Structures and Linked List

Winter Workshop Programming

Robotics Club IITKanpur

Motivation: Mixing Types

- A geometry package – we want to define a point as having an x coordinate, and a y coordinate.
- Student data – Name and Roll Number
- First strategy: Array of size 2?
    - Roadblock: Can not mix TYPES
- Two variables,

```
int point_x , point_y ;      char *name; int roll_num;
```

    - No way to indicate that they are part of the same name
    - We need to be very careful about variable names.
- Is there any better way ?

Structures

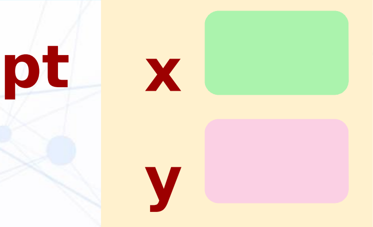
- A structure is a collection, of variables, under a common name
- The variables can be of different types (including arrays, pointers or structures themselves!)
- Structure variables are called fields

```
struct point {
     int x;
     int y;
};
struct point pt;
```

Defines a structure called point containing two integer variables (fields), called x and y.

struct point pt defines a variable pt to be of type struct point

A shelf with different compartments



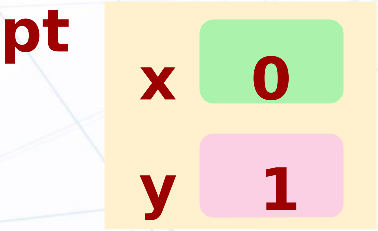
memory depiction of pt

Careful about the semicolon at the end

Structures

- The x field of pt is accessed as pt.x.
- Field pt.x is an int and can be used as any other int.
- he y field of pt is accessed as pt.y

```
struct point {
     int x;
     int y;
};
struct point pt;
pt.x = 0;
pt.y = 1;
```



memory depiction of pt

Structures

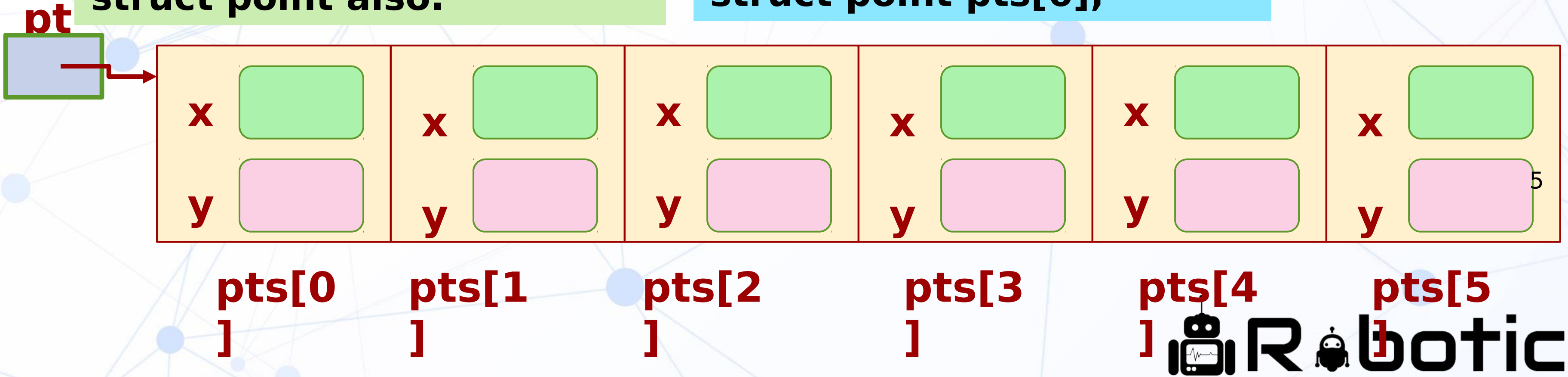
```
struct point {
      int x; int y;
}
```

struct point is a type.
It can be used just like int, char etc..

For now, define structs in the beginning of the file, after #include.

We can define array of struct point also.

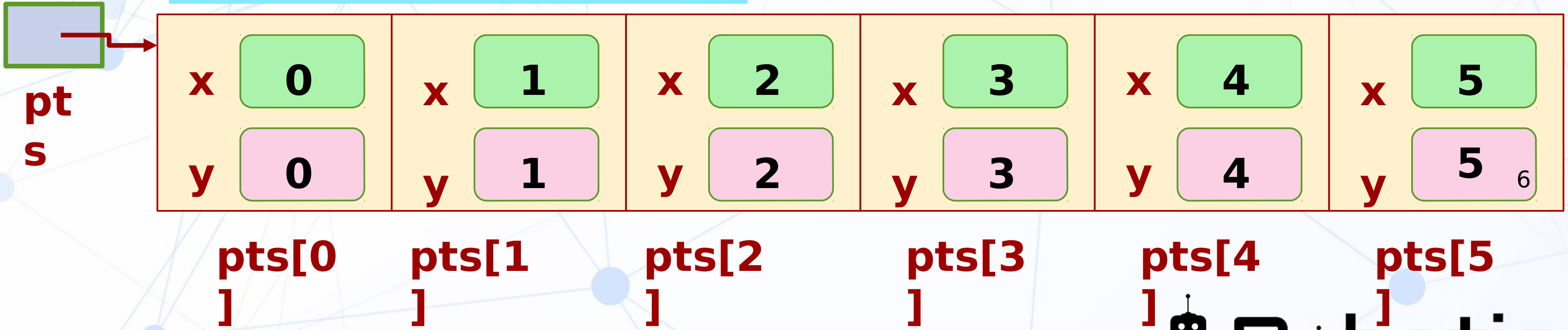
```
struct point pt1,pt2;
struct point pts[6];
```



Structures

```
struct point {
      int x; int y;
};
struct point pts[6];
int i;
for (i=0; i < 6; i=i+1) {
      pts[i].x = i;
      pts[i].y = i;
}
```

State of memory after the code executes.



Reading Structures (scanf ?)

```
struct point {
     int x; int y;
};
```

```
int main() {
    int x, y;
    struct point pt;
    scanf(“%d%d”, &(pt.x),&(pt.y));
    return 0;
}
```

- You **cannot** read a structure directly using scanf!
- **Read individual fields** using scanf (note the &).
- A better way is to define our own functions to read structures
    - to avoid cluttering the code

Functions Returning Structures

```
struct point make_pt(int x, int y) {
     struct point temp;
     temp.x = x;
     temp.y = y;
     return temp;    }

int main() {
     int x, y;
     struct point pt;
     scanf(“%d%d”, &x,&y);
     pt = make_pt(x,y);
      return 0;
}
```

```
struct point {
     int x; int y;
};
```

- **make_pt(x,y): creates a struct point with coordinates (x,y), and returns a struct point.**
- **Functions can return structures just like int, char, int *, etc..**
- **struct can be passed as arguments (pass by value).**

Given int coordinates x,y, make_pt(x,y) creates and returns a struct point with these coordinates.

Functions with struct as Parameters

```
# include <stdio.h>
# include <math.h>
struct point {
      int x; int y;
};
double norm2(struct point p)  {
   return sqrt (p.x*p.x + p.y*p.y);
}
int main() {
      int x, y;
      struct point pt;
      scanf(“%d%d”, &x,&y);
      pt = make_point(x,y);
      printf(“Euclidean distance
from origin is %f ”, norm2(pt) );
       return 0;}
```

The norm2 or Euclidean norm of point (x,y) is

$$\sqrt{x^2 + y^2}$$

Desired function:

norm2(struct point p) returns Euclidean norm of point p

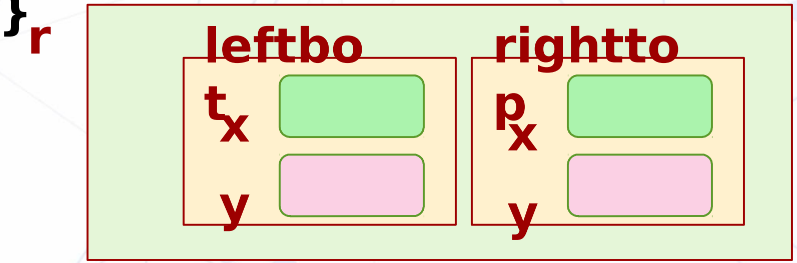
Passing Structures?

```
struct rect { struct point leftbot;
              struct point righttop; };
int area(struct rect r) {
   return
       (r.righttop.x – r.leftbot.x) *
       (r.righttop.y – r.leftbot.y);
}
void fun() {
   struct rect r1 ={{0,0}, {1,1}};
   area(r1);
}
```

We can pass structures as parameters, and return structures from functions, like the basic types int, char, double etc.

But is it efficient to pass structures or to return structures?

Usually NO. E.g., to pass struct rect as parameter, 4 integers are copied. This is expensive.



Same for returning structures

So what should be done to pass structures to functions?

Passing Structures?

```
struct rect { struct point leftbot;
               struct point righttop;};
int area(struct rect *pr) {
  return
 ((*pr).righttop.x – (*pr).leftbot.x) *
 ((*pr).righttop.y – (*pr).leftbot.y);
}
void fun() {
   struct rect r ={{0,0}, {1,1}};
   area (&r);
 }
```

Instead of passing structures, pass pointers to structures.

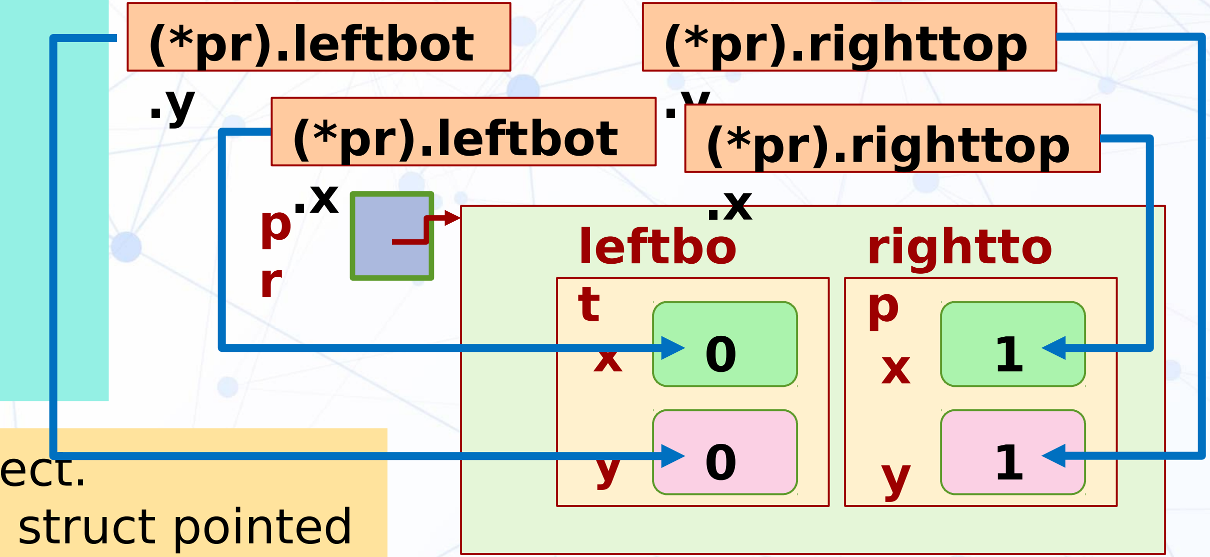
area() uses a pointer to struct as a parameter, instead of struct rect itself.

Only one pointer instead of large struct.

Robotics Club IITKanpur

Structure Pointers

```
struct point {
      int x; int y;};
struct rect {
   struct point leftbot;
   struct point righttop;
};
struct rect *pr;
```

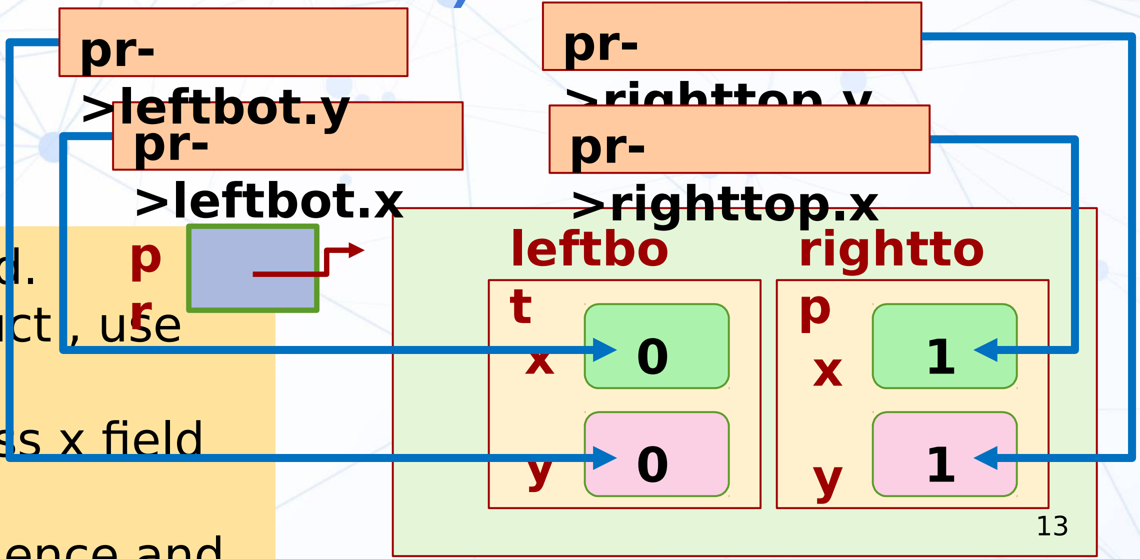


1. pr is pointer to struct rect.
2. To access a field of the struct pointed to by struct rect, use
   (*pr).leftbot
   (*pr).righttop
3. Bracketing (*pr) is essential here. * has lower precedence than .
4. To access the x field of leftbot, use (*pr).leftbot.x

Addressing fields via the structure's pointer

Addressing Fields via the Pointer (Shorthand)

1. Shorthand: (->) is provided.
2. To access a field of the struct , use pr->leftbot
3. -> is one operator. To access x field of leftbot, pr->leftbot.x
4. -> and . have same precedence and are left-associative. Equivalent to (pr->leftbot).x



pr->leftbot is equivalent to (*pr).leftbot

Data Structure

- What is a data structure?
- According to Wikipedia:
    - ... is a data organization, management and storage format that enables efficient access and modification
    - ... highly specialized to specific tasks.
- Examples: array, stack, queue, linked list, trees

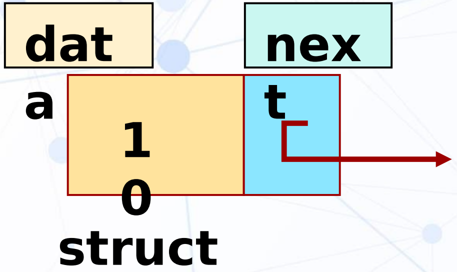
Linked List

- A linear, dynamic data structure, consisting of nodes. Each node consists of two parts:
  - a “data" component, and
  - a “next" component, which is a pointer to the next node (the last node points to nothing).

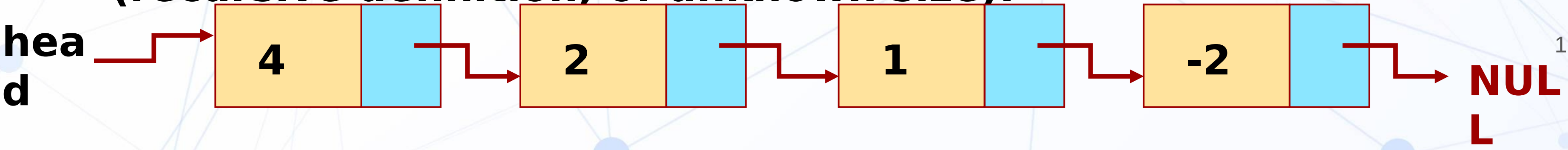
Robotics Club IITKanpur

Linked List: Self-referential Structure

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



1. Defines **struct node**, used as a node (element) in the “linked list”
2. Note that the field **next** is of type **struct node ***
3. **next** can’t be of type **struct node**
   (recursive definition, of unknown size).

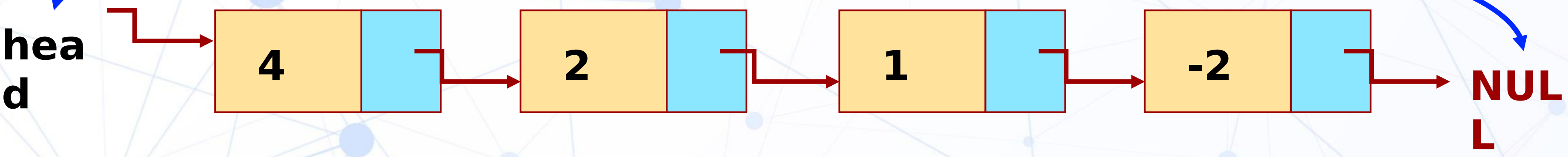


Only one link (pointer) from each node, hence “singly linked list”

Linked Lists

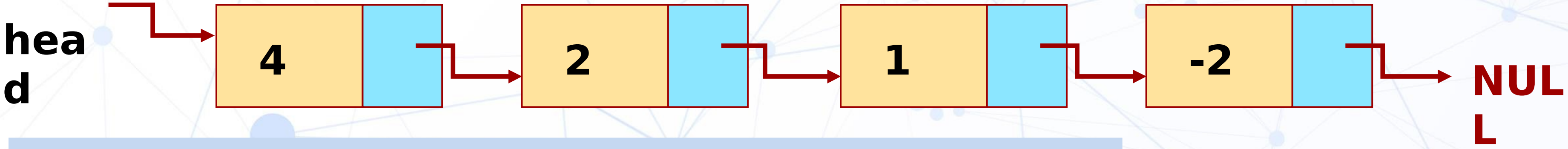
List starts at node pointed to by **head**

next field == NULL pointer indicates the last node of the list



1. The list is modeled by a variable (**head**): points to the first node of the list.
2. **head == NULL** implies empty list.
3. The next field of the **last** node is **NULL.**
4. Name **head** is just a convention – can give any name to the pointer to first node, but **head** is used most often.

Displaying a Linked List


head

4

2

1

-2

NULL

```
void display_list(struct node *head)
{
  struct node *cur = head;
  while (cur != NULL) {
     printf("%d ", cur->data);
     cur = cur->next;
  }
  printf("\n");
}
```

OUTPUT

```
4 2 1 -2
```

Create a New Node

Function make_node returns pointer to the starting of the list

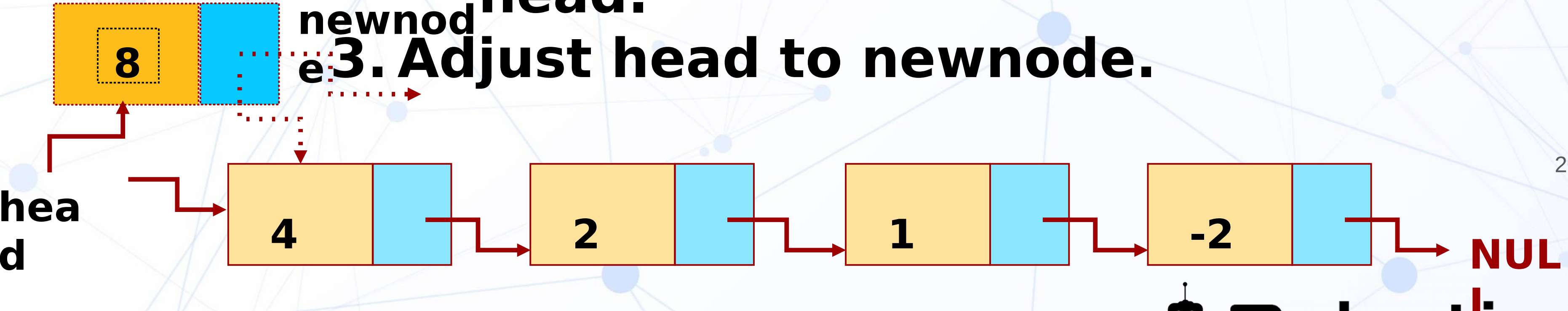
/* Allocates new node pointer and sets the data field to val, next field is NULL */

```
struct node * make_node(int val) {
    struct node *nd;
    nd = (struct node *) malloc(sizeof(struct node));
    nd->data = val;
    nd->next = NULL;
    return nd;
}
```

Insert at Front

Inserting at the front of the list.

1. Create a new node of type struct node. Data field set to the value given.
2. “Add’’ to the front:
   a. its next pointer points to target of head.
3. Adjust head to newnode.



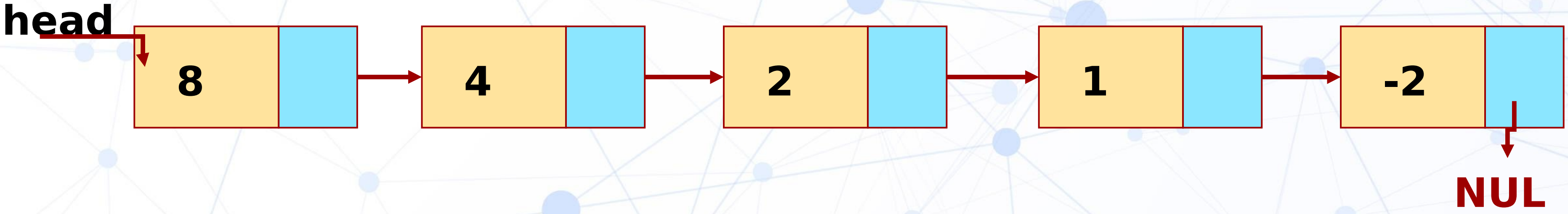
Robotics Club IITKanpur

```
struct node *insert_front(int val, struct node
*head) {
    struct node *newnode= make_node(val);
    newnode->next = head;
    head = newnode;
    return head;
}
```

Inserts newnode at the head of the list (pointed by head).

Returns pointer to the head of new list.

Works even when list is empty, i.e. head == NULL



Let’s start with an empty list and insert in sequence -2, 1,2, 4 and 8, given by user. Final list should be as

```
struct node *head = NULL;
int val; scanf (“%d”, &val);
while (val != -1) {
   head = insert_front (val, head);
   scanf (“%d”, &val);
```

INPUT: -2 1 2 4 8 -1

Creates list in the reverse order: head points to the last element inserted.
How to create list in the same order as input? (do it yourself)