or functions or routines.

An *algorithm* is a finite set of instructions which, if followed, accomplish a particular task.

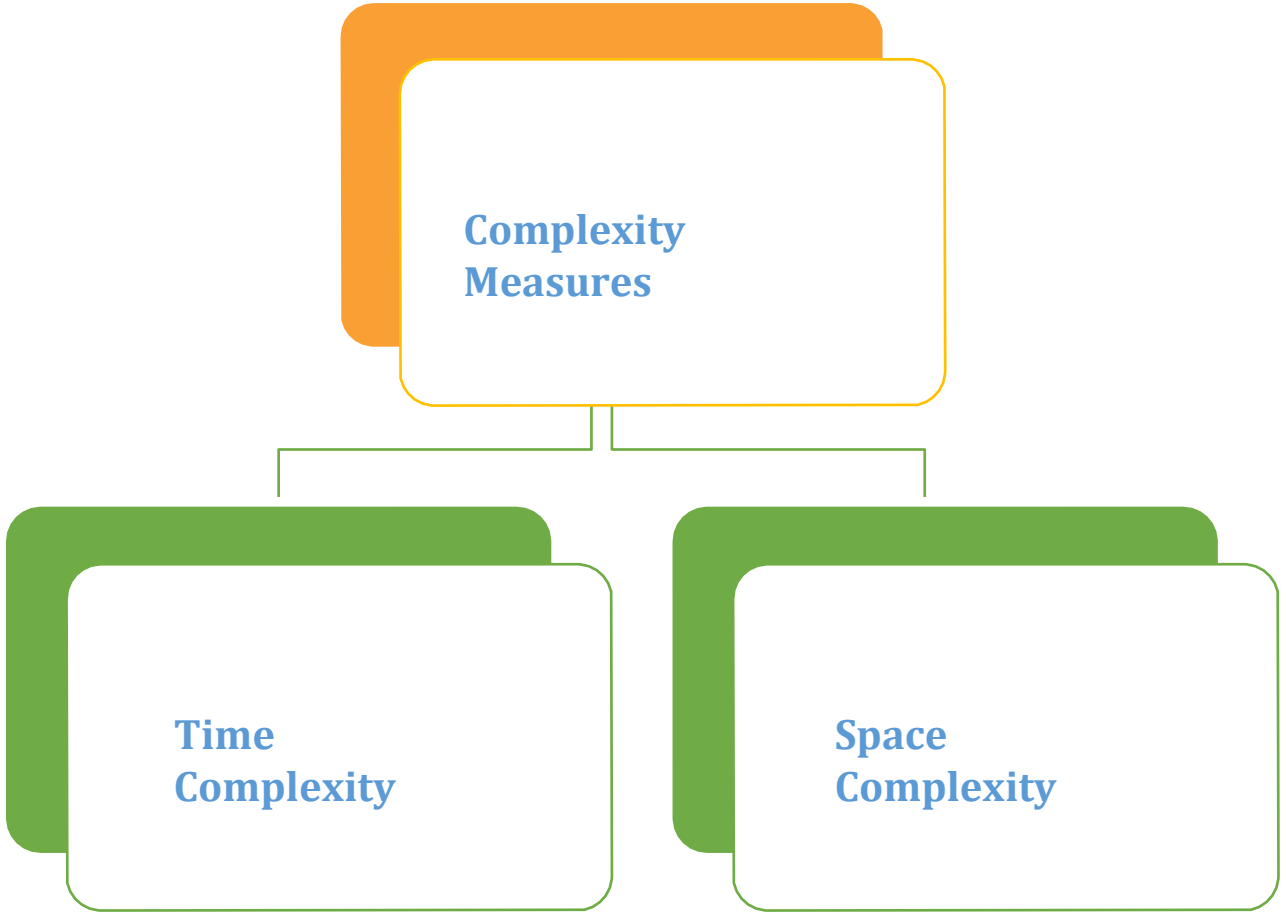
In addition every algorithm must satisfy the following criteria:

Criteria

- (i) *input:* there are zero or more quantities which are externally supplied;
- (ii) *output:* at least one quantity is produced;
- (iii) *definiteness:* each instruction must be clear and unambiguous;
- (iv) *finiteness:* if we trace out the instructions of an algorithm, then for all cases the algorithm will terminate after a finite number of steps;
- (v) *effectiveness:* every instruction must be sufficiently basic that it can in principle be carried out by a person using only pencil and paper. It is not enough that each operation be definite as in (iii), but it must also be feasible.

Procedure to Calculate Time and Space Complexities

Time and Space Complexity



Time Complexity

It describes amount of time an algorithm takes in terms of amount of input to the algorithm.

Expressed in :

- Number of memory access made
- No. of comparisons between two integers
- Number of times a particular loop got executed

Time complexity is not "wall clock" time.

Space Complexity

- It is a function which describes the amount of memory utilized by an algorithm in terms of the

  amount of input to the algorithm.

- We use natural, but fixed-length units, to measure space complexity.

String Processing with its Functions

String Processing

Textual data is represented using arrays of characters called a *string*.

End of string is marked with a null character.

The null or string-terminating character is represented by another character escape sequence, \0.

Eg: char string[] = "Hello, world!";

Commonly used string handling functions are:

- Strlen()
- Strcpy()
- Strcat()
- Strcmp()
- Strlwr()
- Strupr()

More string processing

Function	Description
Strcmpi()	Compares two strings with case insensitivity
Strrev()	Reverses a string
Strlwr ()	Converts an string letters to lowercase
Strupr()	Converts lowercase string letters to uppercase
Strchr()	Finds the first occurrence of a given character in a string
Strrchr()	Finds the last occurrence of a given character in a string
Strset()	Sets all characters in a string to a given character
Strnset()	Sets the specified number of characters in a string to a given character
Strdup()	Used for duplicating a string

Strcmpi()

- The strcmpi() function is a built-in function in C and is defined in the “string.h” header file.
- The strcmpi() function is same as that of the strcmp() function but the only difference is that strcmpi() function is not case sensitive and on the other hand strcmp() function is the case sensitive.
- Syntax:
- int strcmpi (const char * str1, const char * str2);

- **Example**:

```
#include <stdio.h>
#include <string.h>

int main( )
{
  char str1[] = “ABCDE" ;
  char str2[] = “ABCDE" ;
     int j = strcmpi ( str1, str2 ) ;
     printf ( "The function returns = %d",j ) ;
  return 0;
}
```

strrrv()

- The strrev() function is a built-in function in C and is defined in string.h header file. The strrev() function is used to reverse the given string.

- **Syntax:**

- char *strrev(char *str);

- **Example:**

```
#include <stdio.h>
#include <string.h>
 int main()
{
    char str[50] = "123456789";

    printf("The given string is =%s\n", str);

    printf("After reversing string is =%s", strrev(str));
     return 0;
}
```

strlwr()

- The strlwr( ) function is a built-in function in C and is used to convert a given string into lowercase.
- Syntax:
- char *strlwr(char *str);

- **Example**
- #include<stdio.h>
- #include <string.h>
-
- int main()
- {
- char str[] = "CompuTer ScienCe PoRTAl fOr geeKS";
- printf("Given111 string is: %s\n",str);
-
- printf("\nString after converting to the "
- "lowercase is: %s",strlwr(str));
- return 0;
- }

Strupr()

- The strupr( ) function is used to converts a given string to uppercase.
- Syntax:

- char *strupr(char *str);

Example:

```
#include<stdio.h>
#include <string.h>

int main()
{
  char str[] = "CompuTer ScienCe PoRTAl fOr geeKS";

  printf("Given string is: %s\n", str);

  printf("\nstring after converting to the uppercase is: %s", strupr(str));
  return 0;
}
```

Strchr()

- The C library function char *strchr(const char *str, int c) searches for the first occurrence of the character c (an unsigned char) in the string pointed to by the argument str.

- Declaration
- char *strchr(const char *str, int c)

```
#include <stdio.h>
#include <string.h>
int main ()
{
   const char str[] = "Use strchr() function in C.";
   const char ch = 's'; // it is searched in str[]
   char *ptr; // declare character pointer ptr
        printf (" Original string is: %s \n", str);
   // use strchr() function and pass str in which ch is to be searched
   ptr = strchr( str, ch);
   printf (" The first occurrence of the '%c' in '%s' string  is: '%s' ", ch, str, ptr);
   return 0;
}
```

Strrchr()

- The C library function char *strrchr(const char *str, int c) searches for the last occurrence of the character c (an unsigned char) in the string pointed to, by the argument str.

- Declaration
- Following is the declaration for strrchr() function.
- char *strrchr(const char *str, int c)

- **Example:**
- #include <stdio.h>
- #include <string.h>
- int main ()
- {
- int len;
- const char str[] = “MyString";
- const char ch = ‘S';
- char *ret;
- ret = strrchr(str, ch);
- printf("String after |%c| is - |%s|\n", ch, ret);
- return(0);
- }

Strnset()

- The strnset() function is a built-in function in C and it sets the first n characters of a string to a given character. If n is greater than the length of string, the length of string is used in place of n.
- Syntax:
- char *strnset(const char *str, char ch, int n);

- **Example:**

```
#include <stdio.h>
#include <string.h>
 int main()
{
   char str[] = “MyStringFunctions";

   printf("Original String: %s\n", str);

   // First 5 character of string str
   // replaced by character '*'
   printf("Modified String: %s\n", strnset(str, '*', 5));
        return 0;
}
```

Strdup()

- The **strdup()** and functions is used to duplicate a string.
  strdup() :
- Syntax : *char *strdup(const char *s);*
- This function returns a pointer to a null-terminated byte string, which is a duplicate of the string pointed to by ***s***. The memory obtained is done dynamically using malloc and hence it can be freed using free.

- **Example:**

```
#include<stdio.h>
#include<string.h>
 int main()
{
    char source[] = “MyString";
     // 5 bytes of source are copied to a new memory
    // allocated dynamically and pointer to copied
    // memory is returned.
    char* target = strndup(source, 5);
     printf("%s", target);
    return 0;
}
```

Quiz / Assessment

10)Which of the following function compares 2 strings with case-insensitively?

a)Strcmpi(s, t)

b)strcmpcase(s, t)

c) Strcasecmp(s, t)

d) strchr(s, t)

11)How will you print \n on the screen?

a)printf("\n");

b) printf('\n');";

c)echo "\\n;

d)printf("\\n");

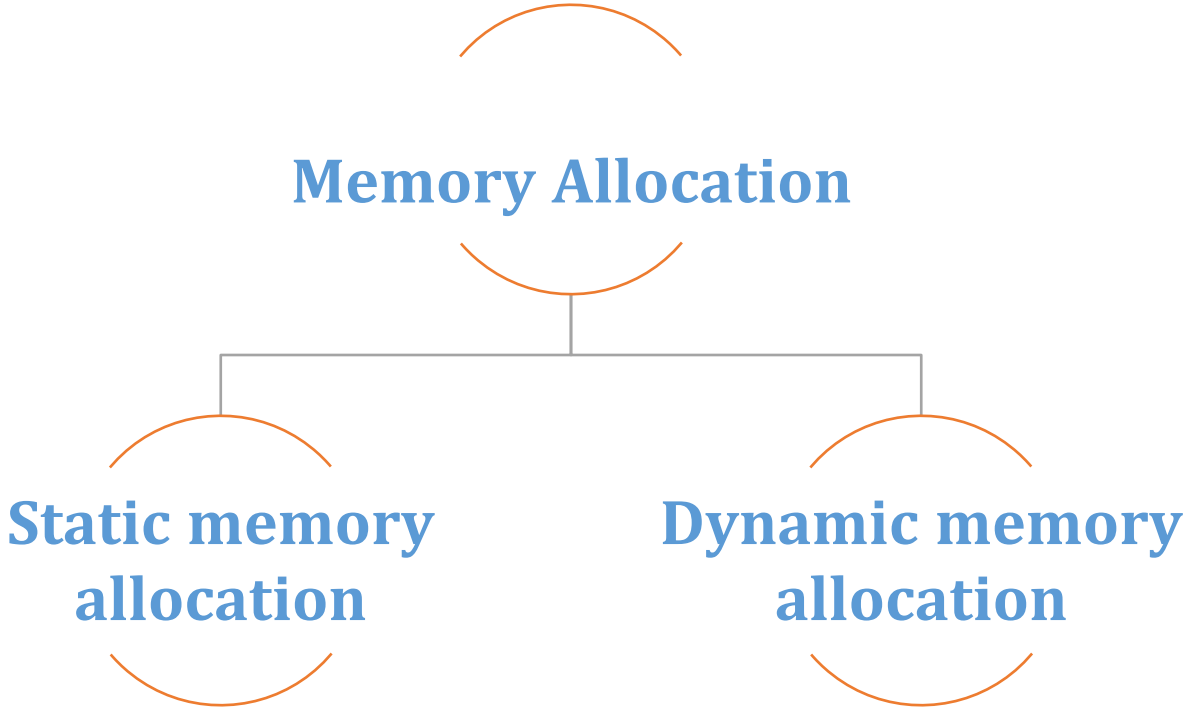
Memory Allocation and Address Variable

Memory Allocation: Memory allocation is a process by which computer programs and services are assigned with physical or virtual memory space.

The memory allocation is done either before or at the time of program execution. There are two types of memory allocations:

Compile-time or Static Memory Allocation
Run-time or Dynamic Memory Allocation

Memory Allocation



Static Memory Allocation:

When memory for the program is allocated during compile time, it is called static memory allocation.

The compiler allocates the required memory for the program before the execution of the program.

Since memory allocation takes place during compile time, It is also called compile-time memory allocation.

No special functions are required for static allocation, just normally declare variables. Memory is allocated From Stack Memory.

Example

```
#include <stdio.h>
int main()
{
   int a;
   int b[10];
   return 0;
}
```

Here memory allocation is done during compile time and is Static Memory Allocation

Static Memory Allocation

- Compiler allocates required memory space
- Every variable declared in program are allocated unique memory addresses
- The variables may be assigned with a pointer variable
- Memory should be reserved in advance.
- Not possible to allocate memory during Runtime

Dynamic Memory Allocation

- Memory need not be Reserved in advance

- Executing program request processor for a memory block during its run time
- It enables us to create data types and structures of any size and length to suit our programs need within the program.

- Doesn't waste memory space.

Dynamic Memory Allocation

When memory for the program is allocated during execution time, it is called Dynamic Memory Allocation.

The compiler allocates the required memory for the program during the execution of the program.

Since memory allocation takes place during run time or execution time, It is also called run-time memory allocation.

Memory is allocated dynamically using malloc() and calloc() keywords.

Memory allocated dynamically must be released using the free() keyword. Here we created two integer arrays dynamically.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int *a=(int*)calloc(10*sizeof(int),0);
   int *b=(int*)malloc(10*sizeof(int));

   free(a);
   free(b);
   return 0;
}
```

S.No	Static Memory Allocation	Dynamic Memory Allocation
1	Memory Allocation is made during the Compilation stage.	Memory Allocation is done during the Execution stage.
2	Memory of the variables are allocated permanently until the program or function call completes.	The variables' memory is allocated only when required and called by the calloc()/malloc() function.
3	Allocation is done from the Stack Memory.	Allocation is done from the Heap Memory.
4	In Static memory, if the memory is allocated for a program, the memory size cannot be changed.	In Dynamic memory, if the memory is allocated for a program, the memory size can be changed later.
5	Less Efficient Memory Management.	More Efficient memory management..

6	Memory allocated cannot be reused.	Allocated memory can be released and used again if not required anymore.
7	Execution of the program is faster than when the memory is allocated dynamically.	Execution of the program is slower than when the memory is allocated statically..
8	Can be considered simple compared to dynamic memory allocation.	It a more complex when it comes to declaring multi-dimensional arrays.
9	Memory declared stays from the beginning to the end of the program execution.	Memory declared can be freed ans reused, and other memory can be allocated.
10	Used for arrays	Used for Linked-Lists

Accessing the address of a variable

- Any variable address can be obtained by using an ampersand sign (&) in front of a variable name.

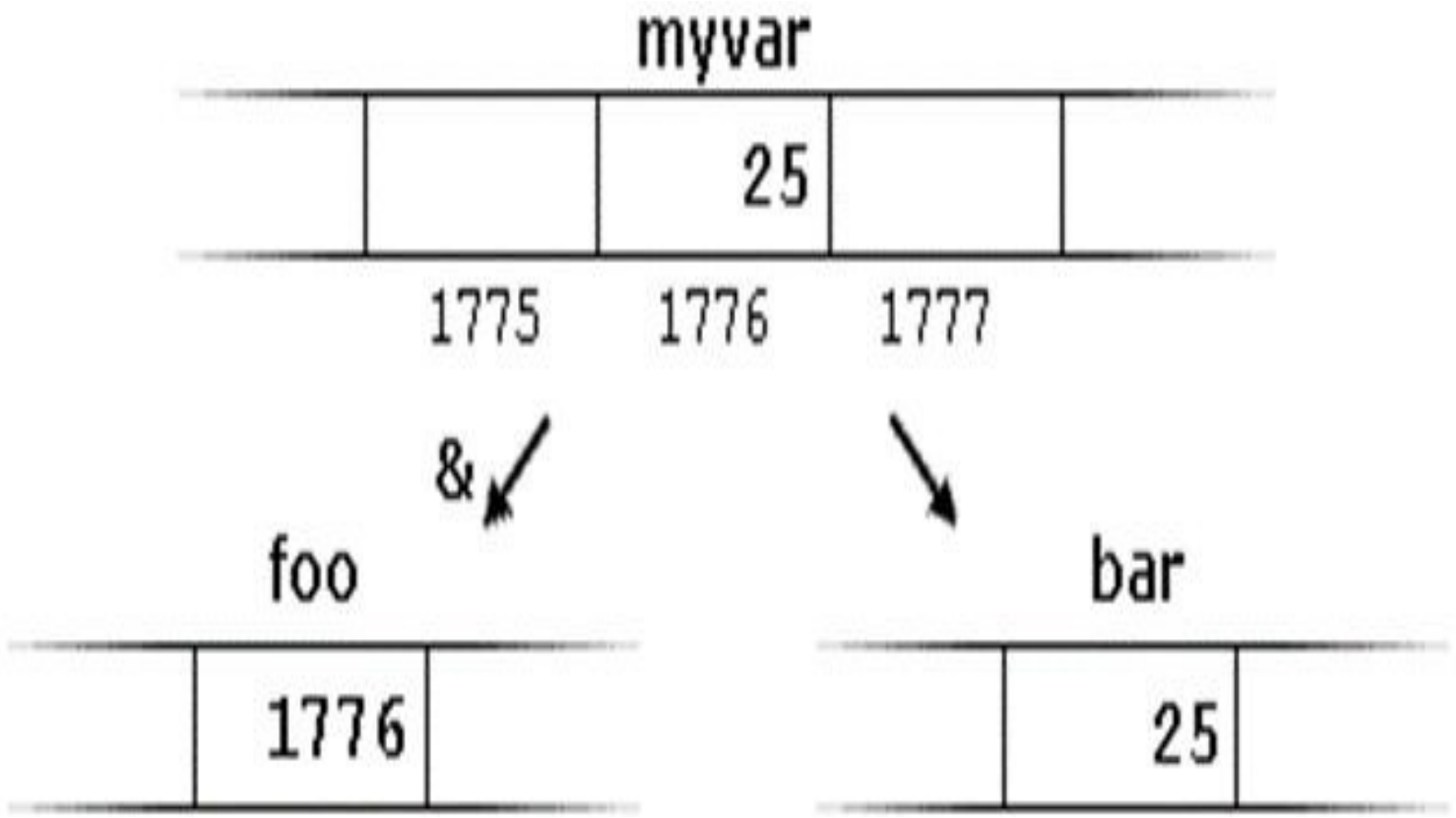
- "& "sign is known as **address-of operator**.

- **Example**

  - fun= &myvar;

- The variable that stores the address of another variable (is what in C is called a pointer.

Accessing the address of a variable (Example)



Quiz / Assessment

16) Choose the right answer. Prior to using a pointer variable

a) It should be declared b) It should be initialized

c) It should be declared and initialized d) It should be neither declared nor initialized

17) The address operator &, cannot act on and

18) The operator > and < are meaningful when used with pointers, if

a) The pointers point to data of similar type

b) The pointers point to structure of similar data type.

c) The pointers point to elements of the same array

d) The pointers point to elements of the another array

Activity

Brief description of

Offline Activity
(15 min)

- Divide the students into two groups.
- Give selection sort algorithm and insertion sort algorithm to each of the groups.
- Students should calculate the time complexities for both these algorithms.

Summary

- ✔ 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 are categorized into two types: linear and nonlinear. Linear data structures are the ones in which elements
  are arranged in a sequence, nonlinear data structures are the ones in which elements are not arranged sequentially
- ✔ The complexity of an algorithm is a function describing the efficiency of the algorithm in terms of the amount of data the algorithm must process.
- ✔ The basic terminologies in the concept of data structures are Data, Data item, Entity, Entity set, Information, Record, file, key etc.
- ✔ String functions like strcmp, strcat, strlen are used for string processing in C
- ✔ Static and Dynamic memory allocation are the core types of memory allocation
- ✔ The address of a variable can be obtained by preceding the name of a variable with the address-of operator (&)

e-References

- cs.utexas.edu, (2016). Complexity Analysis. Retrieved on 19 April 2016, from https://www.cs.utexas.edu/users/djimenez/utsa/cs1723/lecture2.html

- compsci.hunter.cuny.edu, (2016). C Strings and Pointers. Retrieved on 19 April 2016, from http://www.compsci.hunter.cuny.edu/~sweiss/resources/cstrings.pdf

- Msdn.microsoft.com, (2016). An extensive examination of Data Structures. Retrieved on 19 April 2016, from https://msdn.microsoft.com/en-us/library/aa289148(v=vs.71).aspx

External Resources

1. Kruse, R. (2006). *Data Structures and program designing using 'C'* (2nd ed.). Pearson Education.
2. Srivastava, S. K., & Srivastava, D. (2004). *Data Structures Through C in Depth* (2nd ed.). BPB Publications.
3. Weiss, M. A. (2001). *Data Structures and Algorithm Analysis in C* (2nd ed.). Pearson Education.

Thank you