Sri SAI RAM ENGINEERING COLLEGE
INSTITUTE OF TECHNOLOGY
West Tambaram, Chennai - 44

| YEAR | SEM |
|---|---|
| II | III |

# CS8391

# DATA STRUCTURES

UNIT No. 1

APPLICATIONS OF LINKED LIST

- RADIX SORT
- MULTI LIST

Version: 1.XX

## UNIT I APPLICATIONS OF LIST

- In order to store and process data, linked list are very popular data structures.
- This type of data structures holds certain advantages over arrays.

**Demerits of arrays and Merits of linked lists**

- First, in case of an array, data are stored in contiguous memory locations, so insertion and deletion operations are quite time consuming, in insertion of a new element at a desired location , all the trailing elements should be shifted down; similarly, in case of deletion, in order to fill the location of deleted element, all the trailing elements are required to shift upwards. But, in linked lists, it is a matter of only change in pointers.
- Second, array is based on the static allocation of memory: amount of memory required for an array must be known before hand, once it is allocated its size cannot be expanded. This is why, for an array, general practice is to allocate memory, which is much more than the memory that actually will be used. But this is simply wastage of memory space. This problem is not there in linked list.
- **Linked list uses dynamic memory management scheme;** memory allocation is decided during the run-time as and when require. Also if a memory is no more required, it can be returned to the free storage space, so that other module or program can utilize it.
- Third, a program using an array may not execute although the memory required for the data are available but not in contiguous locations rather dispersed. As link structures do not necessarily require to store data in adjacent memory location, so the program of that kind, using linked list can then be executed.

**Demerits of linked lists**

- However, there are obviously some disadvantages: one is the pointer business. Pointers, if not managed carefully, may lead to serious errors in execution.
- Next, linked lists consume extra space than the space for actual data as the links among the nodes are to be maintained.

Five examples are provided that use linked lists.

- The first is a simple way to represent single-variable polynomials.
- The second is a method to sort in linear time, for some special cases.
- Thirdly, linked lists might be used to keep track of course registration at a university.
- The next application is the representation of sparse matrix
- The last application is Dynamic Storage management

## Multi lists:

- Last example shows a more complicated use of linked lists. A university with 40,000 students and 2,500 courses needs to be able to generate two types of reports. The first report lists the class registration for each class, and the second report lists, by student, the classes that each student is registered for.
- The obvious implementation might be to use a two-dimensional array. Such an array would have 100 million entries. The average student registers for about three courses, so only 120,000 of these entries, or roughly 0.1 percent, would actually have meaningful data.
- What is needed is a list for each class, which contains the students in the class. A list for each student is also needed, which contains the classes the student is registered for.

As the figure shows, two lists are combined into one. All lists use a header and are circular.

- To list all of the students in class C3, start at C3 and traverse its list (by going right).
- The first cell belongs to student S1. Although there is no explicit information to this effect, this can be determined by following the student's linked list until the header is reached.
- Once this is done, return to C3's and find another cell, which can be determined to belong to S3. Continue and find that S4 and S5 are also in this class. In a similar manner, for any student, all of the classes in which the student is registered can be determined.

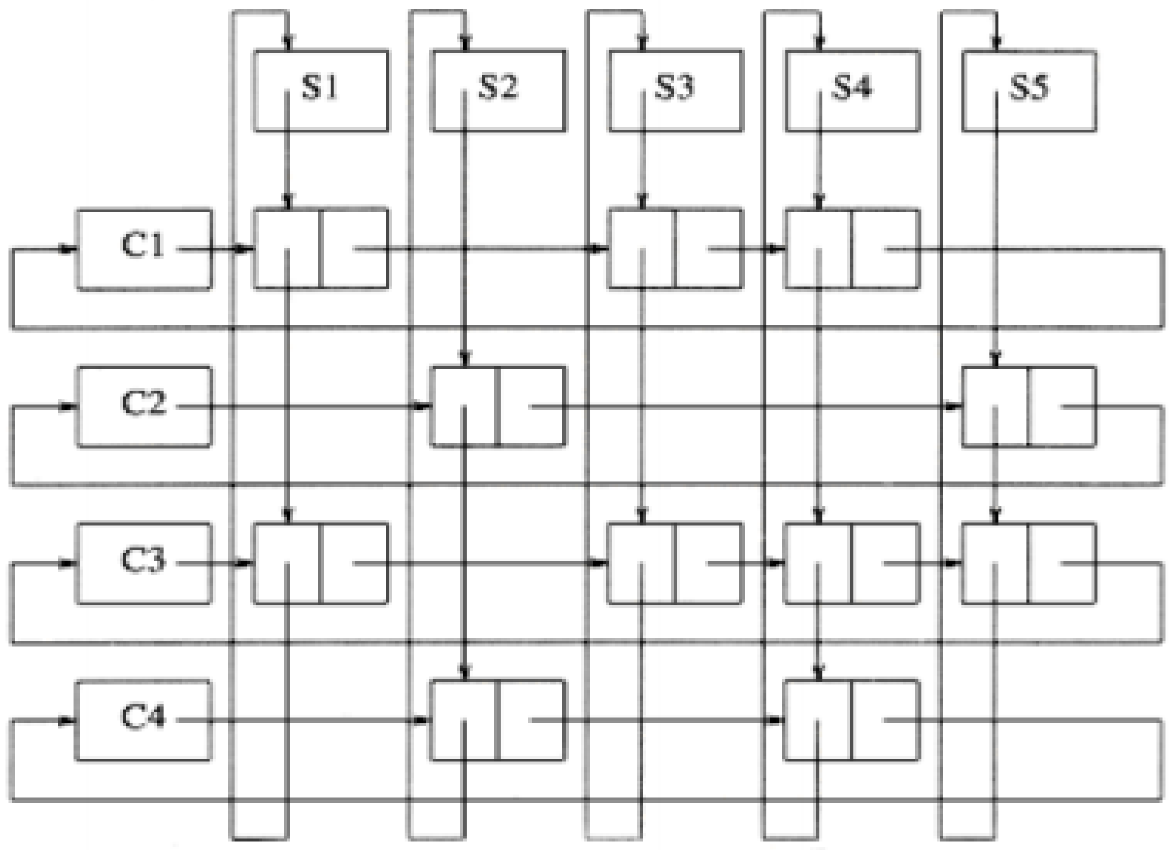

**FIG: Multi list implementation for registration problem**

## RADIX SORT

**Radix sort** is a small method that many people intuitively use when alphabetizing a large list of names. Specifically, the list of names is first sorted according to the first letter of each name, that is, the names are arranged in 26 classes.

- Intuitively, one might want to sort numbers on their most significant digit. However, Radix sort works counter-intuitively by sorting on the least significant digits first.
- On the first pass, all the numbers are sorted on the least significant digit and combined in an array.
- Then on the second pass, the entire numbers are sorted again on the second least significant digits and combined in an array and so on.

**Algorithm:**

**Algorithm: Radix-Sort (list, n)**

```
shift = 1
for loop = 1 to keysize do
   for entry = 1 to n do
      bucketnumber = (list[entry].key / shift) mod 10
      append (bucket[bucketnumber], list[entry])
   list = combinebuckets()
   shift = shift * 10
```

**Example**

**Following example shows how Radix sort operates on seven 3-digits number.**

| Input | 1st Pass | 2nd Pass | 3rd Pass |
|---|---|---|---|
| 329 | 720 | 720 | 329 |
| 457 | 355 | 329 | 355 |
| 657 | 436 | 436 | 436 |
| 839 | 457 | 839 | 457 |
| 436 | 657 | 355 | 657 |
| 720 | 329 | 457 | 720 |
| 355 | 839 | 657 | 839 |

In the above example, the first column is the input. The remaining columns show the list after successive sorts on increasingly significant digits position. The code for Radix sort assumes that each element in an array ***A*** of ***n*** elements has ***d*** digits, where digit ***1*** is the lowest-order digit and ***d*** is the highest-order digit