Introduction to Linked List

IC-100

December, 2023

Today’s Class

- Linked List

Data Structure

- What is a data structure?
- According to Wikipedia:
  - … a particular way of storing and organizing data in a computer so that it can be used efficiently...
  - … highly specialized to specific tasks.
- Examples: array, a dictionary, a set, etc.

Data Structures Examples

- Sorted array:
  - Search is easy
  - Insert and delete are expensive
- Stack (Last in first out):
  - Insert and delete easy
  - Search is expensive
- Queue (First in first out)
  - Insert and delete easy
  - Search is expensive

Real World Data:Customer Info

```
enum act_Type {savings, current, fixDeposit, minor  };
typedef enum act_Type Accounts;

Struct cust_info {
        int Account_Number;
        Accounts Account_Type;
        char *Customer_Name;
        char* Customer_Address;
        bitmap Signature_scan; // user defined type bitmap
    } ;
```

Customer can have more than 1 accounts

- Want to keep multiple accounts for a customer together for easy access

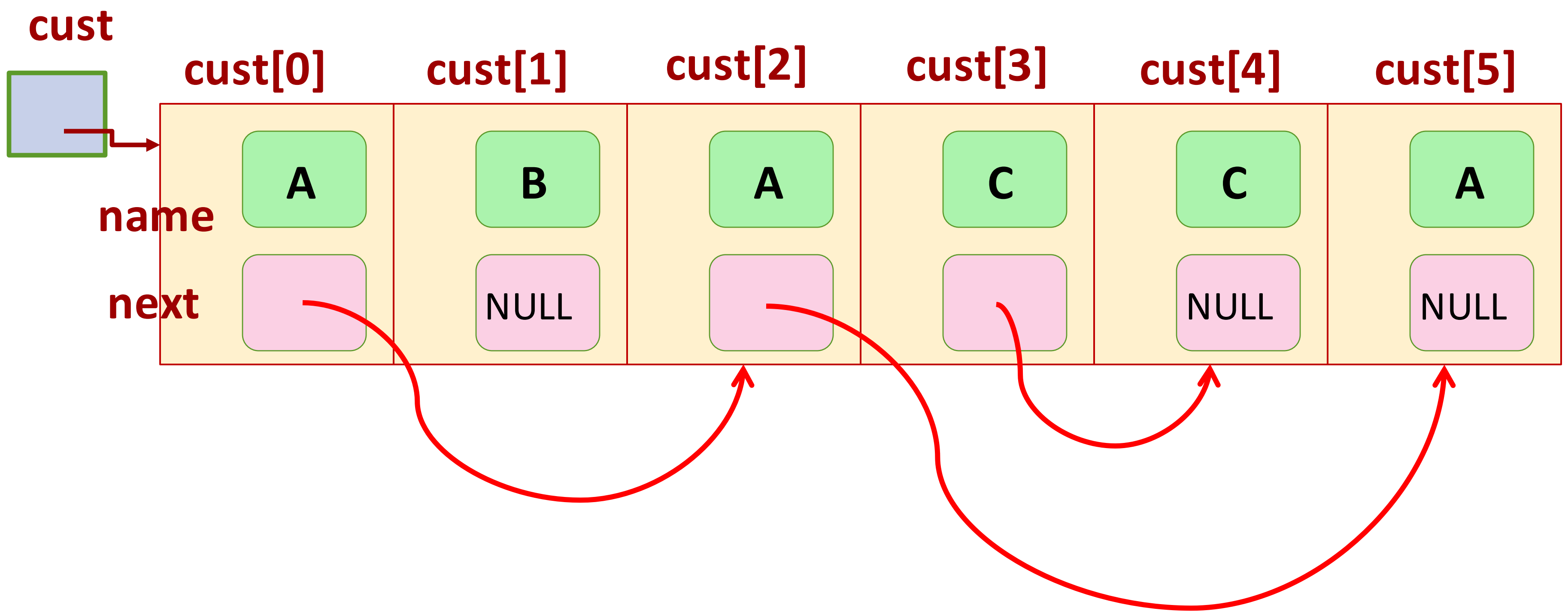
Needs a Dynamic Data Structure

- “Link” all the customer accounts together using a “chain-of-pointers”
- Struct cust_info {

```
        int Account_Number;
        Accounts Account_Type;
        char *Customer_Name;
        char* Customer_Address;
        bitmap Signature_scan; // user defined type bitmap
        struct cust_info* next_account;
    } ;
```

- Why not (?):
  - struct cust_info next_account;

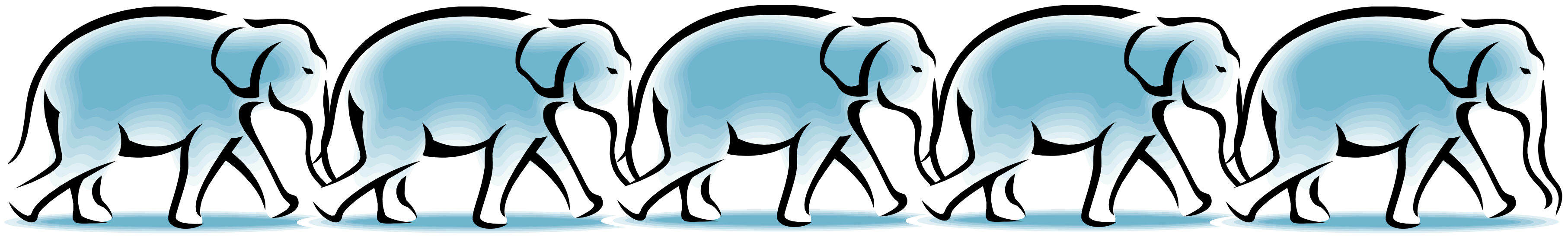
Error: Field next_account has **incomplete type**



cust[i].next, cust[i].next->next, cust[i].next->next->next etc., when **<u>not NULL</u>**, point to the "other" records of the same customer

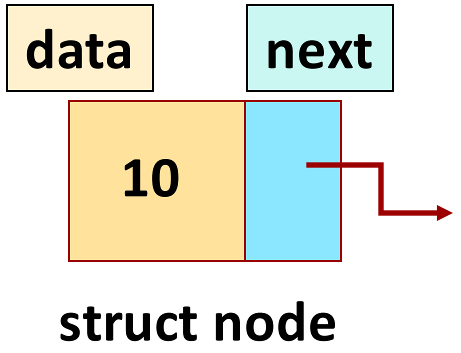
Linked List

- 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).



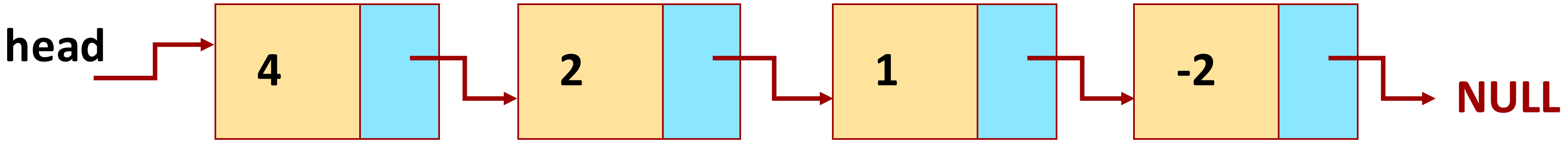
Linked List : A Self-Referential Structure

```
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 or infinite size).

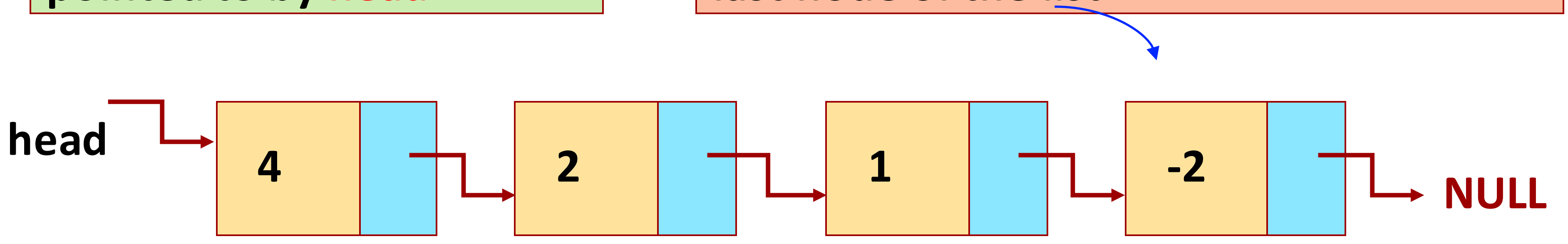


Only one link (pointer) from each node, hence "**singly linked list**".

Linked Lists

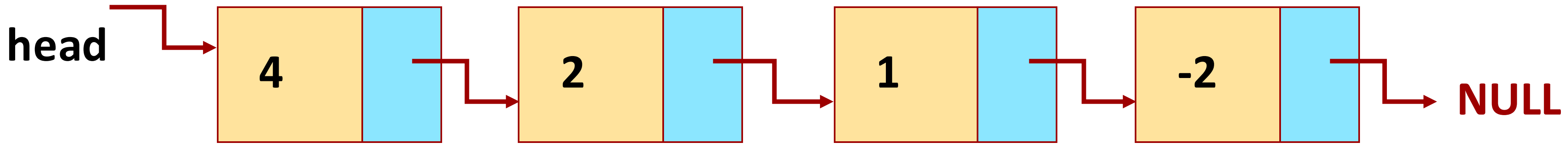
List starts at node pointed to by **head**

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



```
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

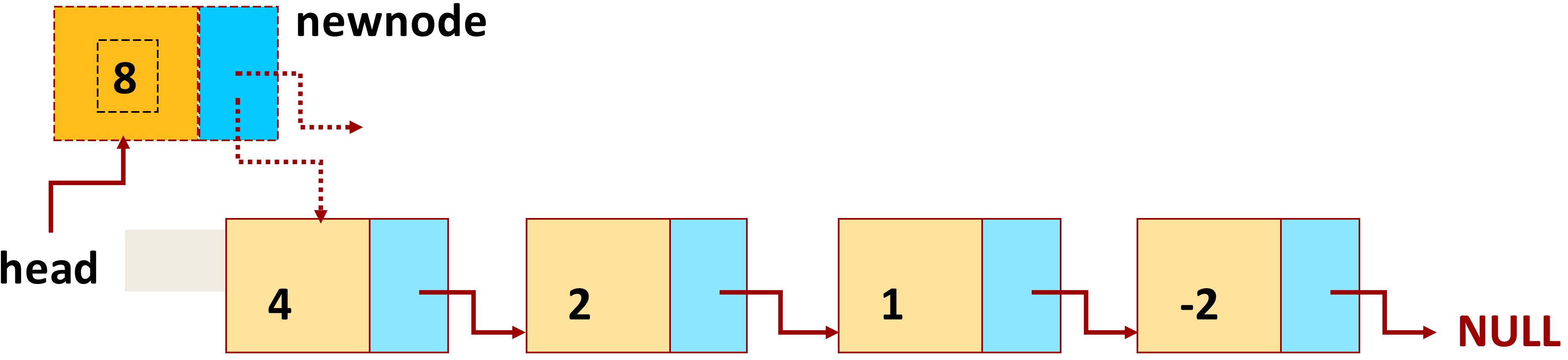
/* Allocates new node pointer and sets the data field to val, next field is NULL */

```
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

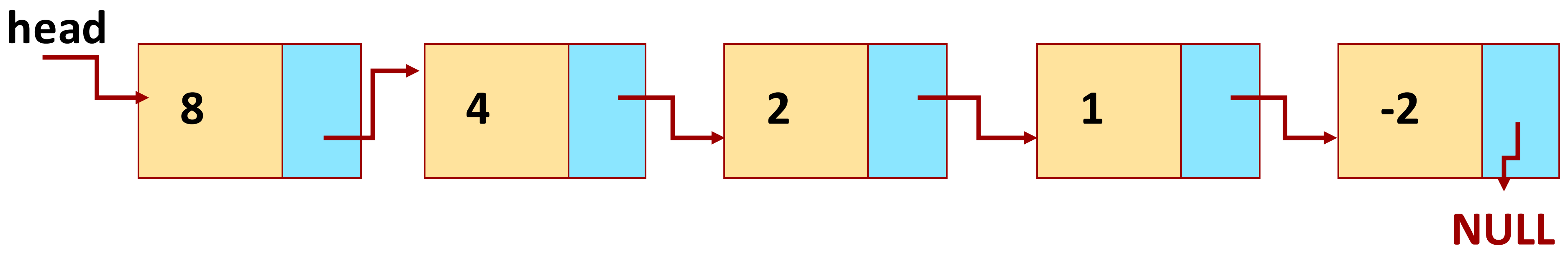
Inserting at the front of the list.

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



```
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 above.

```
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?

Use of typedef

- Repetitive to type "struct node" for parameters, variables etc.
- C allows naming types— the typedef statement.

Define a new type Listnode as struct node *

```
typedef struct node * Listnode;
```

Listnode is a type. It can be used for struct node * in variables, parameters, etc..

```
Listnode head, curr;
 /* search in list for key */
Listnode search(Listnode list, int key);
/* insert the listnode n in front of listnode list */
Listnode insert_front(Listnode list, Listnode n);
 /* insert the listnode n after the listnode curr */
Listnode insert_after(Listnode curr, Listnode n);
```

Next Class

- More About Linked List
- Implementation of Common Data Structures using Linked List