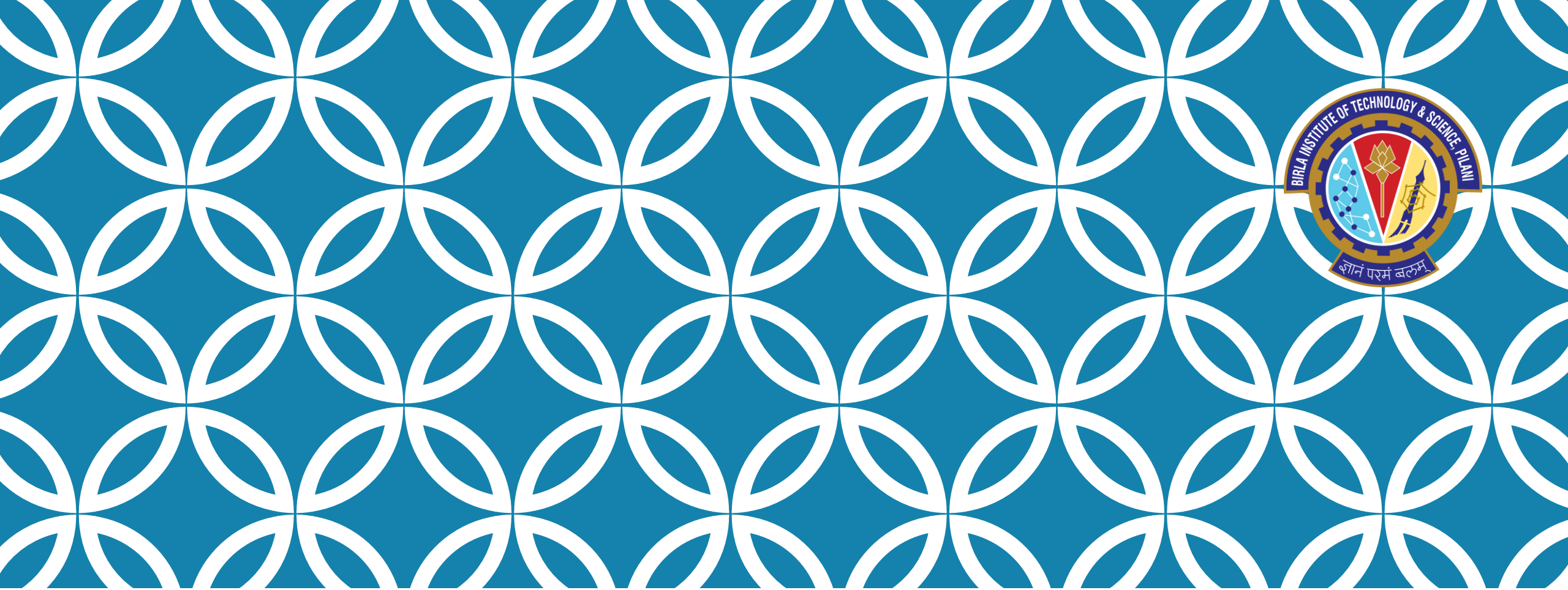
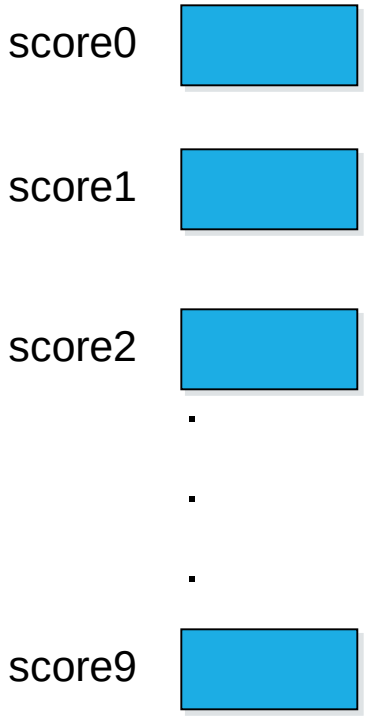


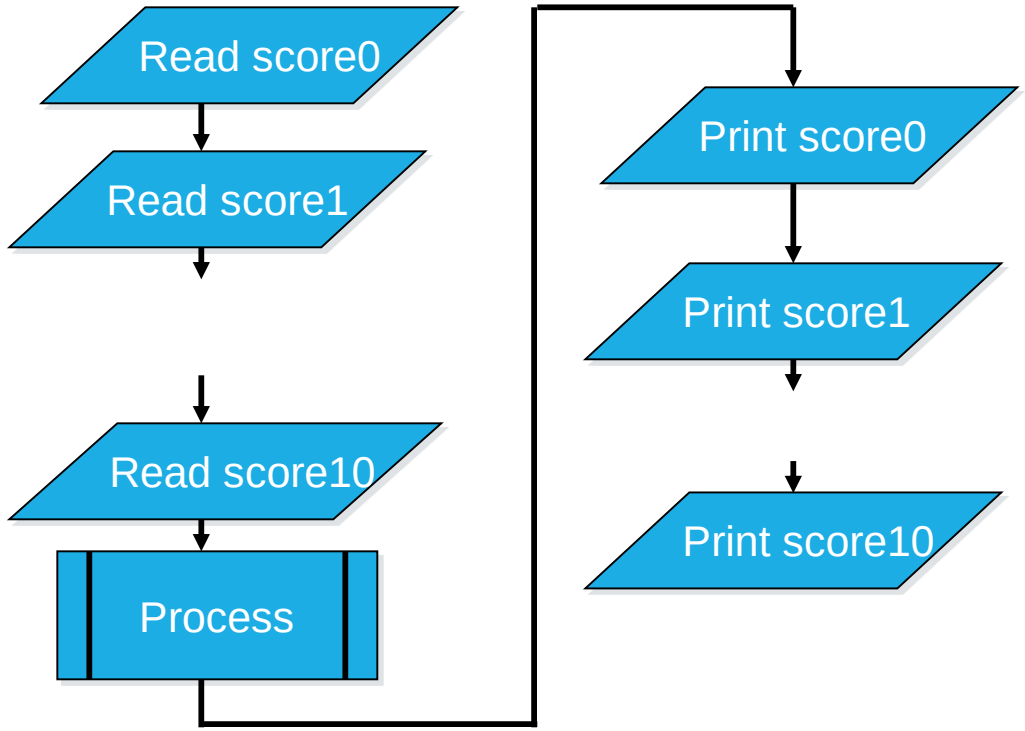
CS F211: DATA STRUCTURES & ALGORITHMS (2ND SEMESTER 2024-25) DYNAMIC ARRAYS & LINKED LISTS

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

ARRAYS: WHY?

Let us assume that we have to read, process and print 10 numbers (integers) and keep those in the memory throughout the execution.

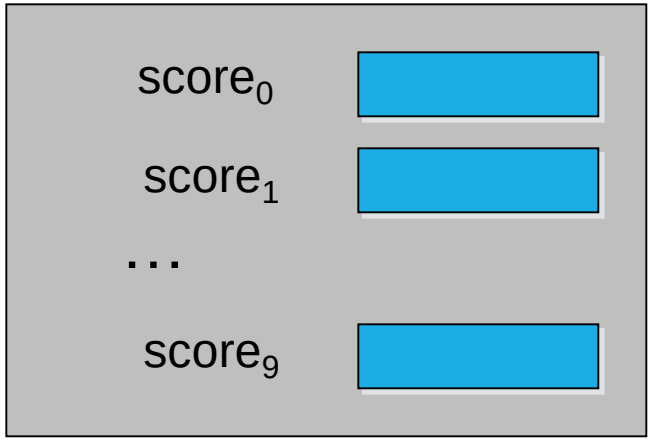




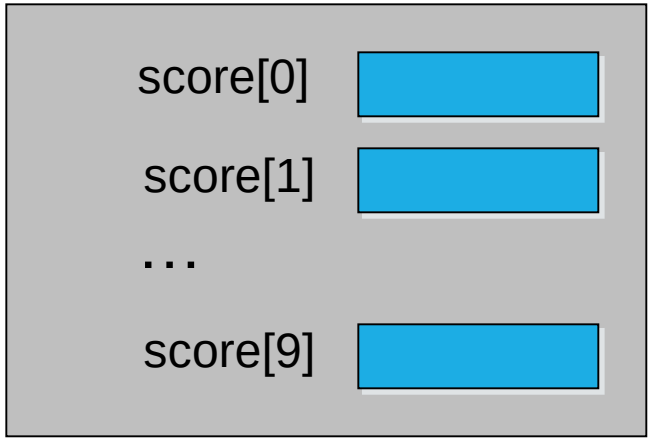
Definitely, not a good idea for 10000 numbers.

CONTINUED…

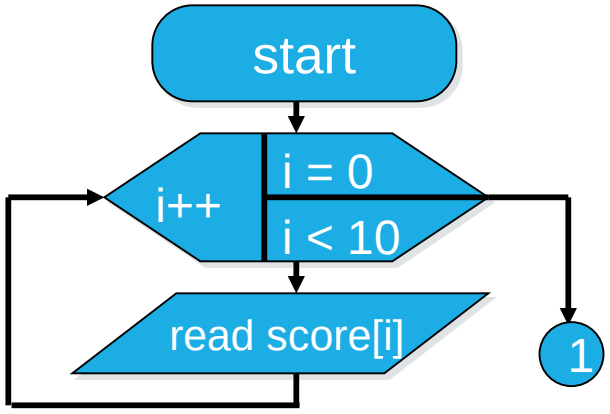
An array of scores: Only one variable is enough.

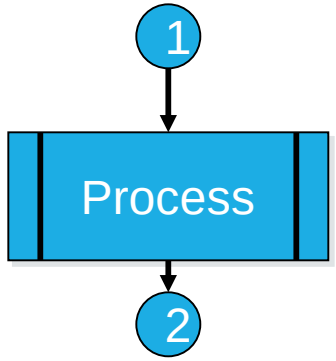


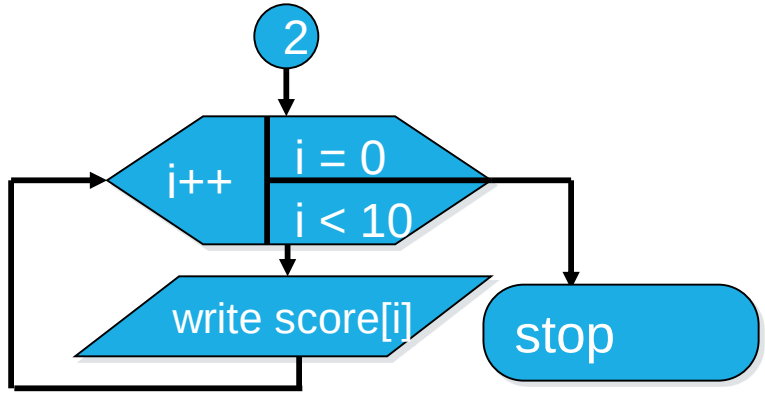
Subscripting



Indexing







DYNAMIC ARRAYS

- What are they?

Applications of arrays: Maths (vectors, matrices, polynomials,…), databases, compilers (control flow), dynamic memory allocations etc.

DYNAMIC ARRAYS EXAMPLE: LAB3

- Let us understand the operations needed to implement a dynamic array: insert, remove etc.

```
244     arr.insertItem(5);
245     arr.insertItem(3);
246     arr.insertItem(11);
247
248     arr.display();
```

```
249
250     arr.insertItemAtIndex(1, 7);
251
252     arr.display();
253
254     arr.sort();
255
256     arr.display();
257
```

```
258     arr.insertItem(15);
259     arr.insertItem(16);
260
261     arr.display();
262
263     cout << arr.getSize() << endl;
264
```

```
265     arr.deleteItem(11);
266
267     arr.display();
268
269     arr.deleteItem(16);
270
271     arr.display();
272
```

(DynamicArray.cpp given in the next week's lab sheet)

```
void Dynamic1DArray :: shrink() {

    capacity >>= 1;

    int *newArr = new int[capacity];

    for (int i = 0; i < size; i++)
        newArr[i] = arr[i];

    // update the global array pointer
    arr = newArr;
}
```

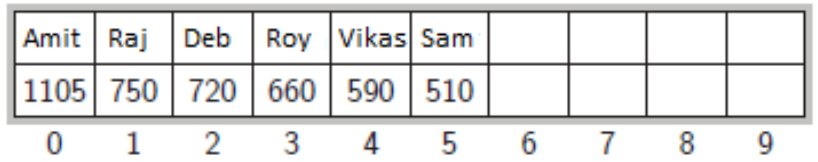
(shrink)

```
273     arr.deleteItemFromIndex(0);
274
275     arr.display();
276
277     arr.deleteItemFromIndex(1);
278
279     arr.display();
280
281     cout << arr.getSize() << endl;
```

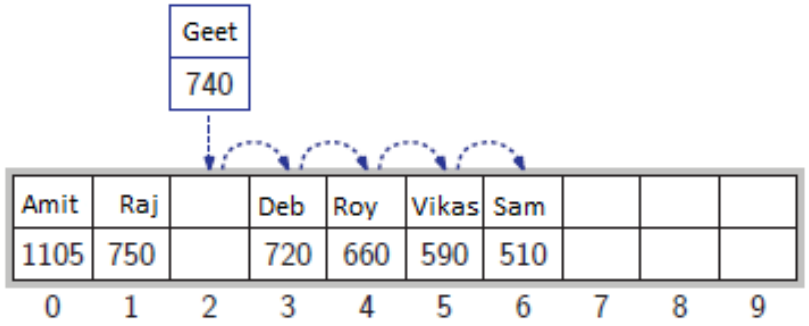
```
5 3 11
5 7 3 11
3 5 7 11
3 5 7 11 15 16
6
3 5 7 15 16
3 5 7 15
5 7 15
5 15
2
```

(Output)

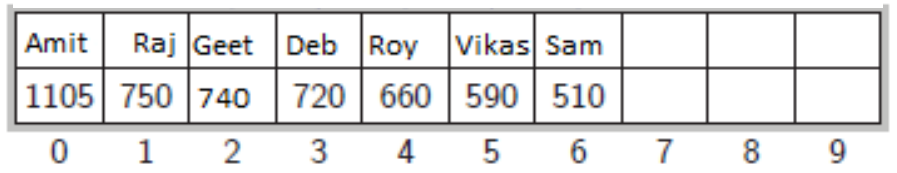
USING ARRAYS: AN EXAMPLE



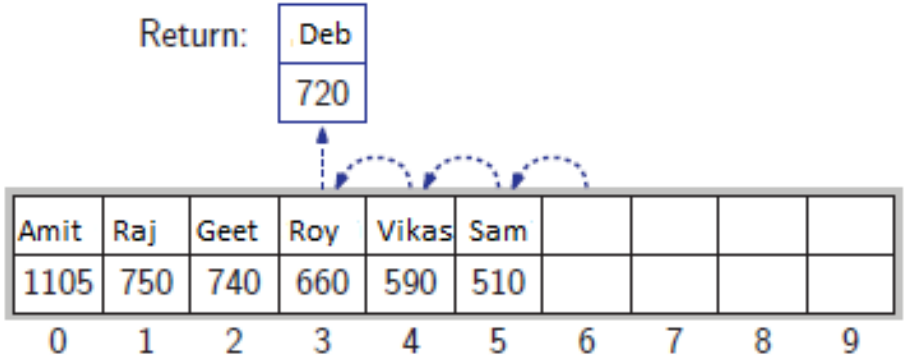
{An entries array of length 10 with 6 GameEntry objects (maxExntries: 10, numEntries: 6)}



{Preparing to add a new GameEntry object by shifting all the erntries with smaller scores to the right by one position}



{Copying the new entry into the position. Scenario after addition}



{Removing an element at index i requires moving all the entries at indices higher than i one position to the left}

IMPLEMENTATION: STORING GAME ENTRIES

```
class GameEntry {
public:
        GameEntry ( const string &n = "", int s = 0);
        string getName() const;
        int getScore() const;
private:
        string name;
        int score;
};
```

(A Class representing a Game entry)

```
GameEntry::GameEntry(const string &n, int s) : name(n),
score(s) { }
string GameEntry::getName() const { return name; }
int GameEntry::getScore() const { return score; }
```

(Constructor and member functions)

```
class Scores {
public:
        Scores(int maxEnt = 10);
        ~Scores();
        void add(const GameEntry &e);
        GameEntry remove(int i) ;
        void printAllScores();
private:
        int maxEntries; //maximum number of entries
        int numEntries;  //actual number of entries
        GameEntry *entries;
};
```

(A Class for storing Game scores)

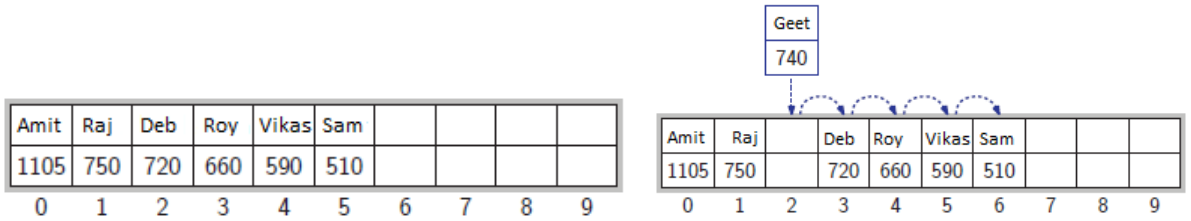
```
Scores::Scores(int maxEnt) {
    maxEntries = maxEnt; // save the max size
    entries = new GameEntry[maxEntries];
    numEntries = 0;
}
```

```
Scores::~Scores() {  delete[ ] entries; }
```

INSERTING INTO AND DELETING FROM ARRAY

```
void Scores::add(const GameEntry &e) {
    int newScore = e.getScore(); // score to add
    if (numEntries == maxEntries) { // the array is full
       if (newScore <= entries[maxEntries - 1].getScore())
            return; // not high enough - ignore
    }
    else  numEntries++; // if not full, one more entry

    int i = numEntries - 2; // start with the next to last
    while (i >= 0 && newScore > entries[i].getScore() ) {
         entries[i + 1] = entries[i]; // shift right if smaller
         i--;
     }
    entries[i + 1] = e; // put e in the empty spot
}
```

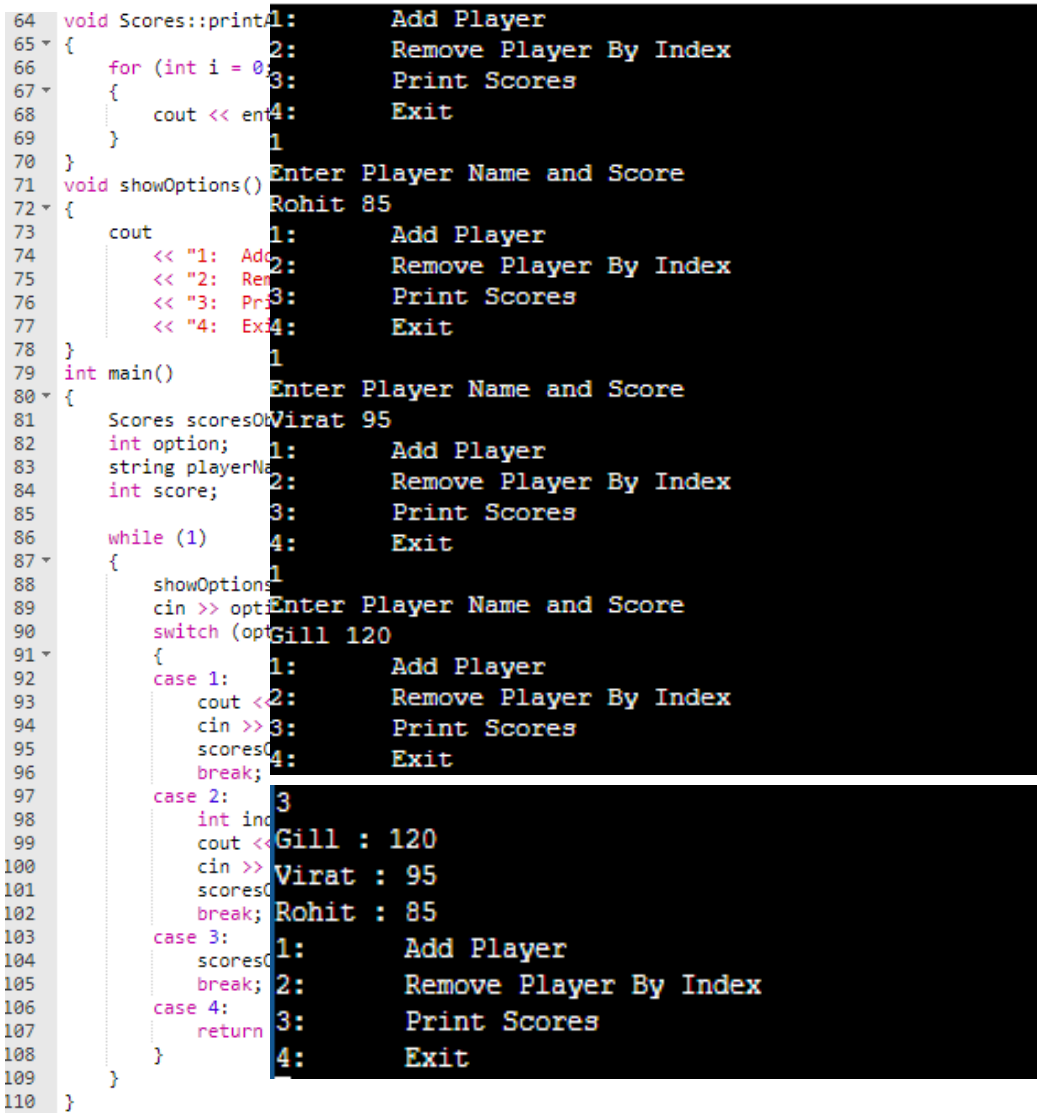
(Inserting a Game entry object)




```
GameEntry Scores::remove(int i)
{
      if ((i < 0) || (i >= numEntries)) // invalid index
            throw("IndexOutOfBounds - Invalid index");
      GameEntry e = entries[i]; // save the removed object
      for (int j = i + 1; j < numEntries; j++)
            entries[j - 1] = entries[j]; // shift entries left
      numEntries--;    // one fewer entry
      return e;        // return the removed object
}
```

(Removing a Game entry object)

DRIVER AND OTHER CLASSES FOR GAME ENTRY EX.



```
void Scores::print
{
    for (int i = 0;
    {
        cout << ent
    }
}
void showOptions()
{
    cout
        << "1:  Ad
        << "2:  Re
        << "3:  Pr
        << "4:  Ex
}
int main()
{
    Scores scoresO
    int option;
    string playerNa
    int score;

    while (1)
    {
        showOptions
        cin >> opti
        switch (opt
        {
        case 1:
            cout <<
            cin >>
            scores
            break;
        case 2:
            int ind
            cout <<
            cin >>
            scores
            break;
        case 3:
            scores
            break;
        case 4:
            return
        }
    }
}
```

```
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
1
Enter Player Name and Score
Rohit 85
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
1
Enter Player Name and Score
Virat 95
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
1
Enter Player Name and Score
Gill 120
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
```

```
3
Gill : 120
Virat : 95
Rohit : 85
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
```

```
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
1
Enter Player Name and Score
Gill 200
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
3
Gill : 200
Gill : 120
Virat : 95
Rohit : 85
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
```

(Lab3: GameEntry.cpp)

LAB3 TASKS: GAME ENTRY

```
4
Gill : 2
Virat : 1
Rohit : 1
1:  Add Player
2:  Remove Player By Index
3:  Print Scores
4:  Print Players Count
5:  Exit
```

(How many number of entries are there for each player? Option 4)

```
Enter max value and min value of the score range
400 300
Gill : 320
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Print Players Count
5:      Print Unique Scores
6:      Print Players in Score Range
7:      Print Master Player
8:      Exit
```

(Display players in a score range: Option 6)

```
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
1
Enter Player Name and Score
Rohit 85
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
1
Enter Player Name and Score
Virat 95
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
1
Enter Player Name and Score
Gill 120
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
1
Enter Player Name and Score
Gill 200
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
3
Gill : 200
Virat : 95
Rohit : 85
1:      Add Player
2:      Remove Player By Index
3:      Print Scores
4:      Exit
```

(display unique entries for each player?)

(GameEntry_Unique.cpp)

SORTING & SEARCHING IN AN ARRAY

```
void Dynamic1DArray ::sort()
{
    for (int j = 1; j < size; j++)
    {
        int key = arr[j];
        int i = j - 1;
        while (i > -1 && arr[i]>key)
        {
            arr[i + 1] = arr[i];
            i = i - 1;
        }
        arr[i + 1] = key;
    }
}
```

(Insertion Sort)

```
int Dynamic1DArray
::binarySearch(const int item)
{
    int low = 0, high = size - 1;
    while (low <= high){
       int mid = low + ((high -
                        low) >> 1);
        if (item == arr[mid])
            return mid;
        if (item < arr[mid])
            high = mid - 1;
        else
            low = mid + 1;
    }
    return -1; }
```

(Binary Search)

More sorting & searching algos later…

MULTI-DIMENSIONAL ARRAYS

Month

Year

	0	1	2	3	4	5	6	7	8	9	10	11
0	30	40	75	95	130	220	210	185	135	80	40	45
1	25	25	80	75	115	270	200	165	85	5	10	16
2	35	45	90	80	100	205	135	140	170	75	60	95
3	30	40	70	70	90	180	180	210	145	35	85	80
4	30	35	40	90	150	230	305	295	60	95	80	30

Average Yearly Rainfall (in mm of Hyd)

Arrays in C++ are one-dimensional. However, we can define a 2D array as "an array of arrays".

3-dimensional

	0	1	2	3	4	5	6	7	8	9	10	11
0	20	60	75	95	130	220	210	185	135	80	40	45
1	29											
2	35											
3	30											
4	30											

Hyd

	0	1	2	3	4	5	6	7	8	9	10	11
0	10	20	35	95	130	220	210	185	135	80	40	45
1	5											
2	35											
3	30											
4	30											

Delhi

	0	1	2	3	4	5	6	7	8	9	10	11
0	10	20	35	95	130	220	210	185	135	80	40	45
1	5	17	9	8	115	270	200	165	85	5	10	16
2	35	45	90	80	100	205	135	140	170	75	60	95
3	30	40	70	70	90	180	180	210	145	35	85	80
4	30	35	40	90	150	230	305	295	60	95	80	30

Goa

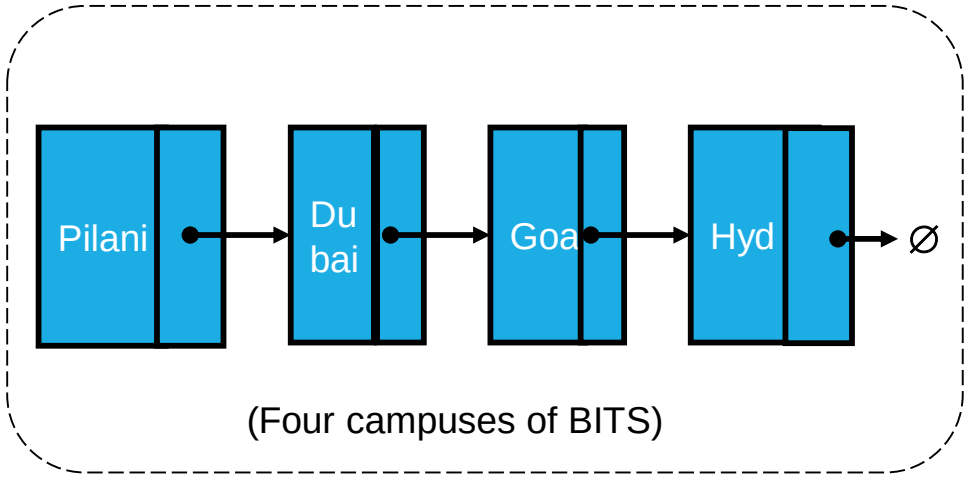
```cpp
#include<iostream>
using namespace std;
int main()
{
    int x[5][12]={{30,40,75,95,130,220,210,185,135,80,40,45},
        {25,25,80,75,115,270,200,165, 85, 5,10, 16},
        {35,45,90,80,100,205,135,140,170,75,60,95},
        {30,40,70,70, 90,180,180,210,145,35,85,80},
        {30,35,40,90,150,230,305,295, 60,95,80,30}
};
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 12; j++)
        {
            cout << "Element at x[" << i
                 << "][" << j << "]: ";
            cout << x[i][j]<<endl;
        }
    }

    return 0;
}
```

```
Element at x[0][0]: 30
Element at x[0][1]: 40
Element at x[0][2]: 75
Element at x[0][3]: 95
Element at x[0][4]: 130
Element at x[0][5]: 220
Element at x[0][6]: 210
Element at x[0][7]: 185
Element at x[0][8]: 135
Element at x[0][9]: 80
Element at x[0][10]: 40
Element at x[0][11]: 45
Element at x[1][0]: 25
Element at x[1][1]: 25
Element at x[1][2]: 80
Element at x[1][3]: 75
Element at x[1][4]: 115
Element at x[1][5]: 270
Element at x[1][6]: 200
Element at x[1][7]: 165
Element at x[1][8]: 85
Element at x[1][9]: 5
Element at x[1][10]: 10
Element at x[1][11]: 16
Element at x[2][0]: 35
Element at x[2][1]: 45
Element at x[2][2]: 90
Element at x[2][3]: 80
Element at x[2][4]: 100
Element at x[2][5]: 205
Element at x[2][6]: 135
Element at x[2][7]: 140
Element at x[2][8]: 170
Element at x[2][9]: 75
Element at x[2][10]: 60
Element at x[2][11]: 95
Element at x[3][0]: 30
Element at x[3][1]: 40
Element at x[3][2]: 70
Element at x[3][3]: 70
Element at x[3][4]: 90
Element at x[3][5]: 180
Element at x[3][6]: 180
Element at x[3][7]: 210
Element at x[3][8]: 145
Element at x[3][9]: 35
Element at x[3][10]: 85
Element at x[3][11]: 80
Element at x[4][0]: 30
Element at x[4][1]: 35
Element at x[4][2]: 40
Element at x[4][3]: 90
Element at x[4][4]: 150
Element at x[4][5]: 230
Element at x[4][6]: 305
Element at x[4][7]: 295
Element at x[4][8]: 60
Element at x[4][9]: 95
Element at x[4][10]: 80
Element at x[4][11]: 30
```

SINGLY LINKED LISTS

- Linked list: A linear data structure?
- A singly linked list is a concrete data structure consisting of a sequence of nodes, where each node has?

Arrays	Vs.	Linked lists
1. Arrays are stored in contiguous location.		1. Linked lists are not stored in contiguous location.
2. Fixed in size.		2. Dynamic in size.
3. Memory is allocated at compile time.		3. Memory is allocated at run time.
4. Uses less memory than linked lists.		4. Uses more memory because it stores both data and the address of next node.
5. Elements can be accessed easily.		5. Element accessing requires the traversal of whole linked list.
6. Insertion and deletion operation takes time.		6. Insertion and deletion operation is faster.



How will you store mid-sem scores of say, 4 students in a linked list?

Applications: Implementation of Stacks, queues, Graphs (Adj list), Keep track of OS process states, Free memory blocks

IMPLEMENTING A SINGLY LINKED LIST

Step 1: Define a class for the Node

```
class StringNode {
   private:  string elem;
             StringNode* next;
   friend class StringLinkedList;
};
```

Step 2: Define a class for the Linked list

```
class StringLinkedList {
   public: StringLinkedList();
           ~StringLinkedList();
   bool empty() const;
   const string& front() const;
   void addFront(const string& e);
   void removeFront();
   private: StringNode* head;
};
```

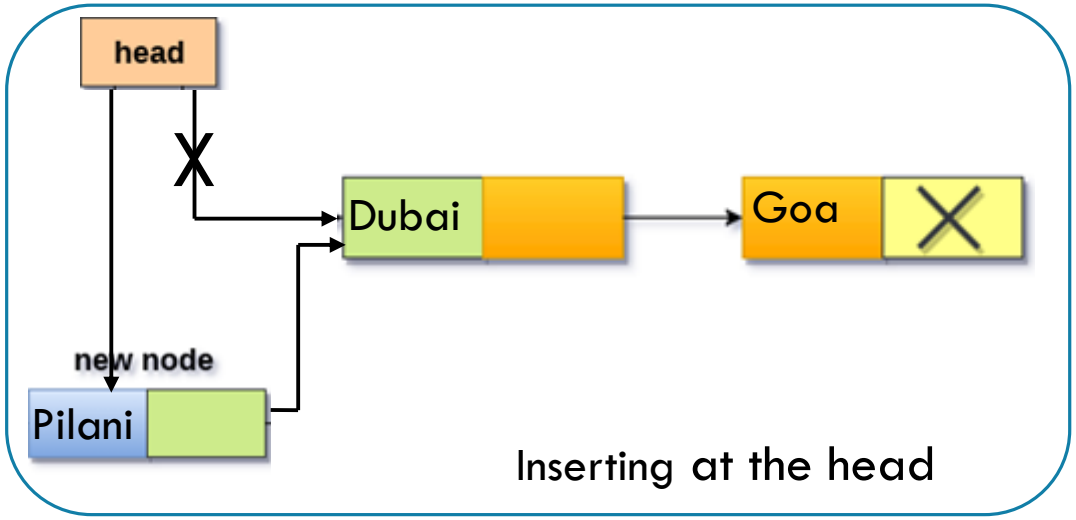
Step 3: Define a set of member functions for the Linked list class defined in Step 2

```
StringLinkedList::StringLinkedList()  :  head(???){  }
StringLinkedList::~StringLinkedList() {
   while(!empty())
        ???;
}

bool StringLinkedList::empty() const {  //Is list empty?
    return head == NULL;
}

const string& StringLinkedList::front() const {
return ???;
}
```

INSERTING & REMOVING AT THE HEAD OF LINKED LIST

1. Create a new node
2. Store data into this node
3. Have new node point to old head
4. Update head to point to new node

```
void StringLinkedList::addFront(const string& e)
{
   StringNode* v = new StringNode;
   v->elem = e;
   v->next = head;
   head = v;
}
```



Inserting at the head

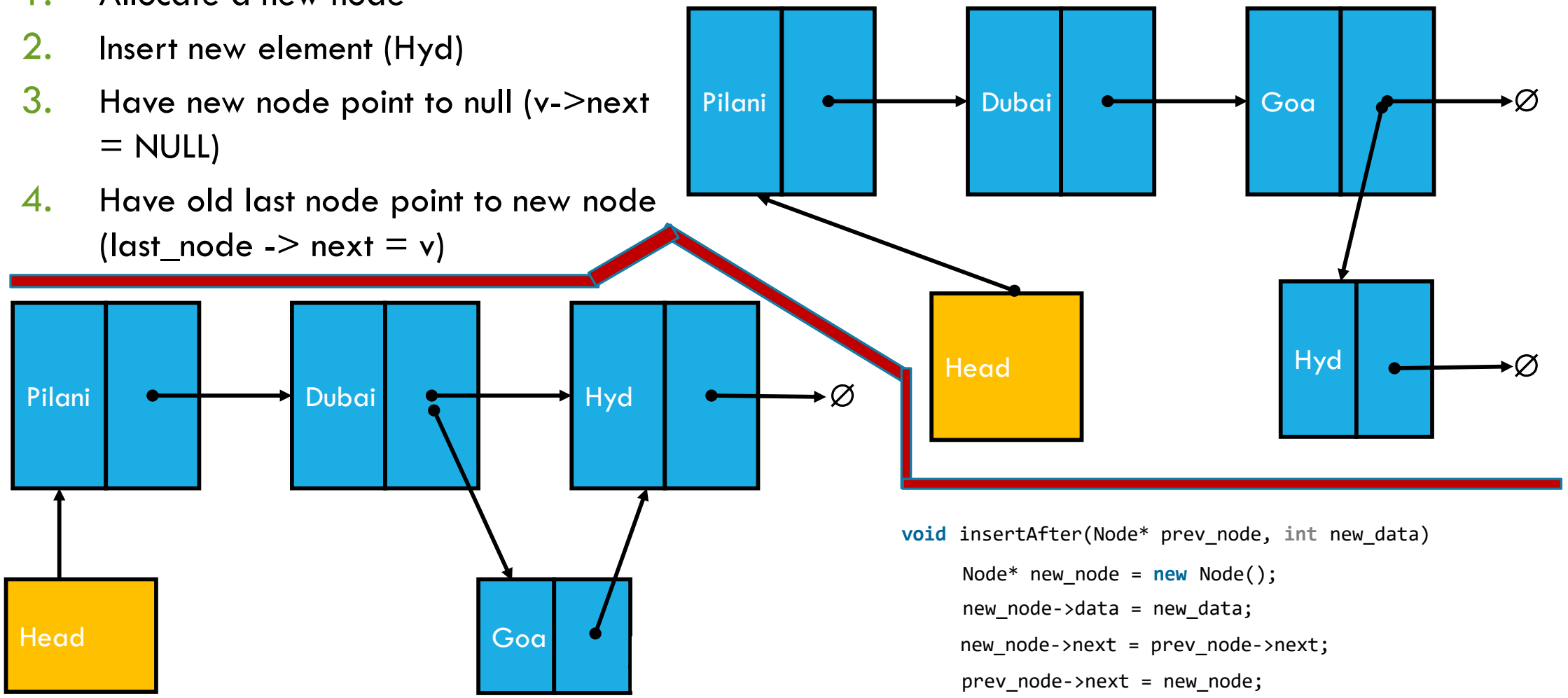
```
void StringLinkedList::removeFront()
{
  StringNode* old = head;
  head = old->next;
  delete old;
}
```

Deleting at the head

1. Save old head
2. Advance head to the next node
3. Delete the old head node

INSERTING AT THE TAIL & INSIDE A LINKED LIST

1. Allocate a new node
2. Insert new element (Hyd)
3. Have new node point to null (v->next = NULL)
4. Have old last node point to new node (last_node -> next = v)




```
void insertAfter(Node* prev_node, int new_data)
    Node* new_node = new Node();
    new_node->data = new_data;
    new_node->next = prev_node->next;
    prev_node->next = new_node;
```

DELETING THE LAST NODE

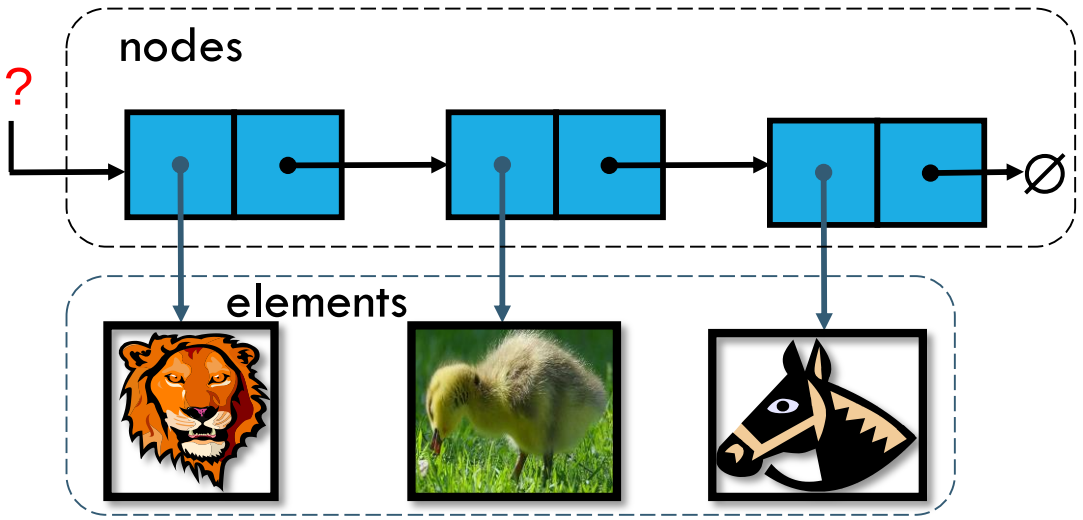
Algorithm:

1. If (headNode == null) //how many nodes in list?
   then what should you do?
2. If (headNode.next == null) //how many nodes in list?
   then what should you do?
3. While secondLast.next.next != null //traverse till secondLast
   secondLast = secondLast.nextNode
4. Delete last node and set the pointer of secondLast to null.

```
After deleting the last node:
Pilani Dubai Goa
```

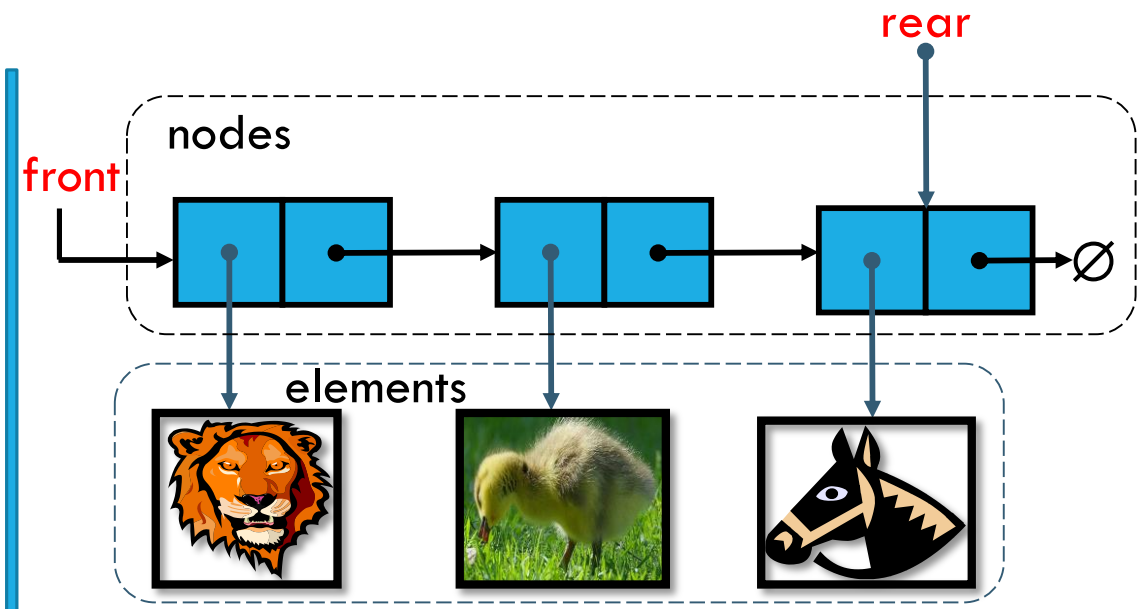
```
#include <iostream>
using namespace std;
struct Node {
    string data;
    struct Node* next;
};
Node* removeLastNode(struct Node* head) {
    if (head == NULL)
        return NULL;
    if (head->next == NULL) {
        delete head;
        return NULL;
    }
    Node* second_last = head;
    while (second_last->next->next != NULL)
        second_last = second_last->next;
    delete (second_last->next);
    second_last->next = NULL;
    return head;
}
void insertNode (struct Node** head_ref, string new_data) {
    struct Node* new_node = new Node;
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}
int main() {
    Node* head = NULL;
    insertNode(&head, "Hyd");
    insertNode(&head, "Goa");
    insertNode(&head, "Dubai");
    insertNode(&head, "Pilani");
    head = removeLastNode(head);
    cout << "After deleting the last node:"<<endl;
    for (Node* temp = head; temp != NULL; temp = temp->next)
        cout << temp->data << " ";
    return 0;
}
```

STACK & QUEUE AS SINGLY LINKED LISTS



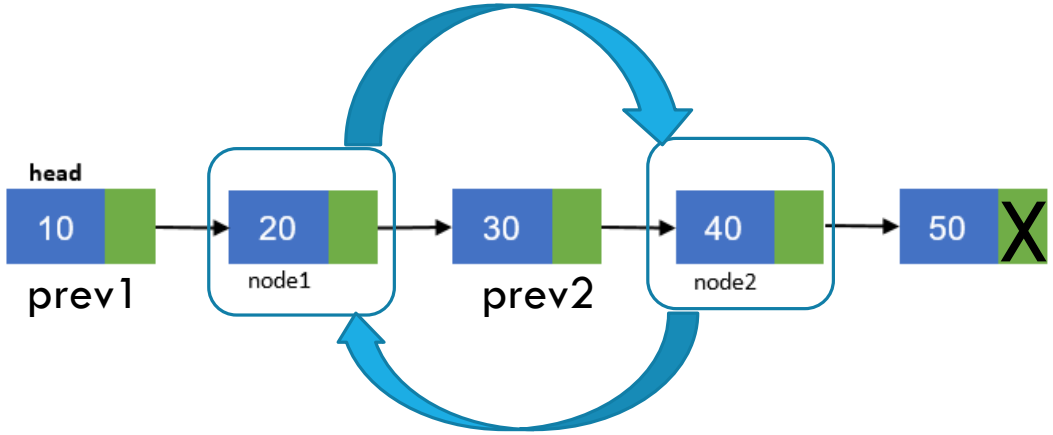
Stack: We can implement stack as a linked list. How will you implement?

Implementation in later chapters…

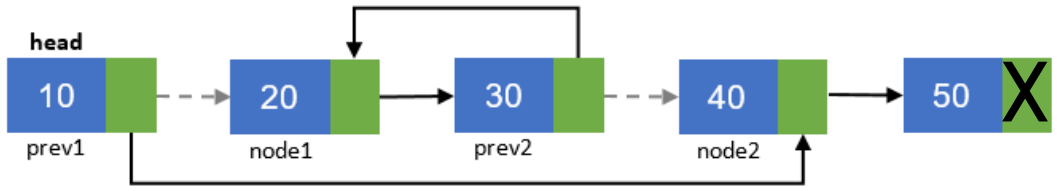


Queue: We can implement a queue as a linked list. Front element is stored as first element of the linked list, and rear element is stored as the last element.

SWAPPING TWO NODES IN A LINKED LIST



?



Lab 4 next week (week no:4)

GENERIC SINGLY LINKED LISTS: USING TEMPLATES

```
1  #include <iostream>
2
3  using namespace std;
4
5  template<typename E>
6  class SLinkedList;
7
8  template <typename E>
9  class SNode {
10 private:
11     E elem;
12     SNode<E>* next;
13     friend class SLinkedList<E>;
14 };
15
16 template <typename E>
17 class SLinkedList {
18 public:
19     SLinkedList();
20     ~SLinkedList();
21     bool empty() const;
22     const E& front() const;
23     void addFront(const E& e);
24     void removeFront();
25     void traverse();
26 private:
27     SNode<E>* head;
28 };
29
30 template <typename E>
31 SLinkedList<E>::SLinkedList()
32     : head(NULL) { }
```

```
+----------------------------------------+
Please enter one of the following choices:
1 : Add at the front
2 : Get frontmost element
3 : Remove front element
4 : Check if list is empty
5 : Traverse the list
6 : Search for an element
7 : Swap two nodes
8 : Exit
1
Enter the element: Rohit
+----------------------------------------+
Please enter one of the following choices:
1 : Add at the front
2 : Get frontmost element
3 : Remove front element
4 : Check if list is empty
5 : Traverse the list
6 : Search for an element
7 : Swap two nodes
8 : Exit
1
Enter the element: Virat
+----------------------------------------+
Please enter one of the following choices:
1 : Add at the front
2 : Get frontmost element
3 : Remove front element
4 : Check if list is empty
5 : Traverse the list
6 : Search for an element
7 : Swap two nodes
8 : Exit
2
Frontmost element is : Virat
+----------------------------------------+
Please enter one of the following choices:
1 : Add at the front
2 : Get frontmost element
3 : Remove front element
4 : Check if list is empty
5 : Traverse the list
6 : Search for an element
7 : Swap two nodes
8 : Exit
```

```
33
34 template <typename E>
35 bool SLinkedList<E>::empty() const      // is list empty?
36 { return head == NULL; }
37
38 template <typename E>
39 const E& SLinkedList<E>::front() const  // return front element
40 { return head->elem; }
41
42 template <typename E>
43 SLinkedList<E>::~SLinkedList()          // destructor
44 { while (!empty()) removeFront(); }
45
46 template <typename E>
47 void SLinkedList<E>::addFront(const E& e) {     // add to front of list
48     SNode<E>* v = new SNode<E>;                 // create new node
49     v->elem = e;                                // store data
50     v->next = head;                             // head now follows v
51     head = v;                                   // v is now the head
52 }
53
54 template <typename E>
55 void SLinkedList<E>::removeFront() {            // remove front item
56     SNode<E>* old = head;                       // save current head
57     head = old->next;                           // skip over old head
58     delete old;                                 // delete the old head
59 }
60
```

```
SNode<E>
*SLinkedList<E>::search(const E &e){
//complete code here
}
```

(Next week Lab 4)

REVERSING A LINKED LIST

```
void listReverse(LinkedList& L) {
        LinkedList T;
        while (!L.empty()) {
            string s = L.front();
            L.removeFront();
            T.addFront(s);
        }
        while (!T.empty()) {
            string s = T.front();
            T.removeFront();
            L.addBack(s);
        }
}
```

Is it In-place reversal?

```
void reverse() {
        if (head == nullptr || head->next == nullptr) {
            return;
        }
        Node* prev = nullptr;
        Node* current = head;
        Node* next;
        while (current != nullptr) {
            next = current->next;
            current->next = prev;
            prev = current;
            current = next;
        }

        head = prev;
    }
```

Is it In-place reversal?

DOUBLY LINKED LIST

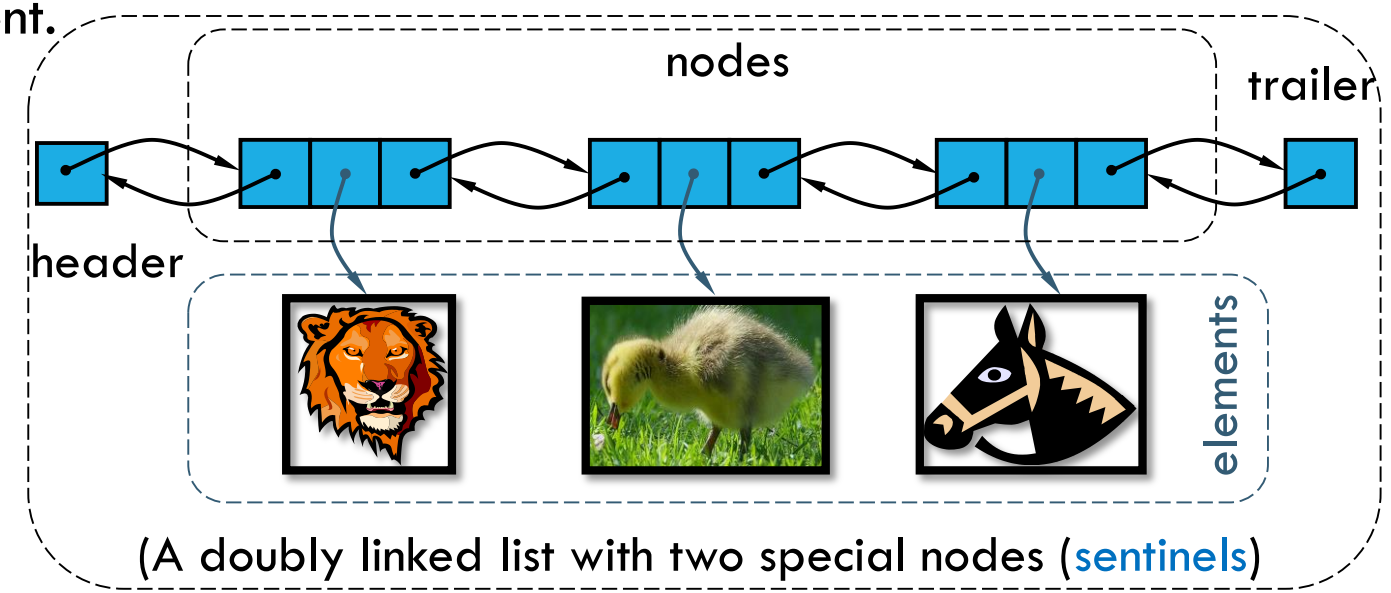
- Deleting the last node in a singly linked list is not efficient. Why? (rather any node other than first one or two)
- What is a doubly linked list?
- Insertions and deletions are more efficient.

```
typedef string Elem;
class DNode {
  private: Elem elem;
           DNode* prev;
           DNode* next;
           friend class DLinkedList;
};
```

(Implementation of DLL Node)

Applications:

- Used by browsers for what functionality?
- Used to implement MRU, and LRU caches?
- Undo/ Redo functionality in Word.
- Used to implement hash tables, stacks, binary tree etc.

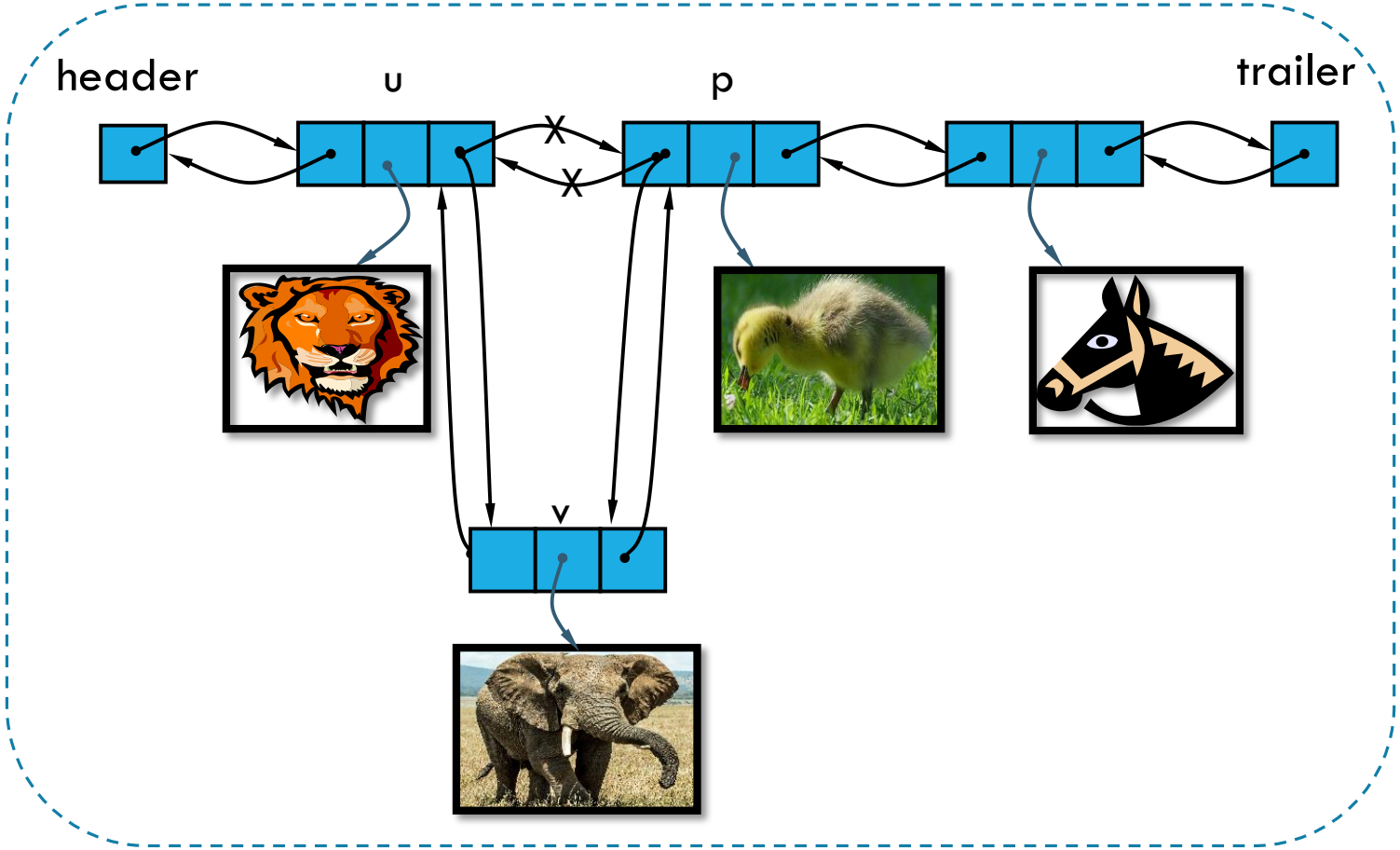


(A doubly linked list with two special nodes (sentinels)

INSERTING INTO DOUBLY-LINKED LIST

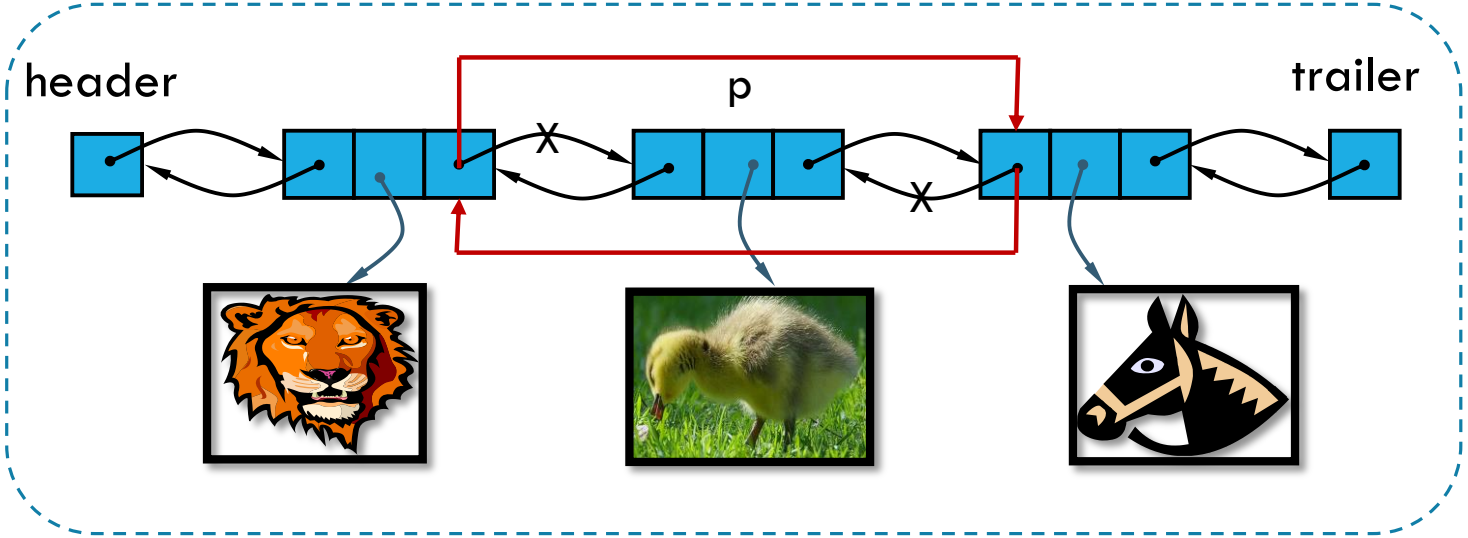
Algorithm insert(p, e): //insert e before p

Let us write the pseudo code in parallel…



REMOVING A NODE IN DOUBLY-LINKED LIST

```
Algorithm remove (p: position ) {
    if (p->previous != nil) // not first
        p->previous->next = ???;
    if (p->next != nil) //not the last
        p->next->previous  = ???;
}
```



FINDING MIDDLE NODE AND LOOP IN A LINKED LIST

```
Node* findMiddle(Node* head) {
   if (head == null) {
      return ???;
   }
   Node* slow = head;
   Node* fast = head;
   while (fast != null && fast->next != null) {
      //what will you do here?
   }
   return slow;
}
```

Floyd's Tortoise and Hare algorithm (Lab 4)

How about detecting a closed loop using this algo?

```
if (slow == fast) {
   return true;
}
```

How about removing a closed loop using this algo?

```
// start node of loop
slow = head;
while (slow != fast) {
   slow = slow->next;
   fast = fast->next;
}
```

```
// node before loop start
Node* prev = slow;
while (prev->next != fast) {
    prev = prev->next;
}
// Break the loop
   prev->next = nullptr;
```

CIRCULAR LINKED LISTS

- A circular linked list is a singly-linked list except for the last element of the list pointing to the first. Without starting over we can go back to the first.
- What is the need of cursor node?

```
class CircleList;
typedef string Elem;
class CNode {
private: Elem elem;
         CNode* next;
         friend class CircleList;
};
class CircleList {
public:   CircleList();
          ~CircleList();
           bool empty() const;
           const Elem& front() const;
           const Elem& back() const;
           void advance();
           void add(const Elem& e);
           void remove();
private:   CNode* cursor;
};
```

```
CircleList::CircleList() : cursor(NULL) { }
CircleList::~CircleList() { while (!empty()) remove(); }
bool CircleList::empty() const {return cursor== NULL; }
const Elem& CircleList::back() const {
                                    return cursor->elem; }
const Elem& CircleList::front() const {
                                  return cursor->next->elem; }
void CircleList::advance() { cursor = cursor->next; }
void CircleList::add(const Elem& e) {    //add after the cursor
   CNode* v = new CNode;
   v->elem = e;
   if (cursor == NULL) {
      v->next = v;  cursor = v; }
   else {
      v->next = cursor->next; cursor->next = v;
   }
}
```

(Lab 4: Round robin scheduling)

```
void CircleList::remove() {
  CNode* old = cursor->next;
  if (old == cursor)
     cursor = NULL;
  else
     cursor->next = old->next;
     delete old;
}
```

// remove the node after the cursor

```
Please enter one of the following choices:
1 : Add
2 : Get front element
3 : Get back element
4 : Advance cursor
5 : Remove element pointed by cursor
6 : Check if list is empty
7 : Exit
1
s1
Adding the following element : s1
1
s2
Adding the following element : s2
1
s3
Adding the following element : s3
3
Back element is : s1
4
Advancing the cursor
2
Front element is : s2
5
Removing element pointed by the cursor
6
List is not empty
```

THANK YOU!

Next Class: Algorithm Analysis…