Data Structures & Algorithms

04: Linked List; Part - I

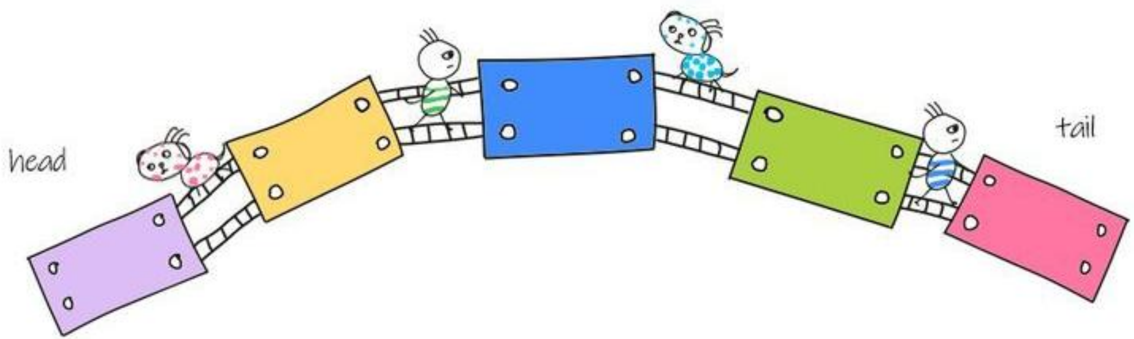
Dr Ram Prasad Krishnamoorthy

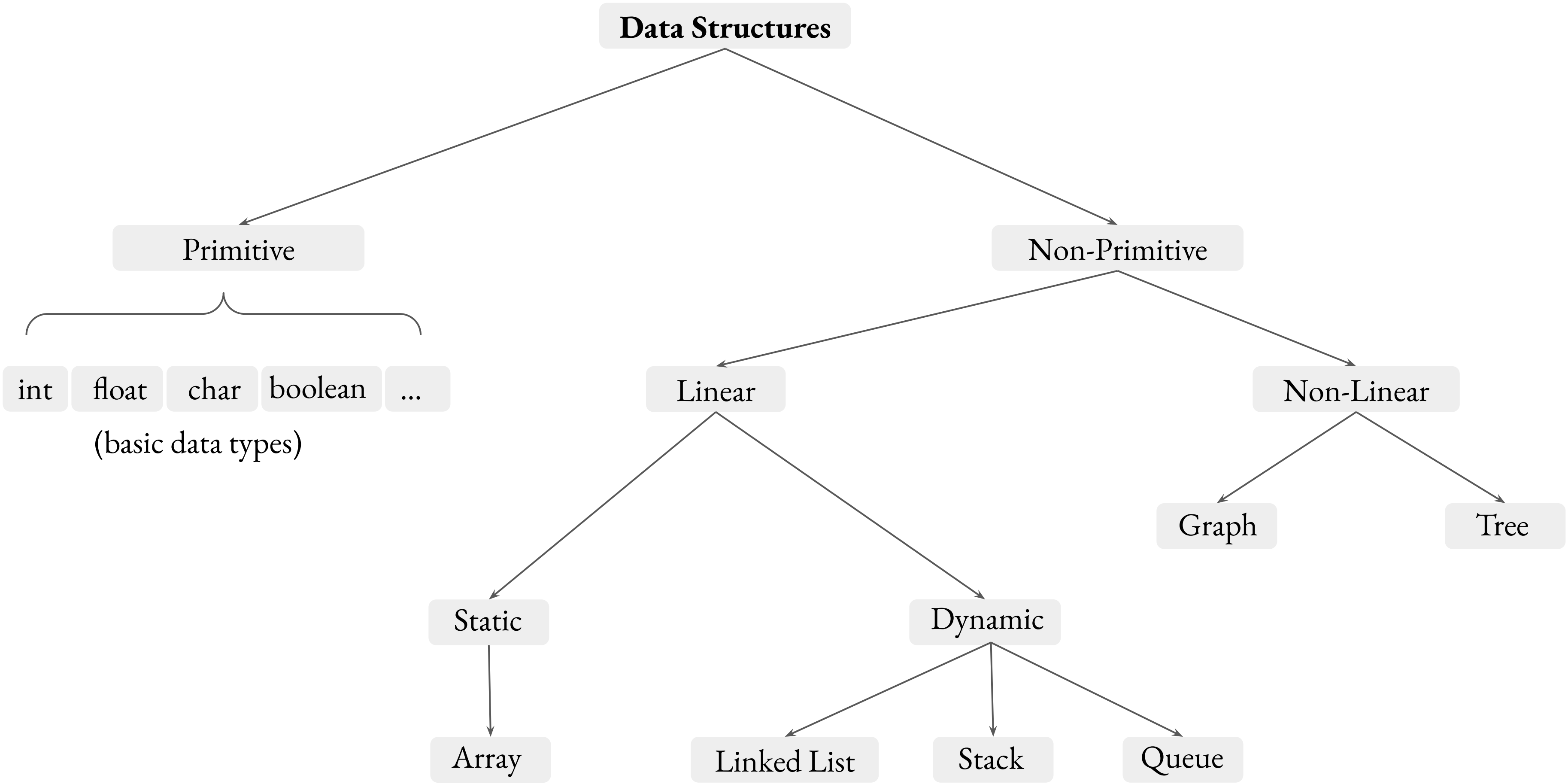
Associate Professor
School of Computing and Data Science

ram.krish@saiuniversity.edu.in

SAI
UNIVERSITY

LINKED LIST





Data Structures

Primitive

int

float

char

boolean

...

(basic data types)

Non-Primitive

Linear

Non-Linear

Graph

Tree

Static

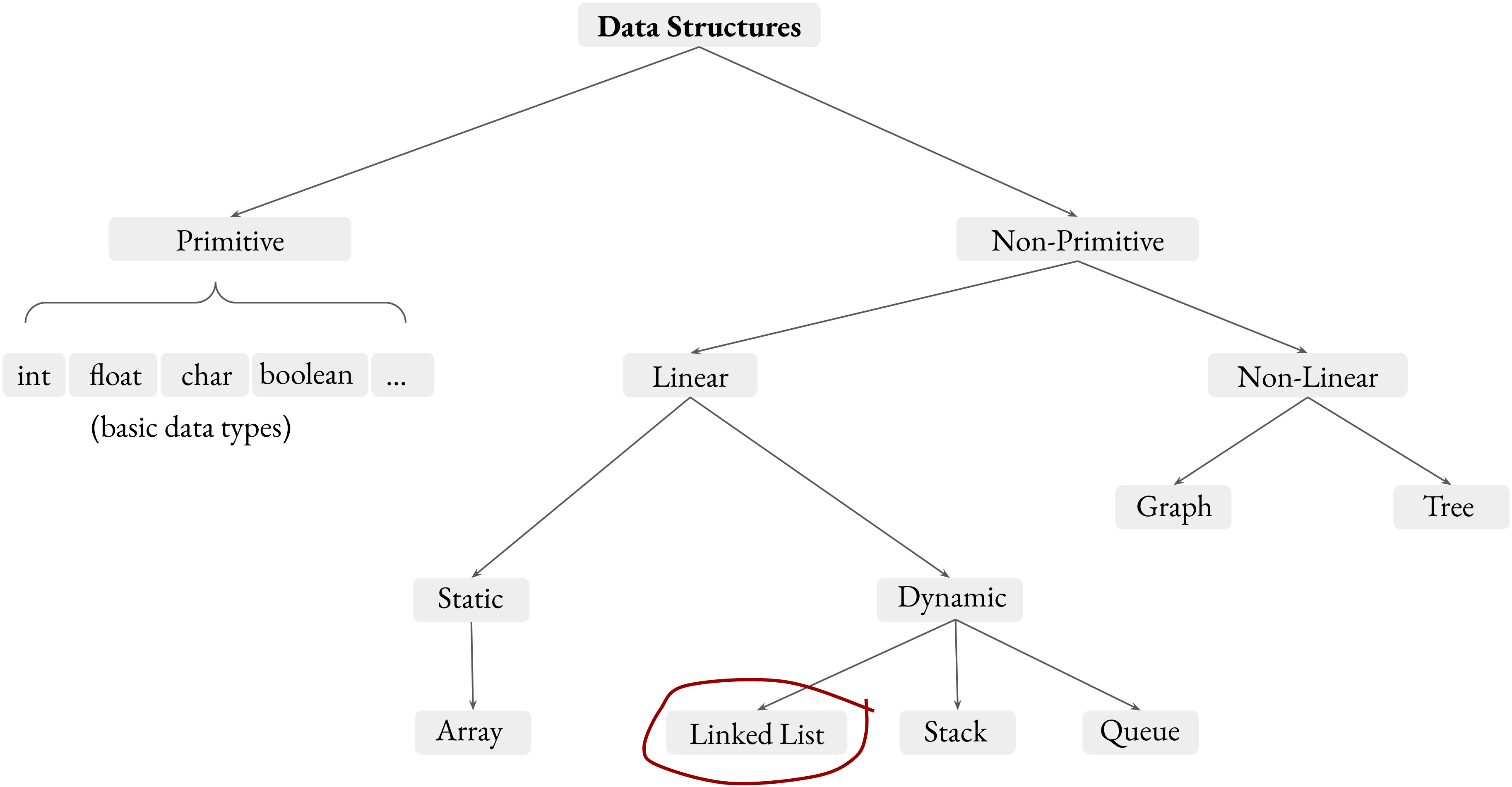
Dynamic

Array

Linked List

Stack

Queue



Data Structures

Primitive

int

float

char

boolean

...

(basic data types)

Non-Primitive

Linear

Non-Linear

Graph

Tree

Static

Dynamic

Array

Linked List

Stack

Queue

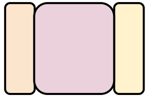
Linked List

Linked List: A linked list is a fundamental data structure that stores elements in a **linear order**, but unlike arrays, **not necessarily in contiguous memory locations**.

- Order of the data stored in a linked list is determined by the pointer in each object/node.

Doubly Linked List

Object / Node



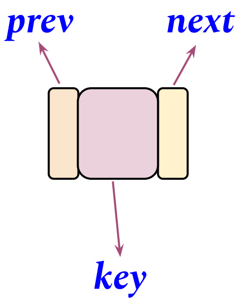
Linked List

Linked List: A linked list is a fundamental data structure that stores elements in a **linear order**, but unlike arrays, **not necessarily in contiguous memory locations**.

- Order of the data stored in a linked list is determined by the pointer in each object/node.

Doubly Linked List

Object / Node



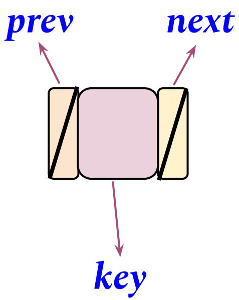
Linked List

Linked List: A linked list is a fundamental data structure that stores elements in a **linear order**, but unlike arrays, **not necessarily in contiguous memory locations**.

- Order of the data stored in a linked list is determined by the pointer in each object/node.

Doubly Linked List

Object / Node

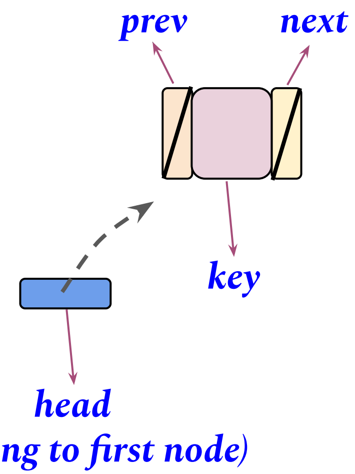


Linked List

Linked List: A linked list is a fundamental data structure that stores elements in a **linear order**, but unlike arrays, **not necessarily in contiguous memory locations**.

- Order of the data stored in a linked list is determined by the pointer in each object/node.

Doubly Linked List
Object / Node

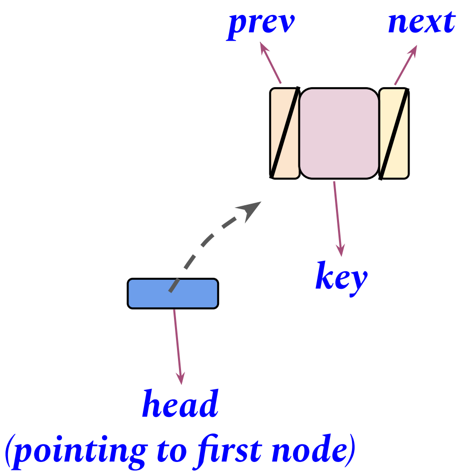


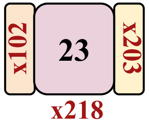
Linked List

Linked List: A linked list is a fundamental data structure that stores elements in a **linear order**, but unlike arrays, **not necessarily in contiguous memory locations**.

- Order of the data stored in a linked list is determined by the pointer in each object/node.

Doubly Linked List
Object / Node



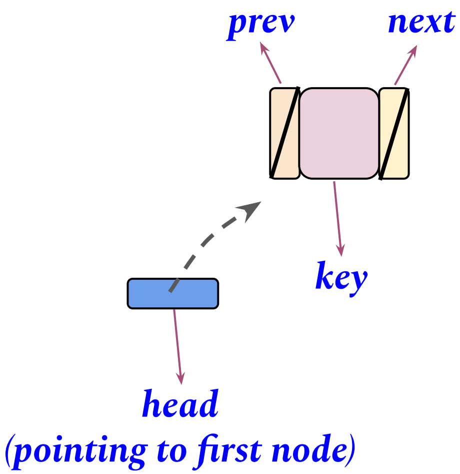


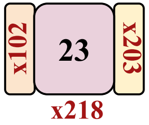
LINKED LIST

Doubly Linked List

In a doubly linked list, each element x has the following attributes:

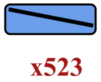
- ***x.key***
- ***x.next***: the successor of x, NIL if x has no **successor** so that it's the tail
- ***x.prev***: the predecessor of x, NIL if x has no **predecessor** so that it's the head
- ***L.head*** points to the first element of the list, NIL if the list is empty.



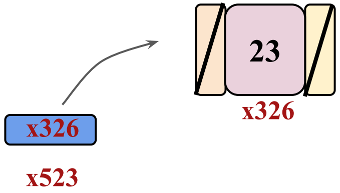


LINKED LIST

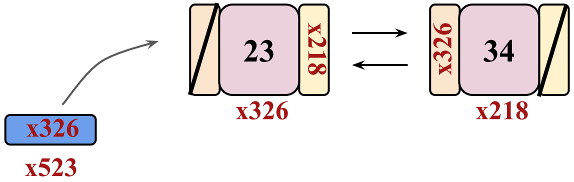
Empty list



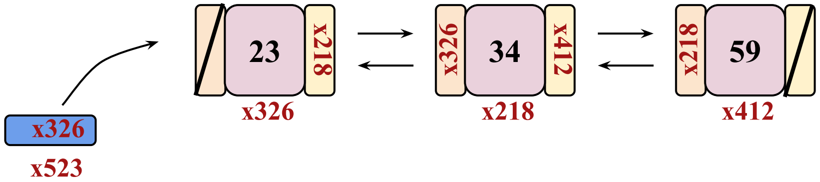
One element list



Two elements list



Three elements list



C Structures

STRUCTURES

Structures

- Collection of one or more variables grouped under a single name.
    - The variables we group together can be of different data types.
        - So, structure is a small heterogeneous collection of dtype.
        - Arrays are homogeneous.
            - We can also have arrays of structures

- We can also define pointers to structure.

- Structures can be dynamically allocated in heap.

- Structures can be defined inside another structure.

Note: Size of a structure is not always the sum of individual dtype sizes.

STRUCTURES

Syntax:

```
struct tag
{
   member1;
   member2;
   ...
   ...
};
```

```
struct tag instance;

instance.member1;
instance.member2;
```

```
member1 → dtype variable;
member1 → int x;
member2 → float y;
```

Syntax:

```
struct pts
{
   int x;
   float y;

};
```

```
struct pts pt1;

pt1.x;
pt1.y;
```

Pointers to Structures

STRUCTURES

Pointers to Structures

- Pointers can be defined to structures similar to any other default dtype.
- To access structure members using pointers, we use -> operator.
- Structures can also be dynamically allocated.

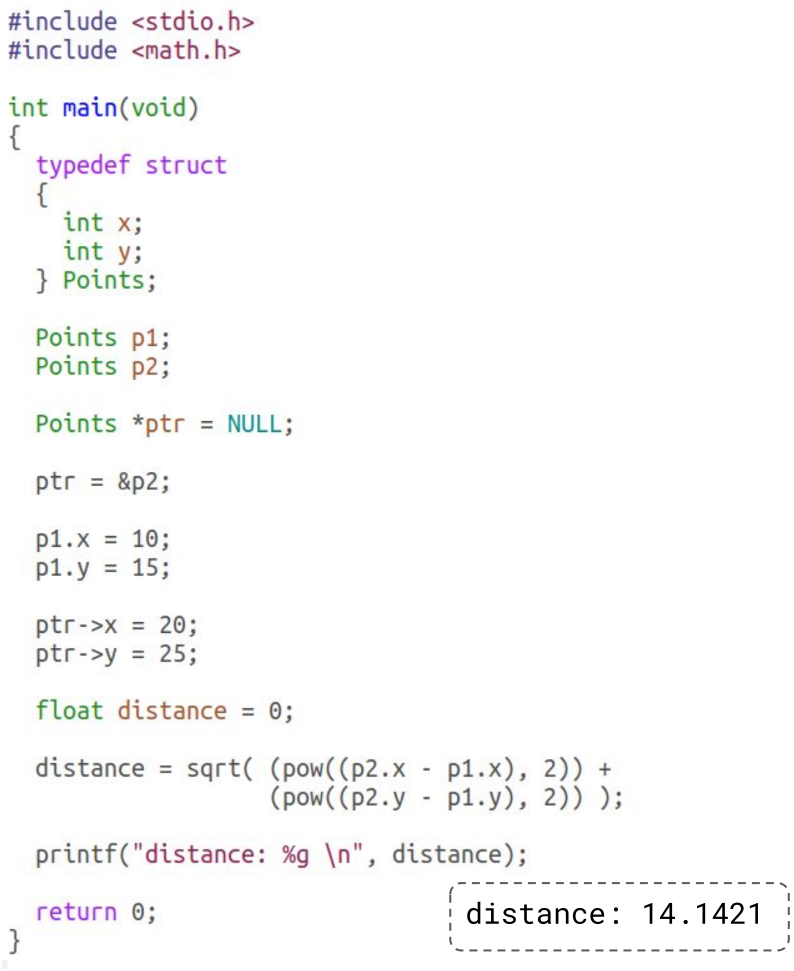
```
typedef struct
{
   int x;
   float y;
} Points;
```

P -> y is equivalent to (***P**).y

```
Points pt1;
Points *ptr = NULL;

pt1.x = 15;
pt1.y = 12.4;
```

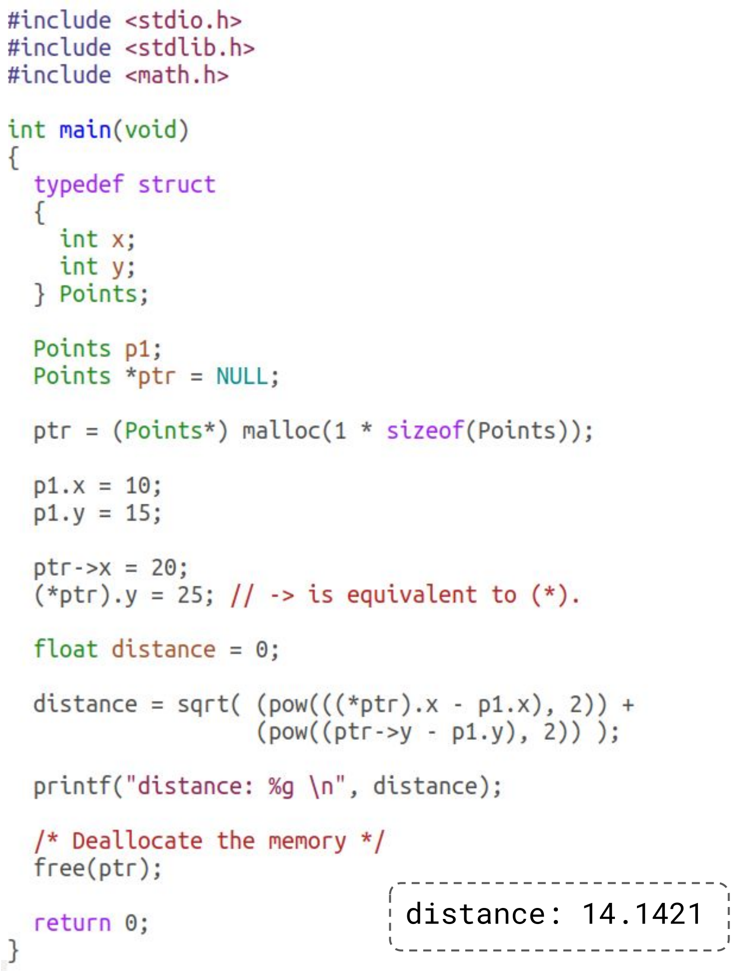
```
ptr = &pt1;

ptr->x = 15;

(*ptr).y = 12.4;
```

Structures



```c
#include <stdio.h>
#include <math.h>

int main(void)
{
  typedef struct
  {
    int x;
    int y;
  } Points;

  Points p1;
  Points p2;

  Points *ptr = NULL;

  ptr = &p2;

  p1.x = 10;
  p1.y = 15;

  ptr->x = 20;
  ptr->y = 25;

  float distance = 0;

  distance = sqrt( (pow((p2.x - p1.x), 2)) +
                   (pow((p2.y - p1.y), 2)) );

  printf("distance: %g \n", distance);

  return 0;
}
```

```
distance: 14.1421
```

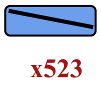


```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(void)
{
  typedef struct
  {
    int x;
    int y;
  } Points;

  Points p1;
  Points *ptr = NULL;

  ptr = (Points*) malloc(1 * sizeof(Points));

  p1.x = 10;
  p1.y = 15;

  ptr->x = 20;
  (*ptr).y = 25; // -> is equivalent to (*).

  float distance = 0;

  distance = sqrt( (pow(((*ptr).x - p1.x), 2)) +
                   (pow((ptr->y - p1.y), 2)) );

  printf("distance: %g \n", distance);

  /* Deallocate the memory */
  free(ptr);

  return 0;
}
```

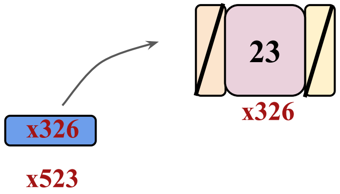
```
distance: 14.1421
```

Linked List

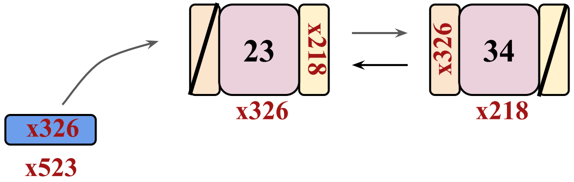
Empty list



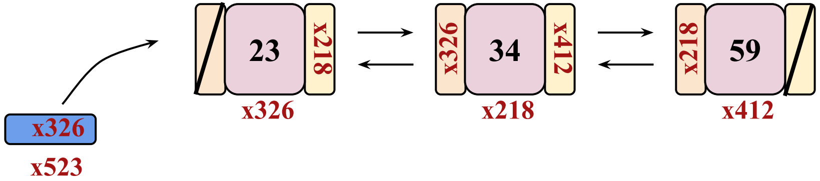
One element list



Two elements list

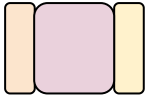


Three elements list



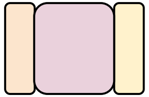
Linked List

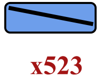
```
struct node
{
 int key;
 struct node *prev;
 struct node *next;
};
```



Linked List

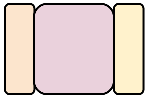
```
struct node
{
 int key;
 struct node *prev;
 struct node *next;
};

struct node *L_head = NULL; // Empty List
```



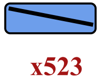


Linked List

```
struct node
{
 int key;
 struct node *prev;
 struct node *next;
};
```



```
struct node *L_head = NULL; // Empty List
```



```
struct node *createNode(int x)
{
 struct node *newNode = (struct node *)malloc(1 * sizeof(struct node));

 newNode->key = x;
 newNode->prev = NULL;
 newNode->next = NULL;

 return newNode;
}
```