Module-2: General Principles- Concepts in discrete event simulation: event scheduling/time advances algorithms, world views. List Processing: properties and operations, data structures and dynamic allocation, techniques

In discrete systems, the changes in the system state are discontinuous and each change in the state of the system is called an **event**. The model used in a discrete system simulation has a set of numbers to represent the state of the system, called as a **state descriptor.** Following is the graphical representation of the behavior of a discrete system simulation.

The basic building blocks of all discrete-event simulation models: entities and attributes, activities and events.
 A system is modeled in terms of
o Its state at each point in time
o The entities that pass through the system and the entities that represent system resources
o The activities and events that cause system state to change.

Discrete-event models are appropriate for those systems for which changes in system state occur only at discrete points in time.

Discrete Event Simulation – Key Features

Discrete event simulation is generally carried out by software designed in high level programming languages such as Pascal, C++, or any specialized simulation language. Following are the five key features −

- **Entities** − these are the representation of real elements like the parts of machines.
- **Relationships** − It means to link entities together.
- **Simulation Executive** − It is responsible for controlling the advance time and executing discrete events.
- **Random Number Generator** − It helps to simulate different data coming into the simulation model.
- **Results & Statistics** − It validates the model and provides its performance measures.

Time Graph Representation

Every system depends on a time parameter. In a graphical representation it is referred to as clock time or time counter and initially it is set to zero. Time is updated based on the following two factors −

- **Time Slicing** − It is the time defined by a model for each event until the absence of any event.
- **Next Event** − It is the event defined by the model for the next event to be executed instead of a time interval. It is more efficient than Time Slicing.

Components of discrete event Simulation

1. System: A collection of entities (e.g., people and machines) that together over time to accomplish one or more goals.

2. Model: An abstract representation of a system, usually containing structural, logical, or mathematical relationships which describe a system in terms of state, entities and their attributes, sets, processes, events, activities, and delays.

3. System state: A collection of variables that contain all the information necessary to describe the system at any time.

4. Entity: Any object or component in the system which requires explicit representation in the model (e.g., a server, a customer, a machine).

5. Attributes: The properties of a given entity (e.g., the priority of a v customer, the routing of a job through a job shop).

6. List: A collection of (permanently or temporarily) associated entities ordered in some logical fashion (such as all customers currently in a waiting line, ordered by first come, first served, or by priority).

7. Event: An instantaneous occurrence that changes the state of a system as an arrival of a new customer).

8. Event notice: A record of an event to occur at the current or some future time, along with any associated data necessary to execute the event; at a minimum, the record includes the event type and the event time.

9. Event list: A list of event notices for future events, ordered by time of occurrence; also known as the future event list (FEL).

10. Activity: A duration of time of specified length (e.g., a service time or arrival time), which is known when it begins (although it may be defined in terms of a statistical distribution).

11. Delay: A duration of time of unspecified indefinite length, which is not known until it ends (e.g., a customer's delay in a last-in, first-out waiting line which, when it begins, depends on future arrivals).

12. Clock: A variable representing simulated time.

The Event-Scheduling/Time-Advance Algorithm

The mechanism for advancing simulation time and guaranteeing that all events occur in correct chronological order is based on the future event list (FEL). This list contains all event notices for events that have been scheduled to occur at a future time.

- ✓ **Future Event List (FEL)**
  - o **To contain all event notices for events that have been scheduled to occur at a future time.**
  - o **To be ordered by event time,** meaning that the events are arranged chronologically; that is, the event times satisfy.
  - o **Scheduling a future event** means that at the instant an activity begins, its duration is computed or drawn as a sample from a statistical distribution and the end-activity event, together with its event time, is placed on the future event list.

The sequence of actions which a simulator must perform to advance the clock system snapshot is called the event-scheduling/time-advance algorithm

The system snapshot at time t=0 and t=t1

CLOCK	System State		Future Event List	
T	(5,1,6)		(3, t1)— Type 3 event to occur at timet1 (1, t2)—Type 1 event to occur at time t2 (1, t3)- Type 1 event to occur at time t3 (2, tn)—Type 2 event to occur at time tn	

Event-scheduling/time-advance algorithm

Step 1. Remove the event notice for the imminent event
(event 3, time t\) from PEL

Step 2. Advance CLOCK to imminent event time
(i.e., advance CLOCK from r to t1).

Step 3. Execute imminent event: update system state,
change entity attributes, and set membership as needed.

Step 4. Generate future events (if necessary) and
place their event notices on PEL ranked by event time.
(Example: Event 4 to occur at time t*, where t2 < t* < t3.)

Step 5. Update cumulative statistics and counters.

New system snapshot at time *t1*

CLOCK	System State		Future Event List	
T1	(5,1,5)		(1, t2)—Type 1 event to occur at time t1 (4, t*)— Type 4 event to occur at time t* (1, t3)—Type 1 event to occur at time t3 (2, tn)—Type 2 event to occur at time tn	

World Views

- During simulation package or manual simulation, a modeler adopts a **world view or orientation for developing a model**.
- Those most prevalent are **the event scheduling world view, the process-interaction worldview, and the activity-scanning world view.**

1. **The process-interaction approach**, a simulation analyst thinks in terms of processes.
   - The **process-interaction approach** is popular because of its intuitive appeal, and because the simulation packages that implement it allow an analyst to describe the process flow in terms of high-level block or network constructs.

2 When using the event-scheduling approach, a simulation analyst concentrates on events and their effect on system state. Both the event-scheduling and the process-interaction approaches use a variable time advance.

3 **The activity-scanning approach** uses a fixed time increment and a rule-based approach to decide whether any activities can begin at each point in simulated time.

The pure activity scanning approach has been modified by what is called the **three-phase approach.** In the three-phase approach, events are considered to be activity duration-zero time units. With this definition, activities are divided into two categories called B and C.

- ✓ **B activities:** Activities bound to occur; all primary events and unconditional activities.
- ✓ **C activities:** Activities or events those are conditional upon certain conditions being true.

With the three-phase approach the simulation proceeds with repeated execution of the three phases until it is completed:

1. **Phase A:** Remove the imminent event from the FEL and advance the clock to its event time. Remove any other events from the FEL that have the event time.
2. **Phase B:** Execute all B-type events that were removed from the FEL.
3. **Phase C:** Scan the conditions that trigger each C-type activity and activate any whose conditions are met. Rescan until no additional C-type activities can begin or events occur.

Manual Simulation Using Event Scheduling

In an event-scheduling simulation, a simulation table is used to record the successive system snapshots as time advances. Let us consider the example of a grocery shop which has only one checkout counter. **(Single-Channel Queue).**The system consists of those customers in the waiting line plus the one (if any) checking out. The model has the following components:

- ✓ **System state** (LQ (t), LS (t)), where LQ (t) is the number of customers in the waiting line, and LS (t) is the number being served (0 or 1) at time t.
- ✓ **Entities:** The server and customers are not explicitly modeled, except in terms of the state variables above.
- ✓ **Events :** Arrival(A) & Departure(D)
  Stopping event (E), scheduled to occur at time 60.
- ✓ **Event notices**
  (A, i). Representing an arrival event to occur at future time t
  (D, t), representing a customer departure at future time t
  (£, 60), representing the simulation-stop event at future time 60
- ✓ **Activities:** Inter arrival time, Service time,
- ✓ **Delay** Customer time spent in waiting line.

In this model, the FEL will always contain either two or three event notices.

List Processing

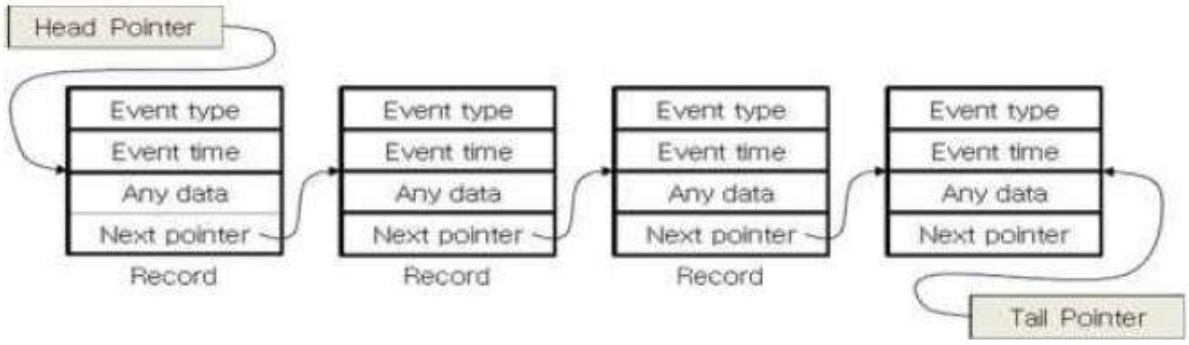
List processing deals with methods for handling lists of entities and the future event list

Basic properties and Operations

1. They have a head pointer/ top pointer
2. Some even have tail pointer

Operations

1. Removing a record from the top of the list
2. Removing a record from any location on the list
3. Adding an entity record to the top or bottom of the list
4. Adding a record to an arbitrary position on the list, determined by the ranking rule.



1. Removing a record from any location on the list.

- If an arbitrary event is being canceled, or an entity is removed from a list based on some of its attributes (say, for example, its priority and due date) to begin an activity.
- By making a partial search through the list.

2. Adding an entity record to the top or bottom of the list.

- When an entity joins the back of a first-in first-out queue.
- by adjusting the tail pointer on the FEL by adding an entity to the bottom of the FEL

3. Adding a record to an arbitrary position on the list, determined by the ranking rule.

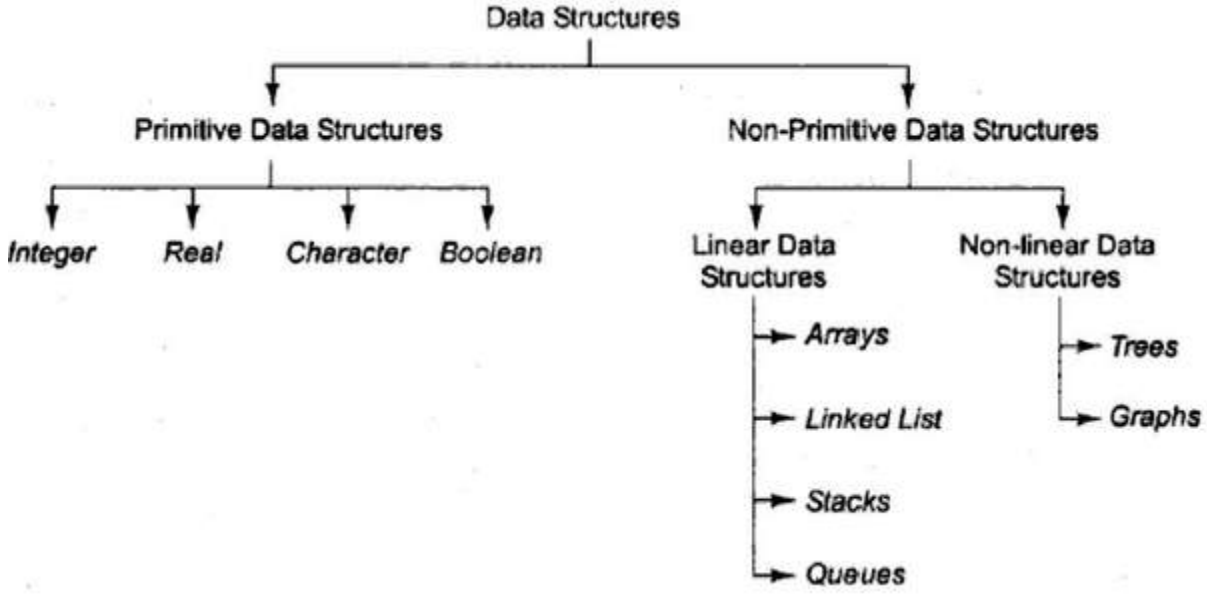
- If a queue has a ranking rule of earliest due date first (EDF).
- By making a partial search through the list.

The goal of list-processing techniques: to make second and fourth operations efficient

- ✓ **The notation R(i) :** the ith record in the array
- ✓ **Advantage:** Any specified record, say the ith, can be retrieved quickly without searching, merely by referencing R (i).
- ✓ **Disadvantage:** When items are added to the middle of a list or the list must be rearranged.
- o Arrays typically have a fixed size, determined at compile time or upon initial allocation when a program first begins to execute.
- o In simulation, the maximum number of records for any list may be difficult or impossible to determine ahead of time, while the current number in a list may vary widely over the course of the simulation run.

Data structures and dynamic allocation and Technique

Data structures are used to store data in a computer in an organized fashion. Different types of data structures are:-



Lists: A group of similar items with connectivity to the previous or/and next data items.
Arrays: A set of homogeneous values
Records: A set of fields, where each field consists of data belongs to one data type.
Trees: A data structure where the data is organized in a hierarchical structure. This type of data structure follows the sorted order of insertion, deletion and modification of data items.
Tables: Data is persisted in the form of rows and columns. These are similar to

records, where the result or manipulation of data is reflected for the whole table.

Stack- Works in first in last out order. The element inserted first in stack is removed last.

Queue- First in First out order. The element inserted first is removed first.

Linked list- Stored data in a linear fashion.

Trees- Stores data in a non linear fashion with one root node and sub nodes.

Simulation of Data structures and Algorithms

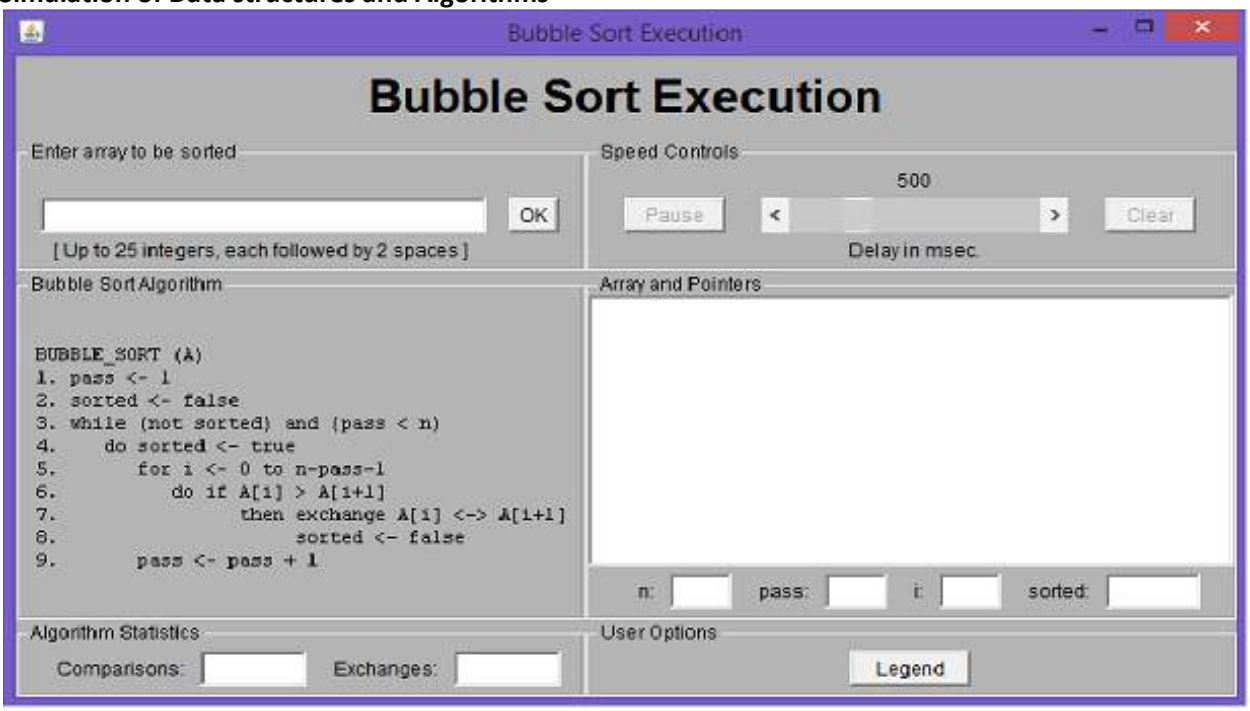


Figure 1: Bubble sort-This contain the text field for input values. The fields in comparisons and Exchanges gives the number of comparison and exchanges. it highlight the current code which is executing.

Code

```
public int[] bubbleSort(int[] data){
int lenD = data.length;
    int tmp = 0;
for(int i = 0;i<lenD;i++){
for(int j = (lenD-1);j>=(i+1);j--){
if(data[j]<data[j-1]){
tmp = data[j];
data[j]=data[j-1];
data[j-1]=tmp;
```

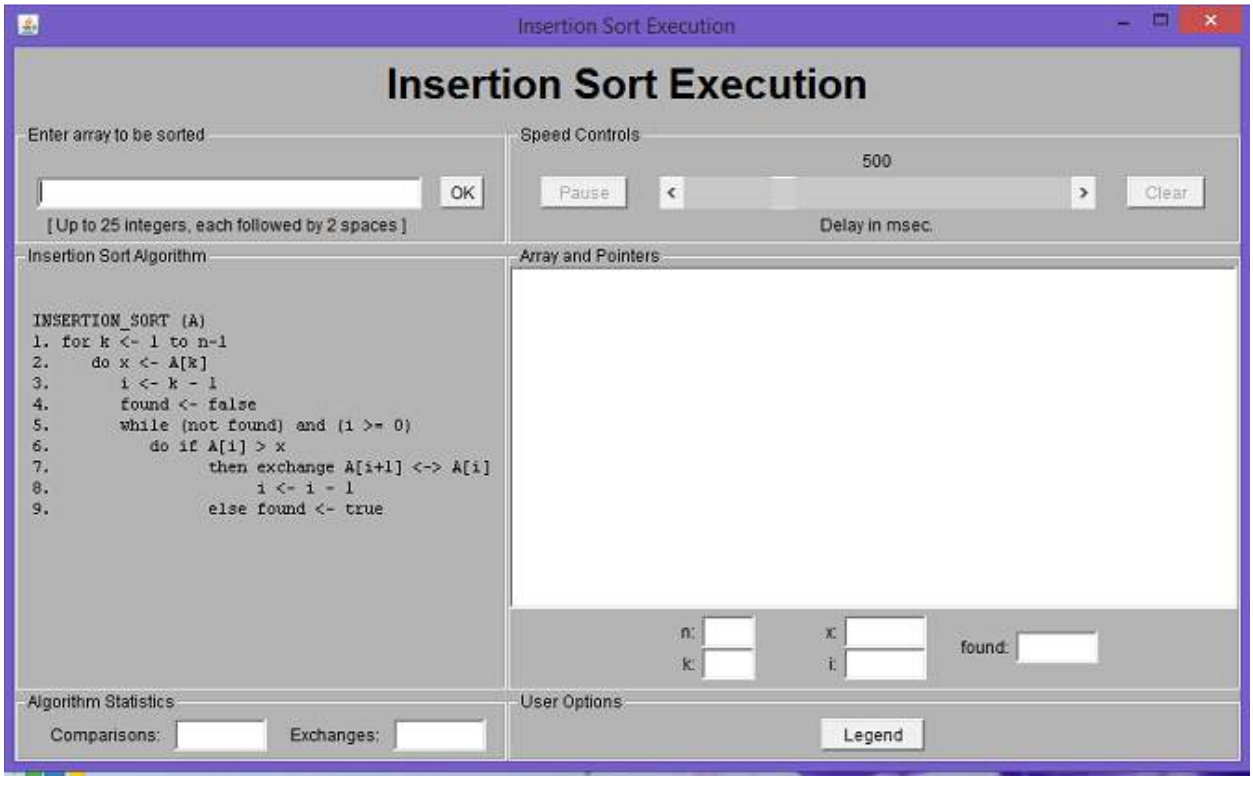
return data;

Insertion Sort

Insertion sort-This contain the text field for input values. The fields in comparisons and Exchanges gives the number of comparison and exchanges. it highlight the current code which is executing.n gives no.of values,x,k and i are the pointers.

Code

```
void SortAlgo::insertionSort(int data[], int lenD)
{
int key = 0;
int i = 0;
for(int j = 1;j<lenD;j++){
key = data[j];
i = j-1;
while(i>=0 && data[i]>key){
data[i+1] = data[i];
i = i-1;
data[i+1]=key;
```



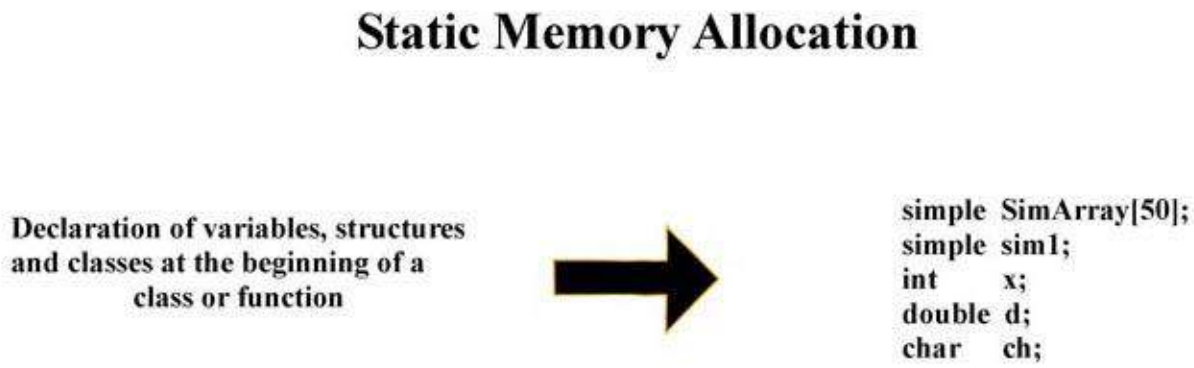
Memory Allocations in Data Structures

Memory allocation is the process of setting aside sections of memory in a program to be used to store variables, and instances of structures and classes. There are two types of memory allocations possible in C/C++:

1. Compile-time or Static allocation.
2. Run-time or Dynamic allocation (using pointers).

Compile-time or Static allocation

- **Static memory allocation** allocated by the compiler. Exact size and type of memory must be known at compile time.



When you declare a variable or an instance of a structure or class. The memory for that object is allocated by the operating system. The name you declare for the object can then be used to access that block of memory.

```
int x, y;
float a[5];
```

When the first statement is encountered, the compiler will allocate two bytes to each variables x and y. The second statement results into the allocction of 20 bytes to the array a (5*4, where there are five elements and each element of float type tales four bytes). Note that as there is no bound checking in C for array boundaries, *i.e.,* if you have declared an array of five elements, as above and by mistake you are intending to read more than five values in the array **a,** it will still work without error. For example you are reading the above array as follows :

```
for ( i = 0 ; i < 10 ; i++)
{
```

```
  scanf ("%d", &a[i]);
}
```

Run-time or Dynamic allocation

- Dynamic memory allocation is when an executing program requests that the operating system give it a block of main memory. The program then uses this memory for some purpose. Usually the purpose is to add a node to a data structure. In object-oriented languages, dynamic memory allocation is used to get the memory for a new object.
- The memory comes from above the static part of the data segment. Programs may request memory and may also return previously dynamically allocated memory. Memory may be returned whenever it is no longer needed. Memory can be returned in any order without any relation to the order in which it was allocated. The heap may develop "holes" where previously allocated memory has been returned between blocks of memory still in use.
- A new dynamic request for memory might return a range of addresses out of one of the holes. But it might not use up all the hole, so further dynamic requests might be satisfied out of the original hole.

C provides the following dynamic allocation and de-allocation functions :

- malloc( )
- calloc( )
- free( )
- realloc( )