Data Structures and Algorithms
CS-206

Linked Lists Part B

Instructor
Dr. Maria Anjum
Assistant Professor

Department of Computer Science
Lahore College for Women University

Review C++ Concepts - Structure

Structure:

A *data structure* is a group of data elements grouped together under one name. These data elements, known as *members*, can have different types and different lengths.

```
struct structure_name {
datatype member_name1;
datatype member_name2;
datatype member_name3;
.
.
}
```

Example

```
struct product {
int weight;
double price;
 } ;
product apple;
product banana, melon;
```

Review C++ Concepts – Structure Example

Example

```
struct product {
int weight;
double price;
 } ;
product apple;
product banana, melon;
```

OR

Example

```
struct product {
int weight;
double price;
 } apple; banana, melon;
```

apple.weight
apple.price
banana.weight
banana.price
melon.weight
melon.price

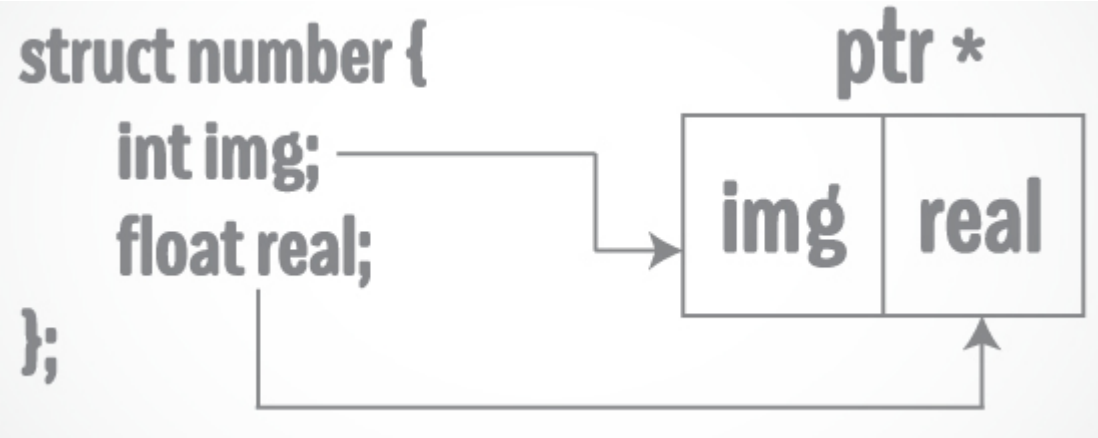
← Calling objects of a structure

Review C++ Concepts - Pointer to Structure:

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

struct temp {
    int i;
    float f;
};

int main() {
    temp *ptr;
    return 0;
}
```



Review C++ Concepts - Pointer to Structure:

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

struct Distance
{
    int feet;
    float inch;
};
```

```
int main()
{
    Distance *ptr, d;
    ptr = &d;
    cout << "Enter feet: ";
    cin >> (*ptr).feet;
    cout << "Enter inch: ";
    cin >> (*ptr).inch;
    cout << "Displaying information." << endl;
    cout << "Distance = " << (*ptr).feet << " feet " << (*ptr).inch << " inches";
    return 0;
}
```

Review C++ Concepts - Pointer to Structure:

Program Output

- Enter feet: 4
- Enter inch: 3.5
- Displaying information.
- Distance = 4 feet 3.5 inches

Review C++ Concepts - Pointer to Structure:

- ptr->feet is same as (*ptr).feet
- ptr->inch is same as (*ptr).inch

Linked List

```
struct node {
    int data;
    node* next;
};
```

```
int main() {
node* curr;      // This won't change, or we would lose the list in memory
node* temp;      // This will point to each node as it traverses the list
node* head;      // This will point to each node as it traverses the list
curr= new node;          // Sets it to actually point to something
curr->next = 0;  // Otherwise it would not work well
curr->data = 5;
```

```
temp=curr;
head= temp;
```

```
curr=new node;
curr->next=0;
curr->data =6;
temp->next=curr;
temp=temp->next;
```

```
curr=new node;
curr->next=0;
curr->data =11;
temp->next=curr;
temp=temp->next;
```

Linked List

To Traverse

```
temp = head;                    // The conductor points to the first node
if ( temp != 0 ) {              //Makes sure there is a place to start
while ( temp->next != 0)
temp = temp->next;
}
```

Linked List

To Print

```
temp=head;

if ( temp != 0 ) {          //Makes sure there is a place to start

while ( temp->next != 0 ) {
   cout<< temp->data;
   temp = temp->next;
  }
  cout<< temp->next; -→ what this will do?
```

Linked List - Class

```
class linkedList                    // linked list class
{
private:
        typedef struct node{
        int data;
        node* next;
        }* nodePtr
        nodePtr head;
        nodePtr temp;
        nodePtr curr;
public:
        linkedList(){
        head = NULL;
        curr = NULL;
        temp = NULL;
                };
    void addNode(int addData); //function to add data
    void delNode(int delData); //function to delete data
    void printList();                           //print list
    ~linkedList();
}
```

Linked List – Class:: void addNode(int addData);

```
void linkedList addNode(int addData){
        nodePtr n= new node;
        n->next=NULL;
        n->data= addData;

        if (head!=NULL){
        curr = head;
                while (curr->next!= NULL)
                {
                        curr=curr->next;
                }
        curr->next= n;
        }

        else {
                head = n;
        }

}
```

Review Singly Linked List

- Linked lists are building blocks for many other data structures like stacks and queues.
- Linked lists are a sequence of nodes containing data fields and pointers to the next node (or) next node and previous nodes based on its type.
- Linked lists permit addition/ removal of node in constant time.
- Unlike arrays the order of linked list elements need not be contiguous in memory.
- Unlike arrays there is no upper limit on the amount of memory reserved.
- A singly linked list is the simplest of linked lists.
- Each node of a singly linked list has data elements and a single link (pointer) that points to the next node of the list (or) NULL if it is the last node of the list.
- Addition/ Deletion of a node to the singly linked list involves the creation/ deletion of the node and adjusting the node pointers accordingly.

References

- https://www.programiz.com/cpp-programming/structure-pointer
- http://www.cplusplus.com/doc/tutorial/structures/
- http://www.cprogramming.com/tutorial/lesson15.html
- https://www.youtube.com/watch?v=o5wJkJJpKtM
- https://www.youtube.com/watch?v=cAZ8CyDY56s
- https://www.youtube.com/watch?v=H5lkmKkfjD0&list=PLTxllHdfUq4fewZGVFPhzbmHTQC2UH7PJ

Practical and Reading Assignment

- https://www.programiz.com/cpp-programming/structure-pointer
- http://www.cplusplus.com/doc/tutorial/structures/
- http://www.cprogramming.com/tutorial/lesson15.html
- https://www.youtube.com/watch?v=o5wJkJJpKtM
- https://www.youtube.com/watch?v=cAZ8CyDY56s
- https://www.youtube.com/watch?v=H5lkmKkfjD0&list=PLTxllHdfUq4fewZGVFPhzbmHTQC2UH7PJ

Practical and Reading Assignment

- Write run able code for

- Create new nodes
- Insert a node at beginning
- Insert a node at the end of the list
- Insert a node somewhere in the list

- Delete a note at the beginning of the list
- Delete a node at the end of the list
- Delete a node somewhere in the list

- Display linked list
- Search a number in the list