Data Structures (15B11CI311)

Odd Semester 2020

3rd Semester , Computer Science and Engineering

Jaypee Institute Of Information Technology (JIIT), Noida

Batches : B1 , B2 , B13
Faculty : Ankita Wadhwa

Lecture 14&15

Multi list and List of List

Topics covered

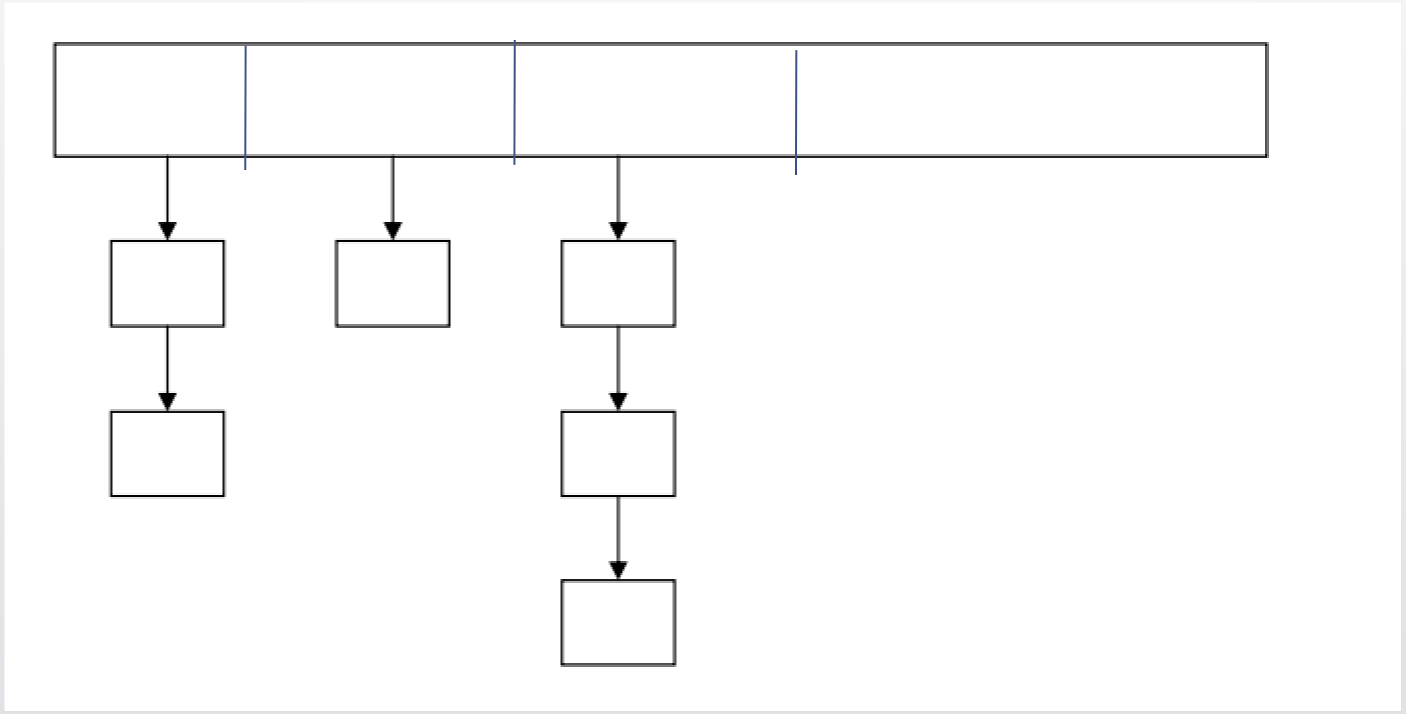
- ✓ Array of Linked Lists (Multi list)
- ✓ Example
- ✓ Creating an Array of Linked Lists
- ✓ Query Implementation
- ✓ List of list
- ✓ Creating list of list
- ✓ Sparse Matrix
- ✓ Implementation
- ✓ Multi dimensional list
- ✓ Flattening the linked list (horizontally)
- ✓ Flattening the linked list (Depth wise)
- ✓ Nested List

1. Array of Linked List (Multi List)

Multi list (An Array of Linked list)

- A linked list is defined as a collection of nodes that can be traversed starting at the head node.
- It is important to note that head is not a node, rather the address of the first node of the list.
- Linked lists are very useful in situations where the program needs to manage memory very carefully and a contiguous block of memory is not needed.
- An array of linked lists is an important data structure that can be used in many applications (like time table).

View of array of linked list



Array of linked list (contd..)

- An array of linked list is an interesting structure as it combines a static structure (an array) and a dynamic structure (linked lists) to form a useful data structure.

- This type of a structure is appropriate for applications, where say for example, number of categories is known in advance, but how many nodes in each category is not known.

Example 1: Time Table of Second year Students

While creating a time table, we assume that data is represented in 2D form.

Row represents the week days and column represent the time slots. Since we are using here array of pointer to form linked list, so week days will be fixed, however column can dynamically grow.

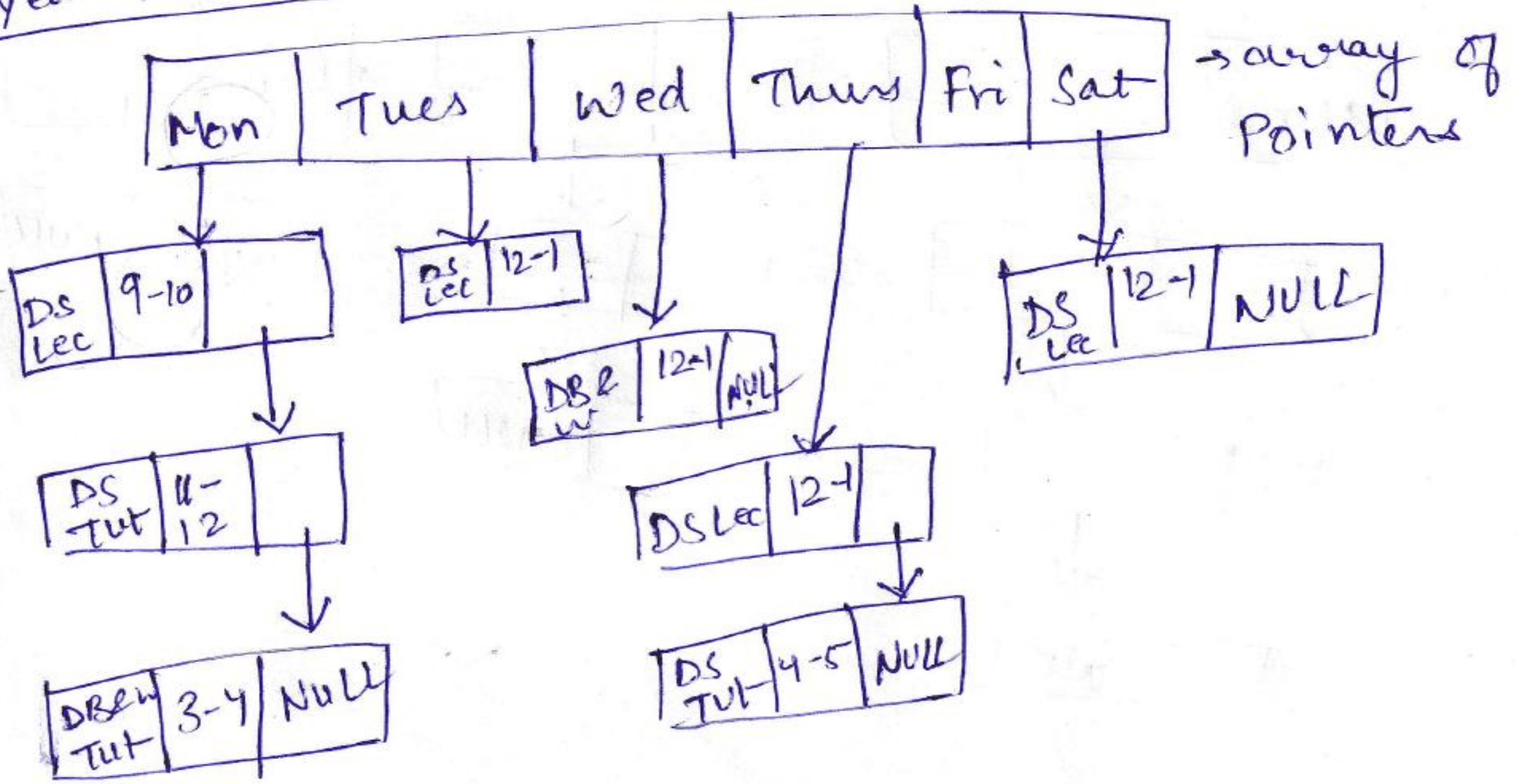
Example (contd..)

- If we use arrays only then structure would be like this

Monday	Tuesday	Wed	Thurs	Fri	Sat
DS Lect					
	DS Lect	DB&W Lect		DB&W Lect	
DS TUT					
DB&W Lect			DS Lect		DS Lect
			DS TUT		
DB&W Tut					

- when we implement using linked list.

II year time table



Implementation Of Array Of List (Multi List)

```
struct Node
{   int slot;
    int subcode;
    Node *next;
};
class ML
{   Node *heads[6];
    public:
    ML()
    {       for(int i=0;i<6;i++)
            heads[i]=NULL;
    }
 void sortedInsert(int listno,Node * new_node);
 void print_given_List(int listno);
 void print_ML()
};
```

```
void ML :: print_given_List(int listno)
  {
    Node* head=heads[listno];
    while (head != NULL)
    {
        cout <<"slot->"<<head->slot<< ",";
         cout <<"subj->" <<head->subcode<< " ";
        head = head->next;
    }
    cout<<endl;
  }

  void ML :: print_ML()
  {
    for(int i=0;i<6;i++)
    {
        cout<<"\n printing list number "<<i;
        print_given_List(i);
    }
  }
```

```
void ML::sortedInsert(int listno,Node* new_node)
{
    Node* current;

         if (heads[listno] == NULL || heads[listno]->slot >= new_node->slot)
    {
        new_node->next = heads[listno];
        heads[listno]= new_node;
    }
    else
    {
         current = heads[listno];
        while (current->next != NULL && current->next->slot < new_node->slot)
        {
            current = current->next;
        }
        new_node->next = current->next;
        current->next = new_node;
    }
}
```

```
int main()
{
  int i,list_no,choice;
  int tempslot;
  int tempcode;

  ML list1;
```

```
do
    {       cout<<"1: add node"<<endl;
            cout<<"2: quit"<<endl;
            cin>>choice;
            if(choice==1)
            {
                    Node *newnode=new Node;

                     cout<<"In which listno of multilist do you want to add slot?\n";
                     cin>>list_no;
                     cout<<"enter slot";
                     cin>>tempslot;
                     cout<<"enter sub_code";
                     cin>>tempcode;

                     newnode->slot=tempslot;
                     newnode->subcode=tempcode;
                     newnode->next=NULL;
                     list1.sortedInsert(list_no,newnode);
             }
    }while(choice==1);

     cout<<"\n printing the mentioned list number:\n";
     list1.print_given_List(3);
     cout<<"\n printing complete list :";
     list1.print_ML();
     return 0;
}
```

Query Implementation

- Query
  - Which day is highly engaged for second year students
    - Count number of list under each head and print the head having maximum list
  - How many lectures of DS are there on Saturday.

```
int ML :: countnodes(int listno)
  {
     Node* current=heads[listno];
     int count=0;
     while (current != NULL)
     {
        count++;
        current = current->next;
     }
     return count;
  }
```

```
int ML :: find_busiest_day()
{
   int global_max=0;
   int local_max=0;
   int busy_day=0;

   for(int i=0;i<6;i++)
   {
      local_max=countnodes(i);
      if(local_max>global_max)
      {
         global_max=local_max;
         busy_day=i;
      }
   }
   return busy_day;
}
```

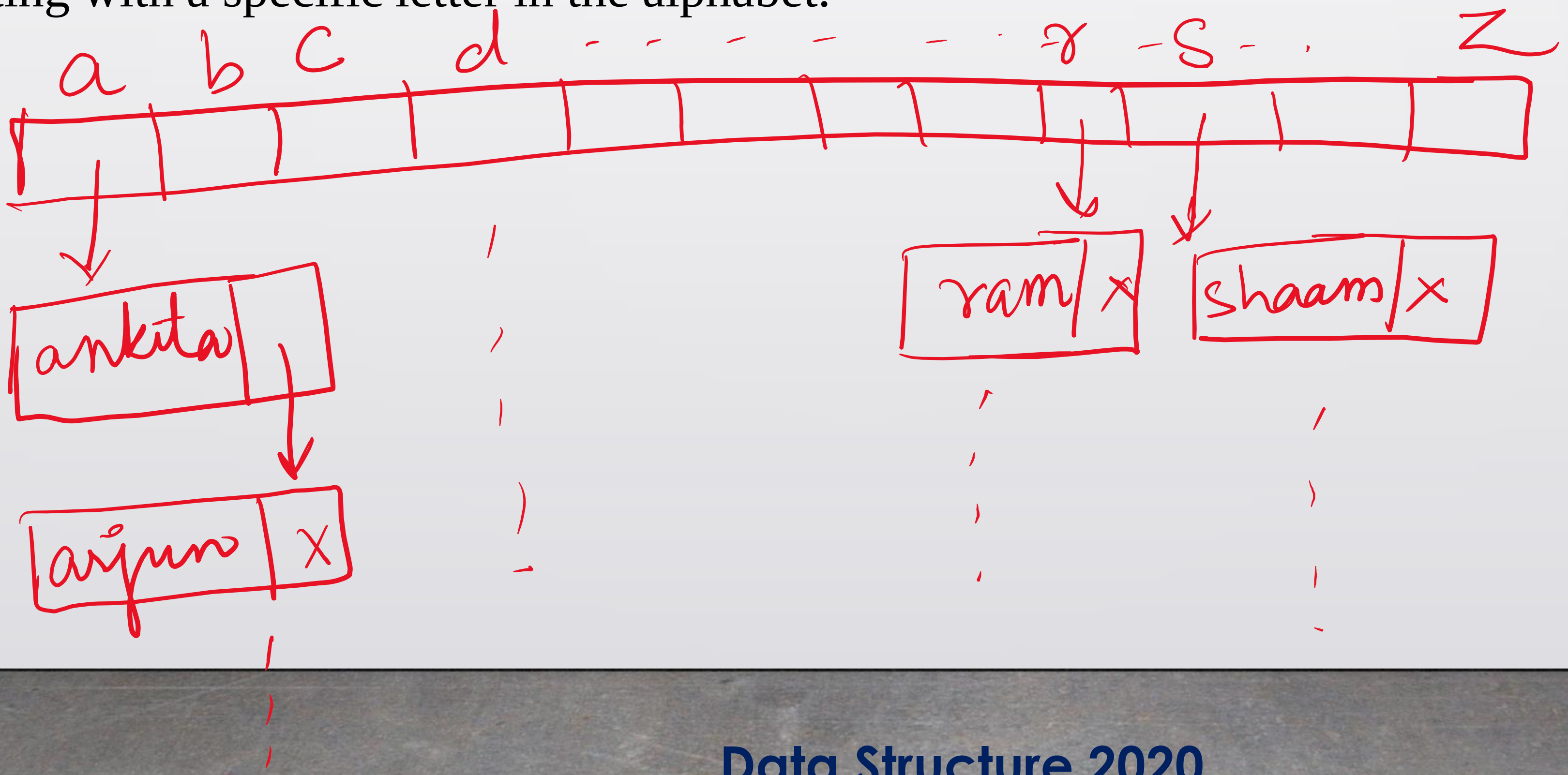
Call the function in main() as :

```
cout<<"\n the busiest day is "<<list1.find_busiest_day();
```

HW: Attempt the second query

Example 2: List of Students in B1B2B13 lecture section alphabetically

- we can use an array (of size 26) of linked lists, where each list contains names of students starting with a specific letter in the alphabet.



Array Of List : Limitations

- Cannot grow the number of categories dynamically. Not suitable for applications where the number of categories can vary.
- Suitable for one type of query.
- For ex 1: If the query is:

  " Print all the days when 2nd year students are free in slot 10-11 am"
- For ex 2: If the query is:

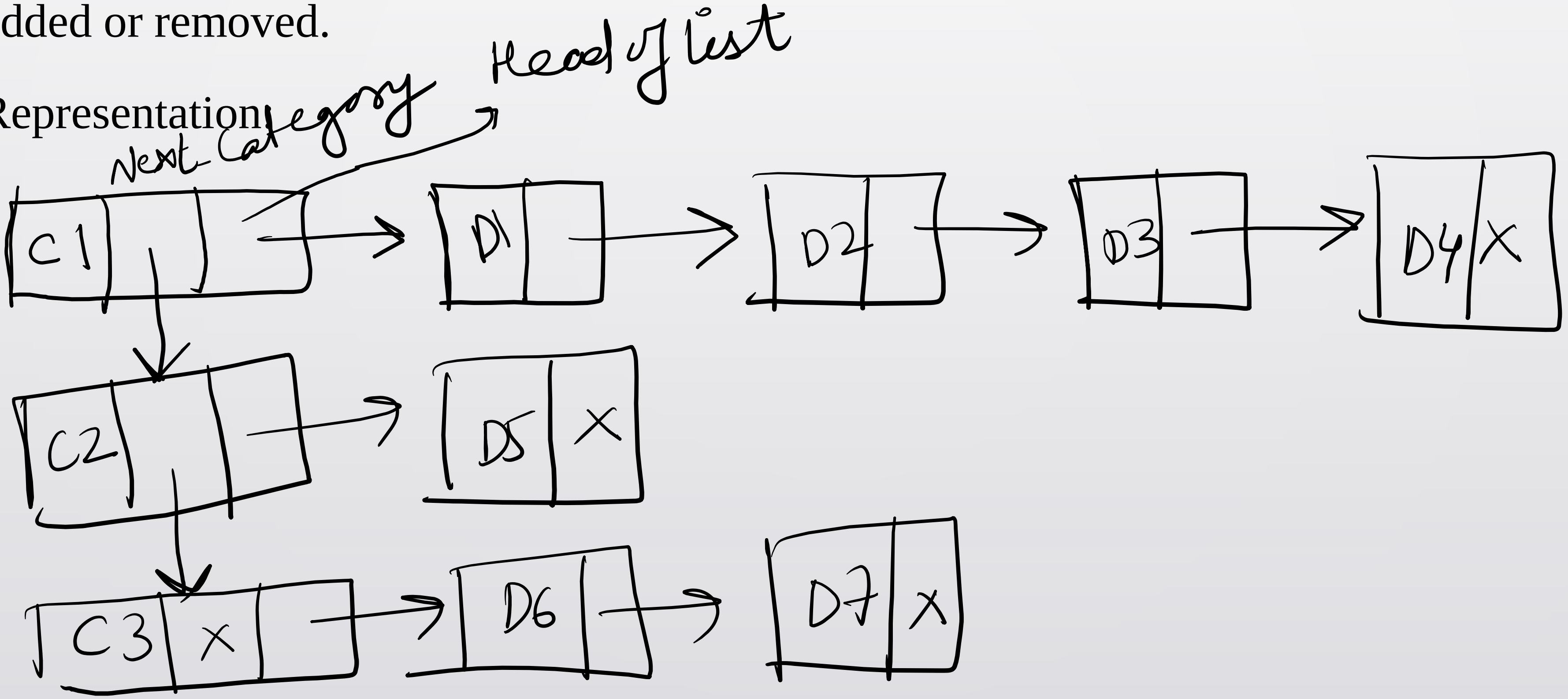
  "Print the name of student having highest CGPA".

This query requires all the nodes to be traversed (no benefit from the arrangement of data).

2. List of List

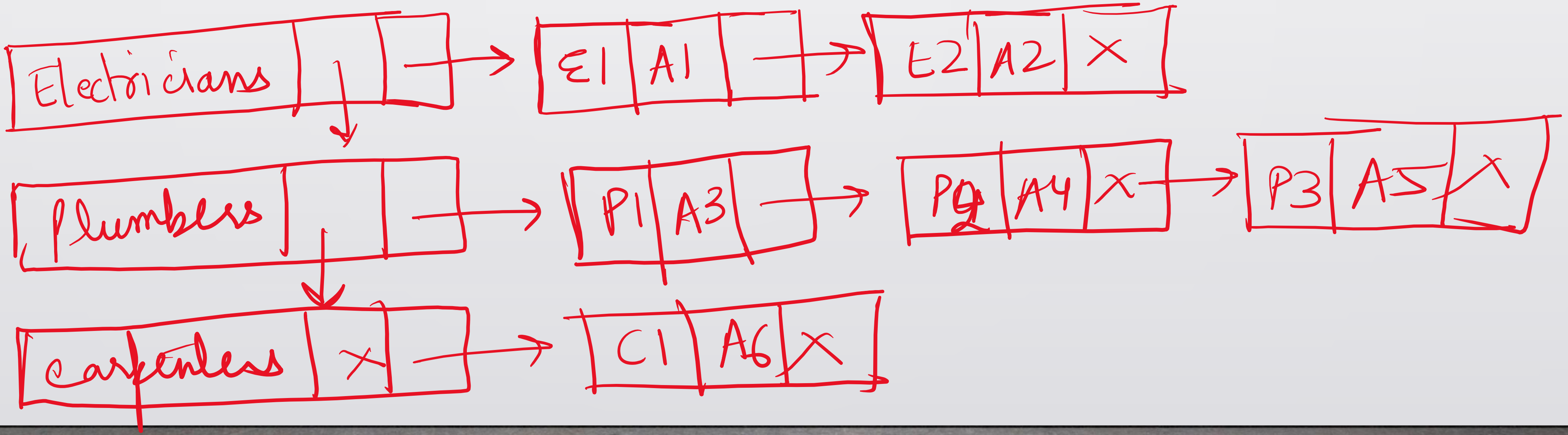
List of List

- When we use linked list instead of array and each node of list points to a linked list then we call it List of list data structure.
- The number of categories as well as number of nodes in each category can be dynamically added or removed.
- Representation:



Example 1: Yellow Pages directory (nowadays urban clap)

- Lets say we want to store contact details of different service providers in a city.
- These service providers can be electricians, plumbers, carpenters etc.
- And for each category of service providers, we can have any number of contacts.
- We need to store it in such a DS that category of service providers as well as number of service providers in each category is dynamic.
- List of List is appropriate here.

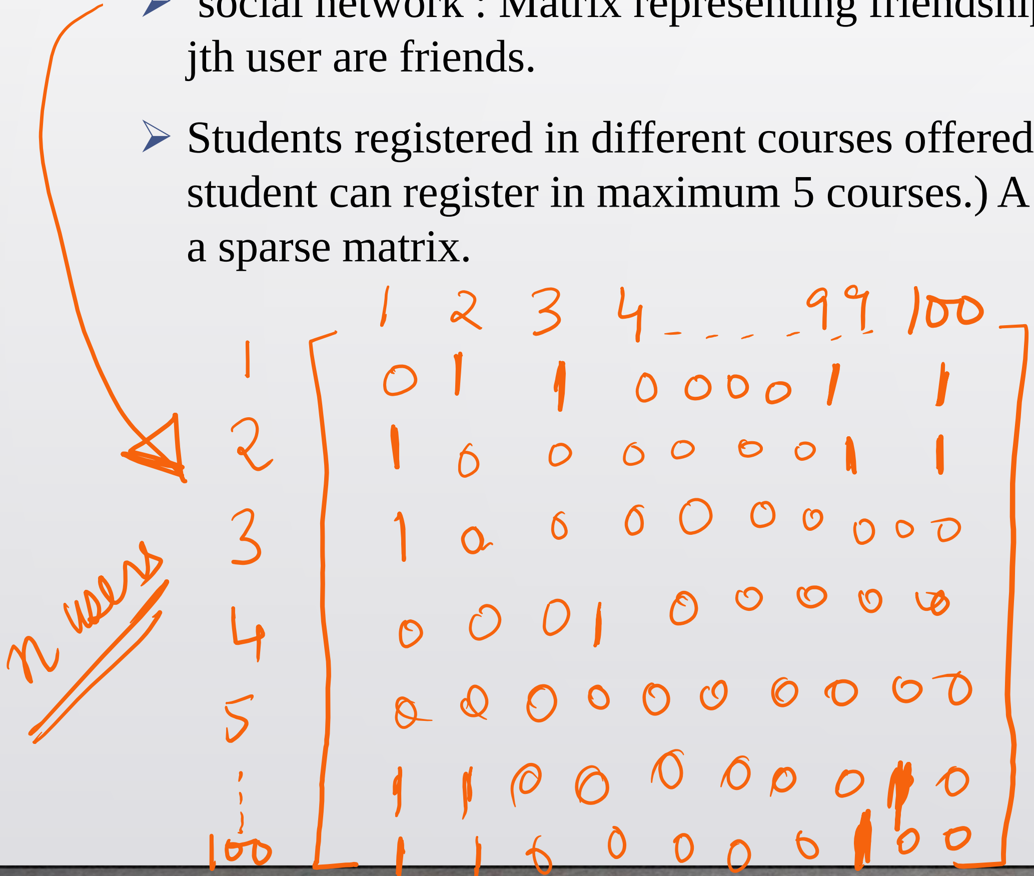


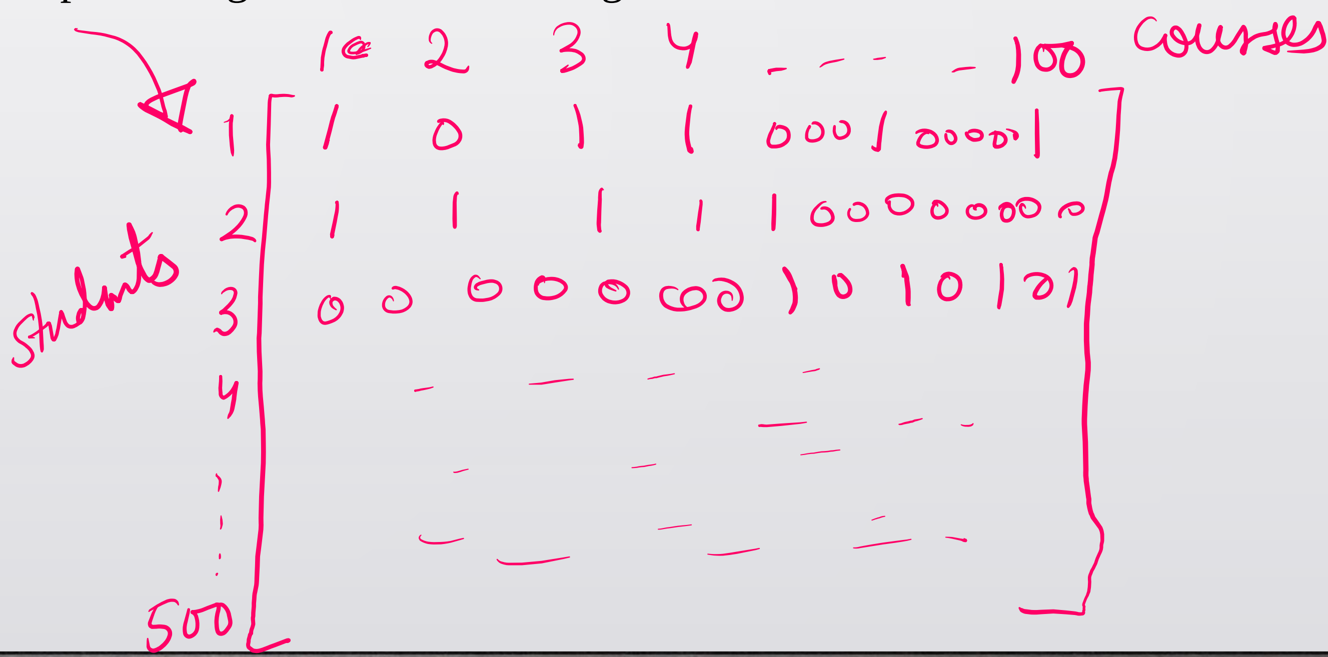
Example 2: Sparse matrix

- A matrix having majority of the elements as zero is called sparse matrix.
- Some real world scenarios where sparse matrix exist.

➢ social network : Matrix representing friendship: A nxn matrix representing ‘n’ users. ‘1’ at (i,j) indicates that ith and jth user are friends.

➢ Students registered in different courses offered by a university. (Lets say a university offers 100 courses and each student can register in maximum 5 courses.) A matrix representing which student is registered in which course will be a sparse matrix.

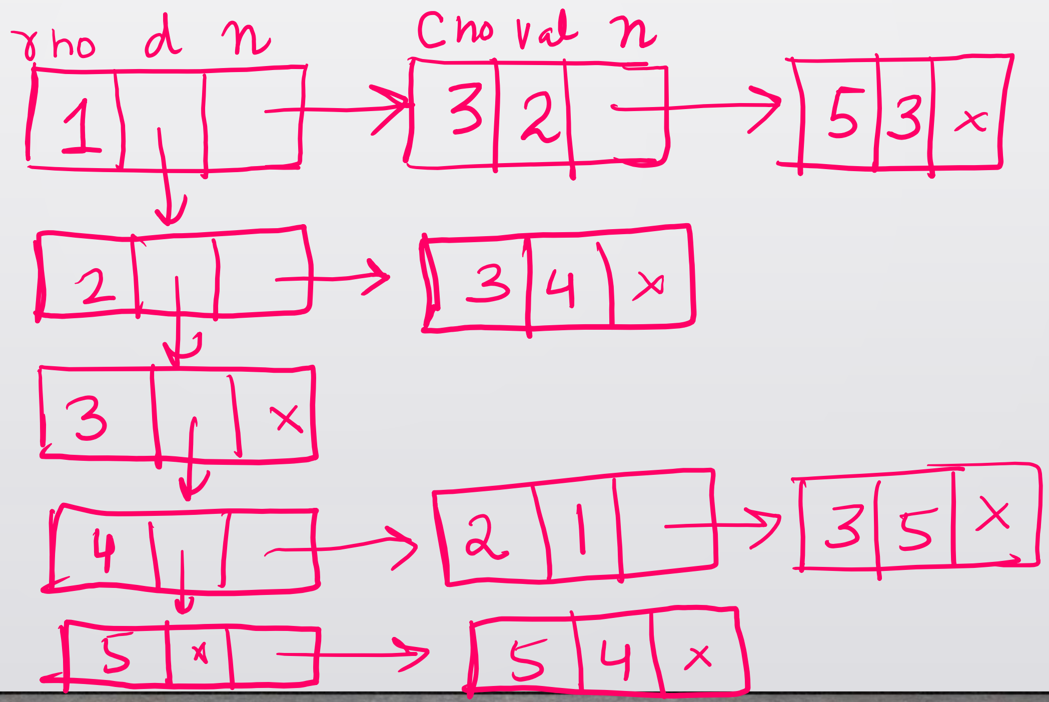




Sparse Matrix

- Linked list representation of sparse matrix saves a lot of memory space.
- This representation uses two linked list.
- First list contains value of row number and two pointers, one points to next node and other pointer points to non zero elements in that row.
- Second list contains column number, value in that position and pointer to next node.

	1	2	3	4	5
1	0	0	2	0	3
2	0	0	4	0	0
3	0	0	0	0	0
4	0	1	5	0	0
5	0	0	0	0	4



Structure of Nodes

```
struct value_list
{
    int column_index;
    int value;
    value_list *next;
};
```

```
struct row_list
{
    int row_number;
    row_list *link_down;
    value_list *link_right;
};
```

```
class sparse_lol
{
    row_list *start;
    int rows;
    public:
    sparse_lol()
    {
        start=NULL;
        rows=0;
    }

    void create_list(int Sparse_Matrix[R][C]);
    void create_value_node(int data, int j, row_list *z);
    void print_LIL();
    void add_row(int row[]);

};
```

```
void sparse_lol::create_list(int Sparse_Matrix[R][C])
{   for (int i = 0; i < R; i++)
    {
        row_list *z, *r;
        z = new row_list;
        rows++;
        z->row_number = i+1;
        z->link_down = NULL;
        z->link_right = NULL;
        if (i==0)
            start = z;
        else
        {   r = start;
            while (r->link_down != NULL)
                r = r->link_down;
            r->link_down = z;
        }
        for (int j = 0; j < C; j++)
        {   if (Sparse_Matrix[i][j] != 0)
            {
                create_value_node(Sparse_Matrix[i][j], j, z);
            }
        }
    }
}
```

```
void sparse_lol::add_row(int row[])
{
        row_list *z, *r;
        z = new row_list;
        z->row_number = rows+1;
        rows++;
        z->link_down = NULL;
        z->link_right = NULL;
        if (rows==0)
            start = z;
        else
        {
            r = start;
            while (r->link_down != NULL)
                r = r->link_down;
            r->link_down = z;
        }
        for (int j = 0; j < C; j++)
        {
            if (row[j]!= 0)
            {
                create_value_node(row[j], j, z);
            }
        }
}
```

```
void sparse_lol::create_value_node(int data, int j, row_list *z)
{
    value_list *temp, *d;
    temp = new value_list;
    temp->column_index = j+1;
    temp->value = data;
    temp->next = NULL;
    if (z->link_right==NULL)
        z->link_right = temp;
    else
    {   d = z->link_right;
        while(d->next != NULL)
            d = d->next;
        d->next = temp;
    }
}
```

```
void sparse_lol::print_LIL()
{
     row_list *r;
     value_list *z;
    r = start;
    while (r != NULL)
    {
        if (r->link_right != NULL)
        {
           cout<<"\nrow= "<< r->row_number;
            z = r->link_right;
            while (z != NULL)
            {
                cout<<"\n column= "<< z->column_index<< " value= "<<z->valu
                z = z->next;
            }
        }
        r = r->link_down;
    }
}
```

```
int main()
{
    int Sparse_Matrix[R][C] =
    {
        {0 , 0 , 3 , 0 , 4 },
        {0 , 0 , 5 , 7 , 0 },
        {0 , 0 , 0 , 0 , 0 },
        {0 , 2 , 6 , 0 , 0 }
    };

    sparse_lol list1;

    list1.create_list(Sparse_Matrix);
    list1.print_LIL();

    int rownew[]={0,0,5,0,0};
    list1.add_row(rownew);
    list1.print_LIL();

    return 0;
}
```

3. Multi Dimensional Lists

Multi Dimensional Lists

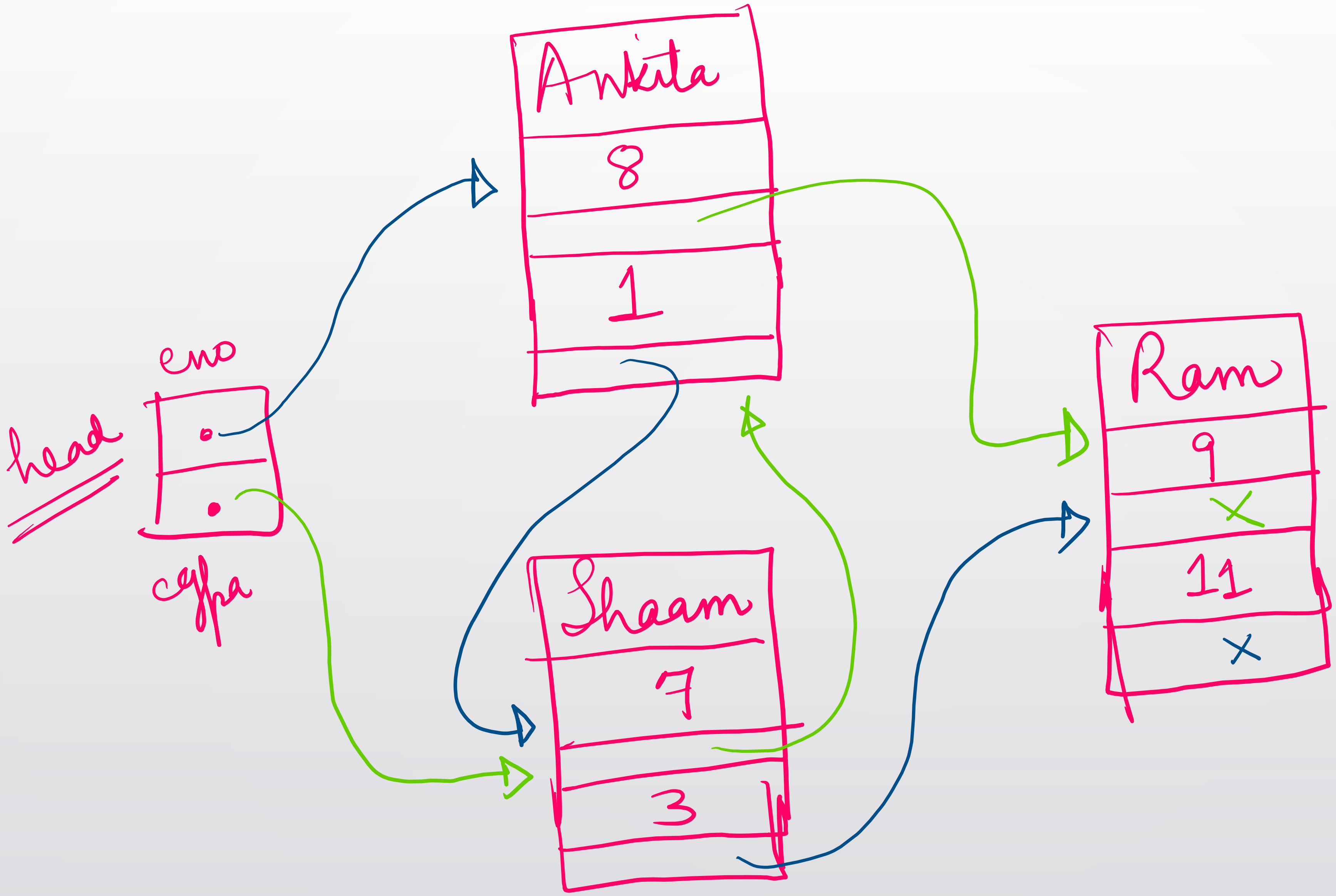
- Multi dimensional lists grow dynamically.
- Multi dimensional queries can be addressed with the help of multiple pointers. (different pointers for different types of queries).
- We can maintain Multiple Orders Of One Set Of Elements using multiple pointers.
- Example :

➢ I want to organize a collection of students in two different ways. For example, suppose my elements include the name, CGPA and enrollment number of a student.

➢ (ANKITA,8,1) (RAM,9,11) (SHAAM, 7, 3)

➢ I might want to order these students by CGPA and also order them by Eno. I would have two pointers -

➢ NEXT-cgpa, NEXT-eno- and the list header would have two pointers, one based on cgpa, the other on eno


head
eno
cgpa
Ankita
8
1
Shaam
7
3
Ram
9
×
11
×

```
class student
{
    public:
    char* name;
    int cgpa;
    int rno;
    student(char* n,int cg,int rn)
    {
        name=n;
        cgpa = cg;
        rno = rn;
    }
};
```

```
class node
{
    public:
    student *sobj;
    node *p1, *p2;

   void  createNode(student *s)
   {    sobj = s;
        p1=p2=NULL;
   }
};
```

```
class MDList
{
    node *head[2];

    public:
    MDList()
    {
        head[0]=head[1]= NULL;
    }

   int addNode(student *s)
    {
        node *newNode = new node;
        newNode->createNode(s);

        if(head[0] == NULL)
        {

            cout<<"at head =0 \n";
            head[0]=head[1] = newNode;
            return 1;
        }

        insertbyCGPA(newNode);
        insertbyRno(newNode);
    }
```

```
void displayWRTcgpa()
    {
        node *temp = head[0];
        cout<<"\n at cgpa wise display \n";
        while(temp)
        {   cout<<temp->sobj->name<<endl;
            temp=temp->p1;
        }
    }

    void displayWRTrn()
    {
        node *temp = head[1];
        cout<<"\n at roll no wise display \n";
        while(temp)
        {   cout<<temp->sobj->name<<endl;
            temp=temp->p2;
        }
    }

    void insertbyCGPA(node* new_node);
    void insertbyRno(node* new_node);


}; //end of class
```

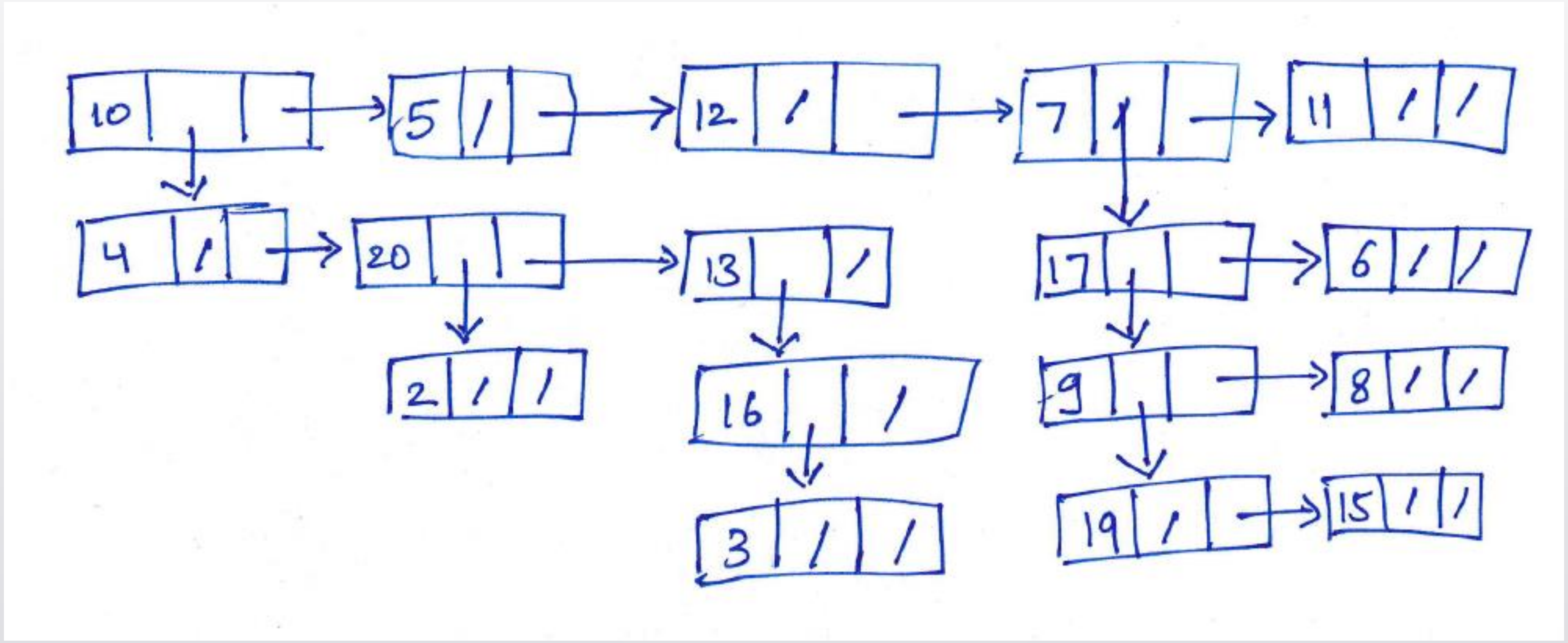
```
void MDList::insertbyCGPA(node* new_node)
{
    node* current;

         if (head[0]->sobj->cgpa >= new_node->sobj->cgpa)
    {
        new_node->p1 = head[0];
        head[0]= new_node;
    }
    else
    {
             current = head[0];
        while (current->p1 != NULL && current->p1->sobj->cgpa < new_node->sobj->cgpa)
        {
            current = current->p1;
        }
        new_node->p1 = current->p1;
        current->p1 = new_node;
    }
}
```

```
void MDList::insertbyRno(node* new_node)
{
    node* current;

         if (head[1]->sobj->rno >= new_node->sobj->rno)
    {
        new_node->p2 = head[1];
        head[1]= new_node;
    }
    else
    {
             current = head[1];
        while (current->p2 != NULL && current->p2->sobj->rno < new_node->sobj->rno)
        {
            current = current->p2;
        }
        new_node->p2 = current->p2;
        current->p2 = new_node;
    }
}
```

```
int main()
{
    student *s[3];
    s[0] = new student("Ankita",8,1);
    s[1] = new student("Ram",9,11);
    s[2] = new student("Shaam",7,3);

    MDList slist;

    slist.addNode(s[0]);
    //cout<<"\n";
    //slist.displayWRTrn();

    slist.addNode(s[1]);

     cout<<"\n";
    slist.addNode(s[2]);
    slist.displayWRTrn();
    slist.displayWRTcgpa();

    return 0;
}
```

4. Few more lists of lists , multi linked lists examples

Example 1



Creating the given list

```
#include <iostream>

using namespace std;
#define SIZE(arr) (sizeof(arr)/sizeof(arr[0]))

struct Node
{
    public:
    int data;
   Node *next;
    Node *child;
};
```

```
Node *createList(int *arr, int n)
{
    Node *head = NULL;
    Node *p;
    int i;
    for (i = 0; i < n; ++i)
    {
        if (head == NULL)
            head = p = new Node();
        else
        {
            p->next = new Node();
            p = p->next;
        }
        p->data = arr[i];
        p->next = p->child = NULL;
    }
    return head;
}
```

```
Node *createList()
{
   int arr1[] = {10, 5, 12, 7, 11};
   int arr2[] = {4, 20, 13};
   int arr3[] = {17, 6};
   int arr4[] = {9, 8};
   int arr5[] = {19, 15};
   int arr6[] = {2};
   int arr7[] = {16};
   int arr8[] = {3};
   Node *head1 = createList(arr1, SIZE(arr1));
   Node *head2 = createList(arr2, SIZE(arr2));
   Node *head3 = createList(arr3, SIZE(arr3));
   Node *head4 = createList(arr4, SIZE(arr4));
 Node *head5 = createList(arr5, SIZE(arr5));
   Node *head6 = createList(arr6, SIZE(arr6));
   Node *head7 = createList(arr7, SIZE(arr7));
   Node *head8 = createList(arr8, SIZE(arr8));

   head1->child = head2;
   head1->next->next->next->child = head3;
   head3->child = head4;
   head4->child = head5;
   head2->next->child = head6;
   head2->next->next->child = head7;
   head7->child = head8;
   return head1;
}
```

```
int main()
{
    Node *head = NULL;
    head = createList();
    return 0;
}
```

Printing a given list of list like previous one row-wise (Flattening List horizontally)

- You are given the head of the first level of the list.
- Flatten the list so that all the nodes appear in a single-level linked list.
- You need to flatten the list in way that all nodes at first level should come first, then nodes of second level, and so on.

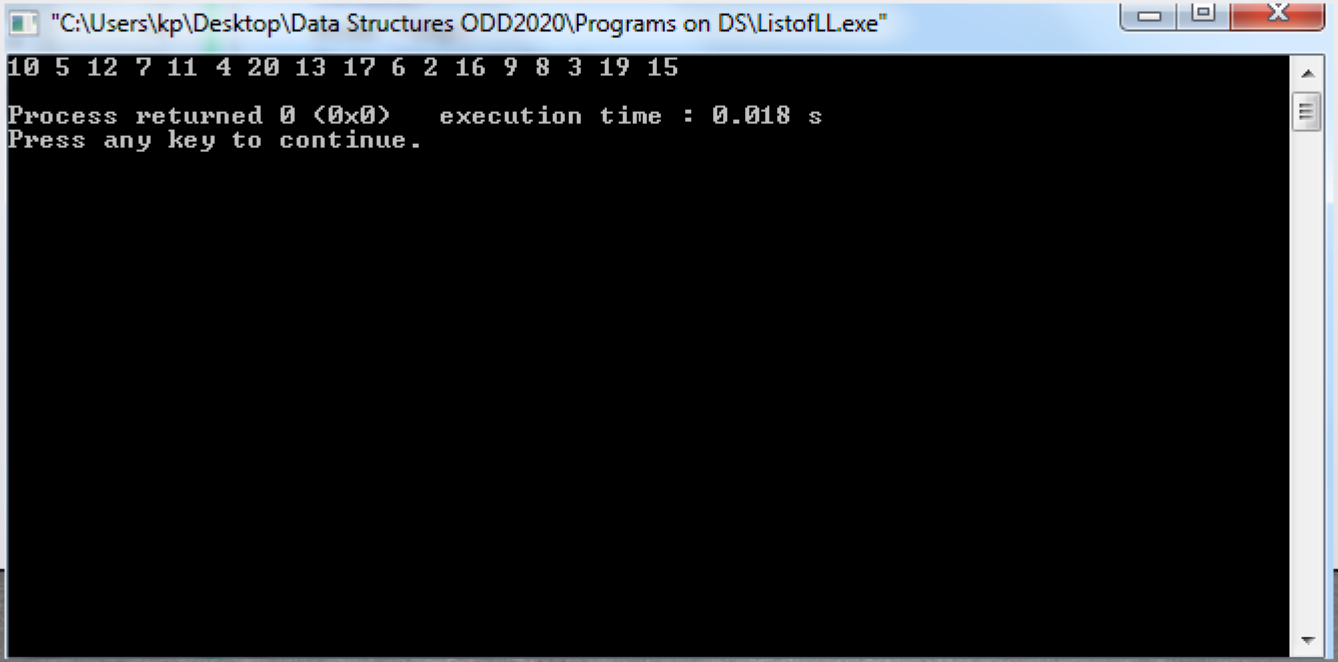
Creating a flatten function

```
void flattenList(Node *head)
{
    if (head == NULL)
            return;
     Node *tmp;
     Node *tail = head;
     while (tail->next != NULL)
         tail = tail->next;
```

```
 Node *cur = head;
while (cur != tail)
    {
 if (cur->child)
        {
            tail->next = cur->child;
             tmp = cur->child;
            while (tmp->next)
                tmp = tmp->next;
            tail = tmp;
        }
         cur = cur->next;
    }
}
```

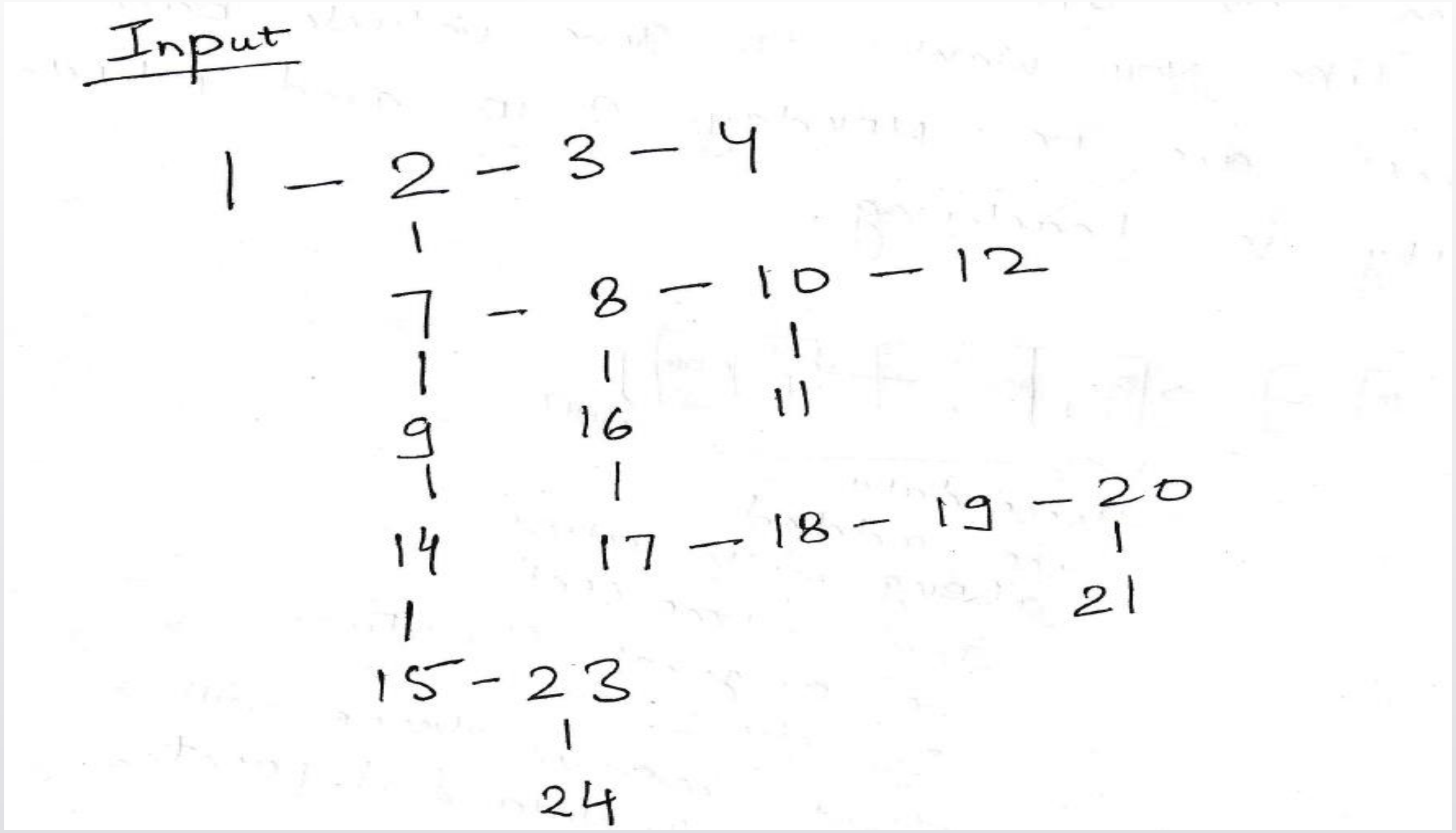
```
void printList(Node *head)
{
    while (head != NULL)
    {
        cout << head->data << " ";
        head = head->next;
    }
    cout<<endl;
}
```

```
int main(void)
{
    Node *head = NULL;
    head = createList();
    flattenList(head);
    printList(head);
    return 0;
}
```


"C:\Users\kp\Desktop\Data Structures ODD2020\Programs on DS\ListofLL.exe"

```
10 5 12 7 11 4 20 13 17 6 2 16 9 8 3 19 15

Process returned 0 (0x0)   execution time : 0.018 s
Press any key to continue.
```

Printing list of list Column-wise (Flattening List depth wise)



```
#include <iostream>
using namespace std;

struct Node
{
    int data;
    struct Node *next;
    struct Node *down;
};

void printFlattenNodes(Node* head)
{
    while (head)
    {
    printf("%d ", head->data);
    head = head->next;
    }
}
```

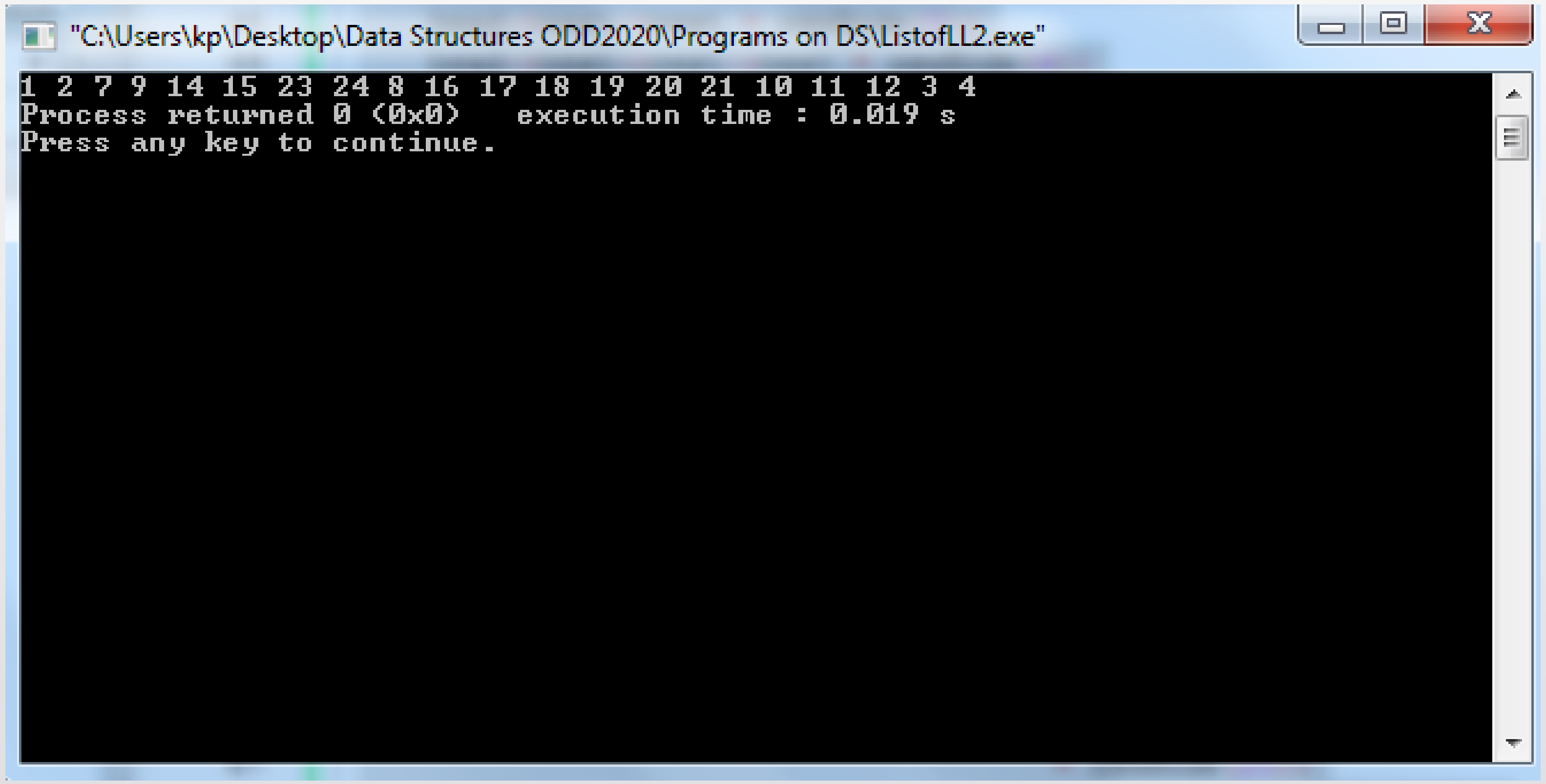
```
Node* newNode(int new_data)

{

    Node* new_node = new Node;

    new_node->data = new_data;

    new_node->next = new_node->down = NULL;

    return new_node;

}
```

Source:https://www.geeksforgeeks.org/flatten-a-multi-level-linked-list-set-2-depth-wise/

```
Node* flattenList(Node* node)

{    if (node == NULL)

         return NULL;

     Node *last;

     last = node;

    Node *next = node->next;

  if (node->down)

       node->next = flattenList(node->down);

   if (next)

       last->next = flattenList(next);

    return node;

}
```

```
int main()

{

    Node* head = newNode(1);

    head->next = newNode(2);

    head->next->next = newNode(3);

    head->next->next->next = newNode(4);

    head->next->down = newNode(7);

    head->next->down->down = newNode(9);

    head->next->down->down->down = newNode(14);

    head->next->down->down->down->down   = newNode(15);

    head->next->down->down->down->down->next   = newNode(23);

    head->next->down->down->down->down->next->down= newNode(24);

   head->next->down->next = newNode(8);

    head->next->down->next->down = newNode(16);

    head->next->down->next->down->down = newNode(17);
head->next->down->next->down->down->next  = newNode(18);
head->next->down->next->down->down->next->next= newNode(19);
head->next->down->next->down->down->next->next->next= newNode(20);
head->next->down->next->down->down->next->next->next->down= newNode(21);
head->next->down->next->next = newNode(10);
    head->next->down->next->next->down = newNode(11);
head->next->down->next->next->next = newNode(12);
head = flattenList(head);
 printFlattenNodes(head);
 return 0;
}
```

output


"C:\Users\kp\Desktop\Data Structures ODD2020\Programs on DS\ListofLL2.exe"

```
1 2 7 9 14 15 23 24 8 16 17 18 19 20 21 10 11 12 3 4
Process returned 0 (0x0)   execution time : 0.019 s
Press any key to continue.
```

Nested list using STL

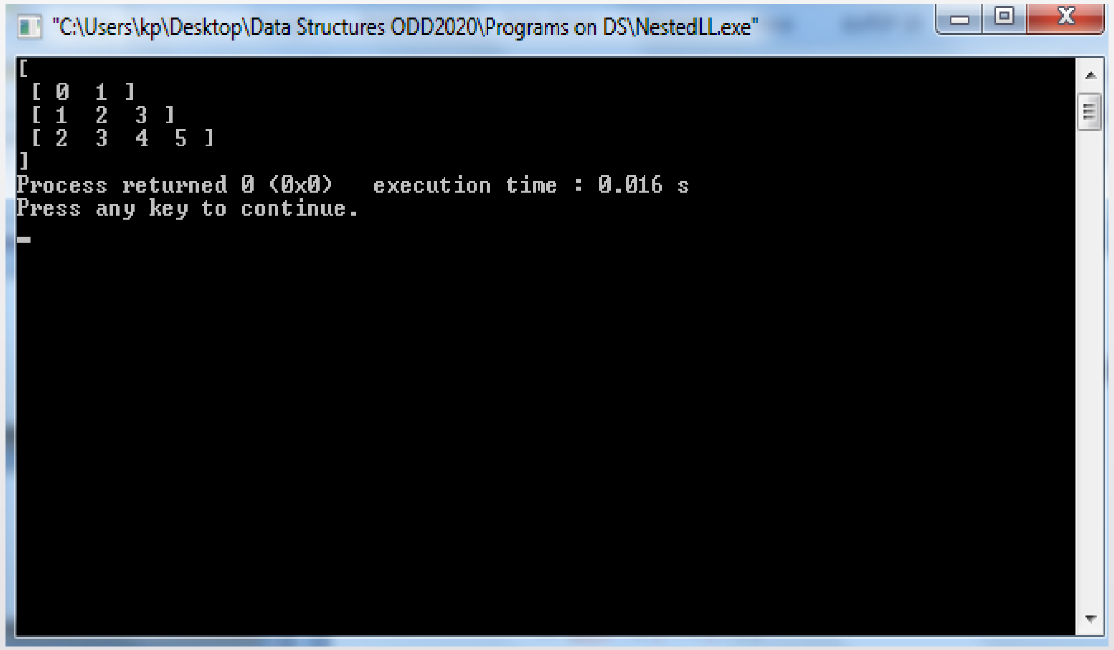
- list in STL is used to represent a linked list in C++.
- We are given n lists, we need to create a list of n lists.

- Input : Number of lists: 2
- 1st list: {1 2}
- 2nd list: {3 4 5 6 }
- Output :
- [[1 2] [3 4 5 6]]
- Input : Number of lists: 3
- 1st list : {0 1}
- 2nd list : {1 2 3}
- 3rd list : {2 3 4 5}
- Output : [[0 1] [1 2 3] [2 3 4 5]]

```
#include <iostream>
#include <iterator>
#include <list>
using namespace std;

void printNestedList(list<list<int> > nested_list)
{
  cout << “[\n";

  list<list<int> >::iterator nested_list_itr;

  for (nested_list_itr = nested_list.begin();
       nested_list_itr != nested_list.end();
       ++nested_list_itr)
{

      cout << "  [";
```

```
      list<int>::iterator single_list_itr;

    list<int> single_list_pointer = *nested_list_itr;

    for (single_list_itr = single_list_pointer.begin();
           single_list_itr != single_list_pointer.end();
           single_list_itr++)
     {
          cout << " " << *single_list_itr << " ";
     }
     cout << "]\n";
  }
  cout << "]";
}
```

```
int main()
{
    list<list<int> > nested_list;
    list<int> single_list;
    int n, m, num;
    n = 3;
    for (int i = 0; i < n; i++) {

            // number of elements in list
            m = i + 2;
            for (int j = 0; j < m; j++) {
                    num = i + j;
                    single_list.push_back(num);
            }
            nested_list.push_back(single_list);
    single_list.erase(single_list.begin(),

            single_list.end());
    }
    printNestedList(nested_list);

    return 0;
}
```



"C:\Users\kp\Desktop\Data Structures ODD2020\Programs on DS\NestedLL.exe"

```
[
 [ 0  1 ]
 [ 1  2  3 ]
 [ 2  3  4  5 ]
]
Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```