**Day:1**

* **Source Code vs Object Code:**

**Source code:** Input to a compiler or other code translator. Source code is normally generated by humans

**Object code:** Object code is code generated by a compiler

**Byte Code:** Byte code (in Java, Python and etc.) is an intermediate code between source code and machine code that is executed by an interpreter such as JVM. e.g., Java class file.

* **Using Compiler with flags:**

Debug mode- Create Breakpoints and check different variables and their contents while compilation and execution of program

* **Keywords/Variables:**

Keywords: Reserved words like if,else,goto,for,while,etc..

Variables: containers which are used to store the data.

DataTypes: int,float,char,double,etc…

|  |
| --- |
|  |
|  |  |
|  |  |

* **Variable Scope,lifetime,initial value,modifiers:**

Variables scope: place Were variables can be accessed is know as variable Scope.

Method level,class level,package level.

|  |
| --- |
|  |
|  | - Lifetime of a variable is said to be the time period in which the variable is residing in the program memory. |
|  | - If we do not initialize a variable with some value then default value is assigned to it by the compiler. |
|  |  |

* **Programming constructs:**

Selection constructs:switch,if,if else

Loop constructs:for,while,do while

* **Declare Array:**

Array ->Store elements of same data type

Declaration-> int array\_name[] = new int[size];

* **Store data in array and process elements:**

Data can be stored and accessed using index.

|  |
| --- |
|  |
|  |  |

**Day:2**

* **Pointers,declare and initialize pointers:**

Pointer variable store address of another variable.

Java- no pointers->for security

|  |
| --- |
|  |
|  |  |

* **Function definitions/invocations:**

Functions are a block of code which performs a specific task and can be invoked from anywhere within the scope.

* **Call function by passing values and addresses:**

i) Call by Value means calling a method with a parameter as value. Through this, the argument value is passed to the parameter.

ii) While Call by Reference means calling a method with a parameter as a reference. Through this, the argument reference is passed to the parameter.

* **Dynamic memory allocation/deallocation:**

When memory is allocated at the runtime of a program then it is termed as the dynamic memory allocation , once the variable or objects goes out of scope then it is deleted from the memory and the memory held by it is freed , this process is called as memory deallocation.In java memory deallocation is done by Garbage Collector.

Tools:jvisualvm,jconsole

* **Memory leakage:**

Memory leak is a scenario that occurs when **objects are no longer being used by the application, but the Garbage Collector is unable to remove them from working memory** – because they’re still being referenced.

* **User defined structures:**

i) A structure is a heterogeneous container object, i.e., it is an object with elements whose values do not have to be of the same data type. ... A user-defined data type is a structure with a fixed set of fields defined by the user.

ii) Class, Structure , Union are User Defined Structures/Datatypes.

* **Access data members:**

Data Members can be accessed using objects/instances of the class or it’s a standred that make variables in class private and access them using getters and setter methods of the class.

* **Array of pointers:**

an array of pointers is an indexed set of variables, where the variables are pointers

**Day:3**

* **Implement Data structures:**

|  |
| --- |
|  |
|  | 1.Array |
|  | ->Sequential Access |
|  | ->stored at contiguous memory locations |
|  | ->Static Allocation |
|  | ->Use for limited data |
|  | ->Array contains elements of same data type |
|  |  |
|  |  |
|  | 2.Linked List |
|  | ->Sequential Access |
|  | ->Dynamic Memory Allocation |
|  | ->A linked list is a linear data structure |
|  |  |
|  |  |
|  | 3.Stack |
|  | ->Linear Data Structure |
|  | ->stack follows the LIFO (Last In First Out) principle |
|  | ->Dynamic Memory Allocation |
|  | ->The stack can contain elements of the different data types. |
|  | ->Stack has only one end open for pushing and popping the data |
|  |  |
|  |  |
|  | 4.Queue |
|  | ->Queues are based on the FIFO principle, |
|  | ->Dynamic Memory Allocation |
|  | ->The queue can contain elements of the different data types. |
|  | -> Queue has both ends open for enqueuing and dequeuing the data |

* **Linked list:**

|  |
| --- |
| //Linked List operations: |
|  | 1.Add element at Last : |
|  | Steps:1.Create New Node |
|  | 2.Add data to new node |
|  | 3.newnode->next=NULL; |
|  | 4.If linked list is empty make newnode as head |
|  | 5.else Traverse till last node and change next of lastnode |
|  | lastnode->next=newnode; |
|  |  |
|  | 2.Add element at front |
|  | steps:1.Create New Node |
|  | 2.Add data to new node |
|  | 3.Make next of newnode as head |
|  | newnode->next=head; |
|  | 4.move head to point to newnode |
|  | head->newnode |
|  |  |
|  | 3.Add element Between two nodes |
|  | Steps:1.Create New Node |
|  | 2.Add data to new node |
|  | 3.newnode->next=prevnode->next; |
|  | 4.prevnode->next=newnode |
|  |  |
|  | 4.Remove element |
|  | Steps:1.Find prev node of node to be deleted |
|  | 2.change next of prevnode |
|  | 3.free memory for node to be deleted |

* **Stack:**

**//Stack Operations:**

Push

Pop

Peek

Isempty

Isfull

Print

Top

* **Queue:**

|  |
| --- |
| //Queue Implementation |
|  | //Front and Rear are two pointers |
|  | 1.Enqueue |
|  | Algorithm:1.Check queue is full or not |
|  | 2.If queue is full show queue is full |
|  | 3.Else Icrement rear to point to next location |
|  | 4.add new data where rare is pointing |
|  |  |
|  |  |
|  | 2.Dequeue |
|  | Algorithm:1.Check queue is empty or not |
|  | 2.If empty show queue empty message |
|  | 3.Else access data where front is pointing |
|  | 4.Increment front to point to next location |

* **Binary Tree:**

|  |
| --- |
| 1.Hierarchical data structure |
|  | 2.Dynamic memory allocation |
|  | 3.Sequential Access |
|  | 4.Traversing start from root node |
|  |  |
|  | Main applications of trees include: |
|  | 1. Manipulate hierarchical data. |
|  | 2. Make information easy to search |
|  | 3. Manipulate sorted lists of data. |
|  |  |
|  |  |
|  | A Tree node contains following parts. |
|  | 1. Data |
|  | 2. Pointer to left child |
|  | 3. Pointer to right child |
|  |  |
|  | * **Types:** |
|  | 1.Full Binary Tree :A Binary Tree is a full binary tree if every node has 0 or 2 children |
|  | 2.Complete Binary Tree: A Binary Tree is a complete Binary Tree if all the levels are completely filled except possibly the last level |
|  | 3.Perfect Binary Tree :A Binary tree is a Perfect Binary Tree in which all the internal nodes have two children and all leaf nodes are at the same level. |
|  | 4.Balanced Binary Tree: A binary tree is balanced if the height of the tree is O(Log n) where n is the number of nodes. |
|  | 5.A degenerate (or pathological) tree: A Tree where every internal node has one child. Such trees are performance-wise same as linked list. |