INFORMATIKA UMN

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA

IF232
ALGORITHMS
&
DATA STRUCTURES

04

LINKED LISTS I

DENNIS GUNAWAN

REVIEW

File Processing:

Data Hierarchy

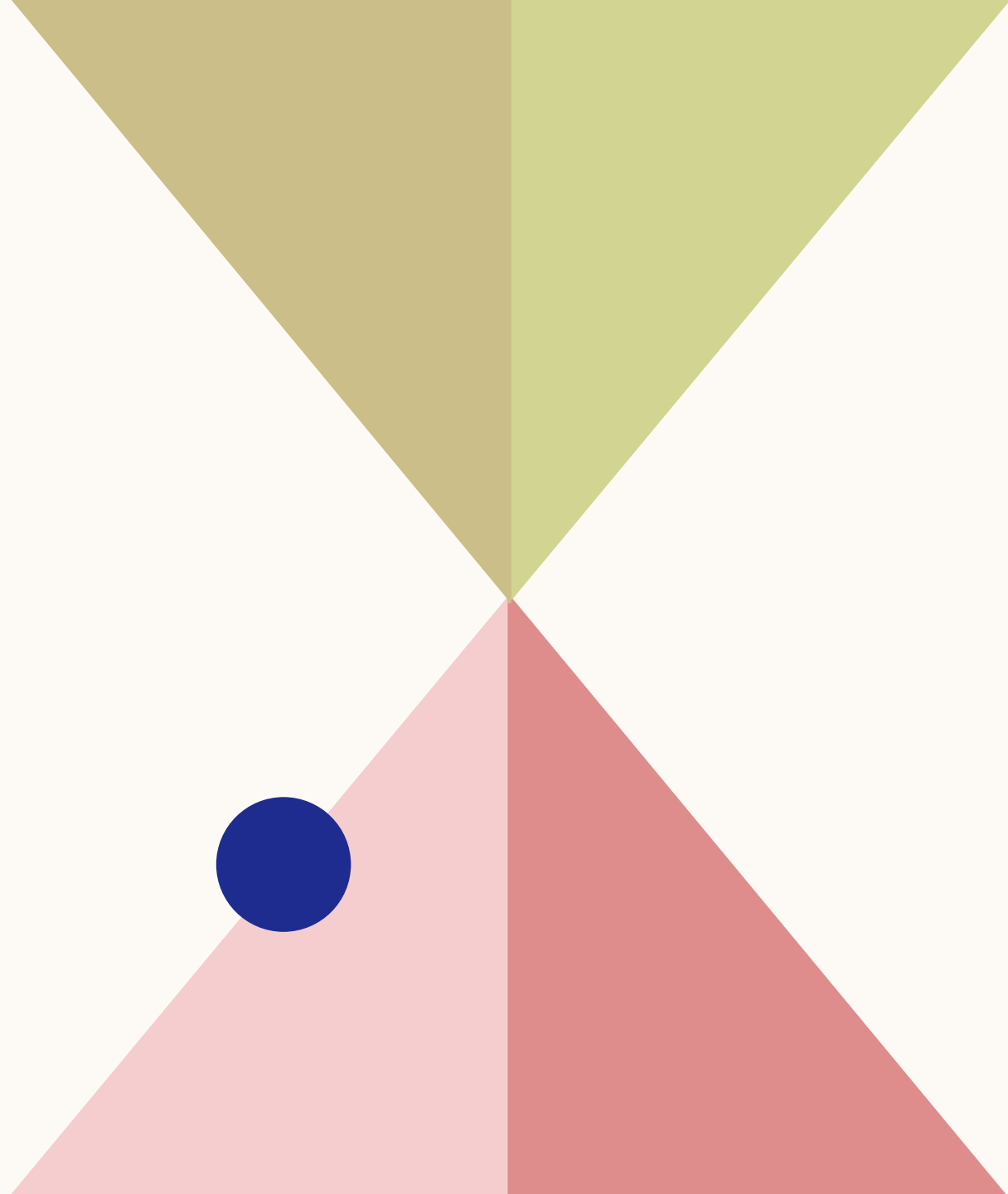
Files and Streams

Sequential-Access File Processing

OUTLINE

Dynamic Memory Allocation

Single Linked Lists



DYNAMIC MEMORY ALLOCATION

Static Memory Allocation	Dynamic Memory Allocation
Allocated at local stack memory	Allocated at heap memory
Allocated at compile time; Does not change during program execution	Allocated at run time
Freed when the program terminates	Can be freed at any time
Used if you know exactly how much data you need to allocate before compile time	Used if you don't know exactly how much data you will need at run time

DYNAMIC MEMORY ALLOCATION

- Creating and maintaining dynamic data structures that can grow and shrink as the program runs requires **dynamic memory allocation**
  - The ability for a program to obtain more memory space at execution time and to release space no longer needed

- Functions **`malloc`** and **`free`**, and operator **`sizeof`**, are essential to dynamic memory allocation

DYNAMIC MEMORY ALLOCATION

- Syntax

```
void * malloc(size_t size);
```

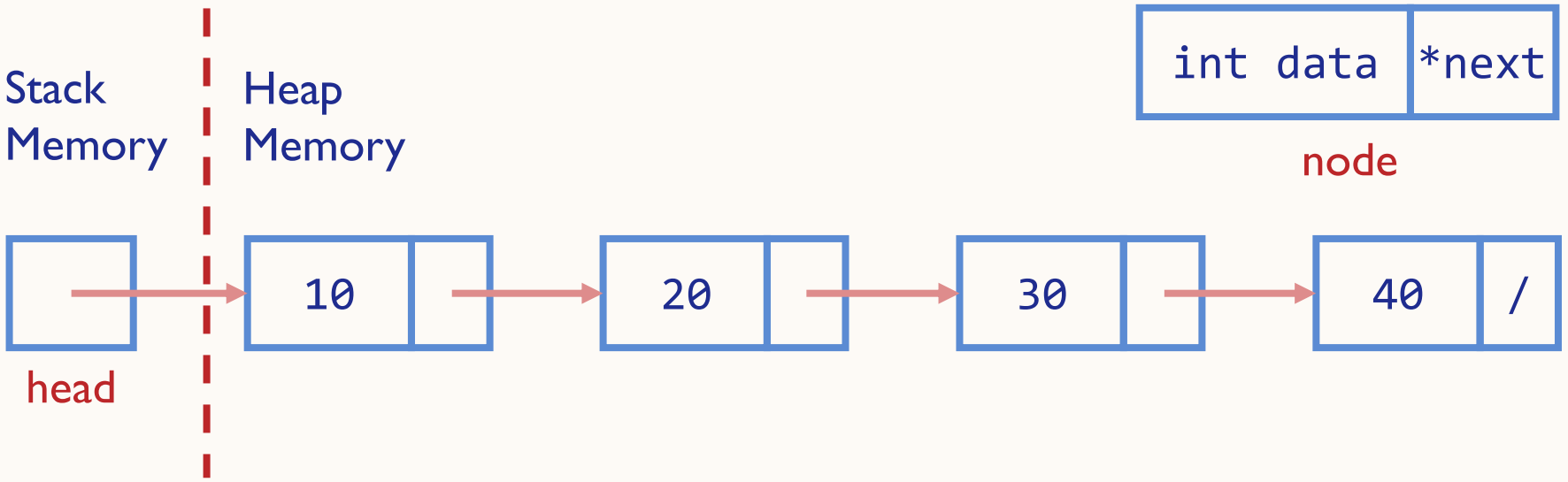
```
void free(void *block);
```

- Example

```
int *n;
n = (int *) malloc(sizeof(int));
...
free(n);
```

LINKED LISTS

- A linked list is **a linear collection of self-referential structures**, called **nodes**, connected by **pointer links**
- Linked list is a data structure which in turn can be used to implement other data structures, such as stacks, queues, and their variations



LINKED LISTS

Array	Linked List
Linear collection of data elements	
Number of data elements is predictable	Number of data elements is unpredictable
The size of an array created at compile time, however, cannot be altered	Dynamic: the length of a list can increase or decrease at execution time as necessary
Arrays can become full	Linked lists become full only when the system has insufficient memory to satisfy dynamic storage allocation requests
Stores its members in consecutive memory locations	Does not store its nodes in consecutive memory locations
Insertions and deletions can be done at any point in the list in a constant time	Does not allow random access of data

DECLARATION

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

struct tnode

```
int main()
{
    struct tnode *head, *node;
    int number;
    ...
    head = NULL;
    ...
}
```

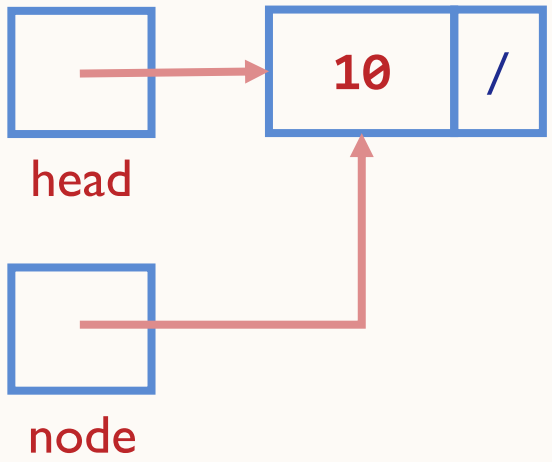
head

node

INSERTION

AT THE BEGINNING OF A LINKED LIST

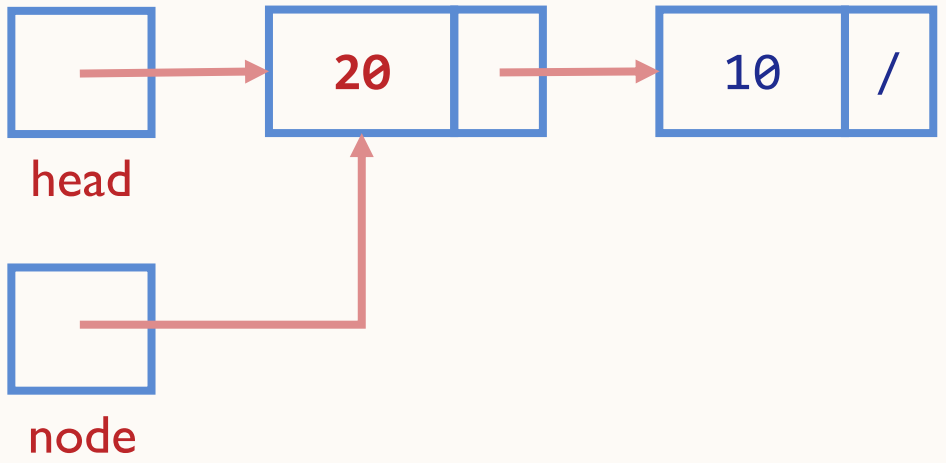
```
scanf("%d", &number); //10
1 node = (struct tnode *) malloc
        (sizeof(struct tnode));
2 node->data = number;
3 if(head == NULL)
    node->next = NULL;
  else
    node->next = head;
4 head = node;
```



INSERTION

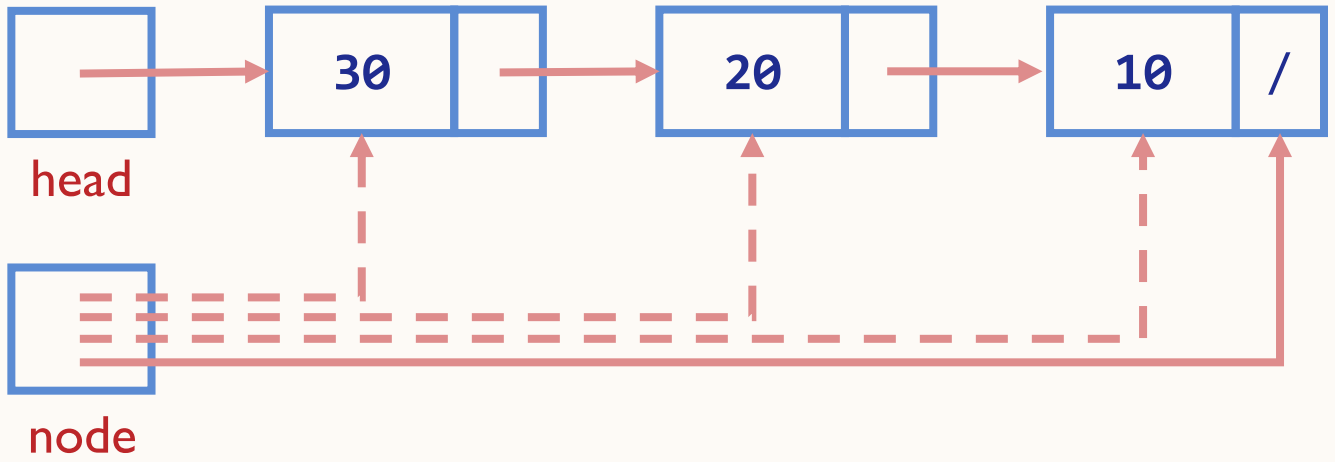
AT THE BEGINNING OF A LINKED LIST

```
scanf("%d", &number); //20
1 node = (struct tnode *) malloc
        (sizeof(struct tnode));
2 node->data = number;
  if(head == NULL)
     node->next = NULL;
  else
3    node->next = head;
4 head = node;
```



TRAVERSAL / DISPLAY

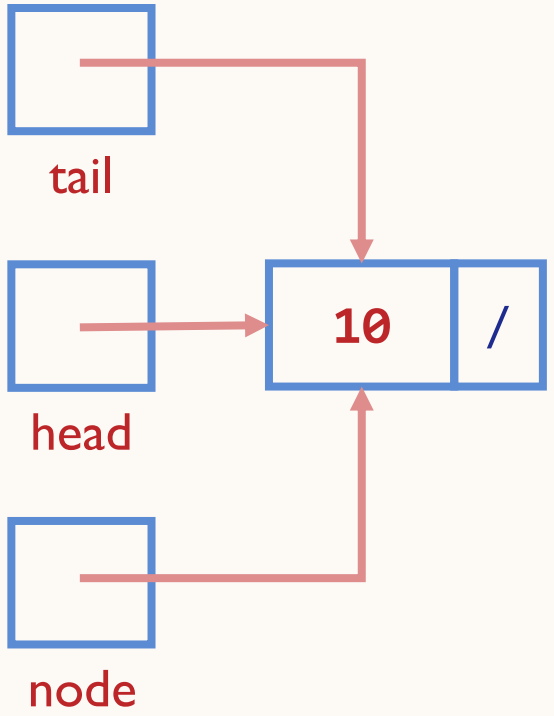
```
node = head;
while(node != NULL){
    printf("%d ", node->data);
    node = node->next;
}
```



INSERTION

AT THE END OF A LINKED LIST

```
scanf("%d", &number); //10
1 node = (struct tnode *) malloc
        (sizeof(struct tnode));
2 node->data = number;
  node->next = NULL;
3 if(head == NULL)
     head = node;
  else
     tail->next = node;
4 tail = node;
```

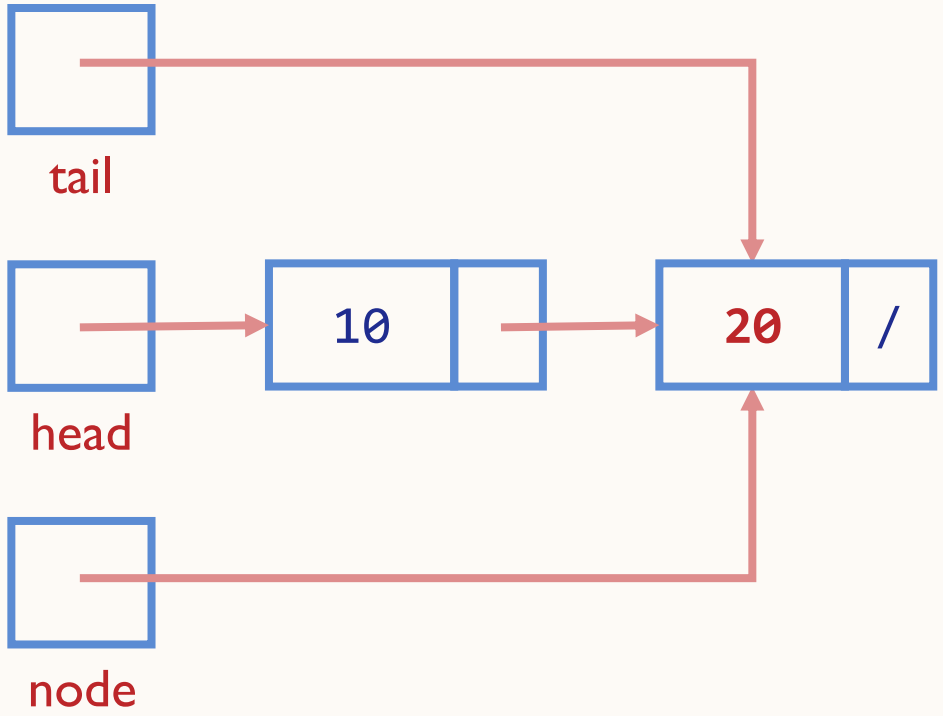


INSERTION

AT THE END OF A LINKED LIST

```
scanf("%d", &number); //20
node = (struct tnode *) malloc
       (sizeof(struct tnode));
node->data = number;
node->next = NULL;
if(head == NULL)
   head = node;
else
   tail->next = node;
tail = node;
```

1
2
3
4

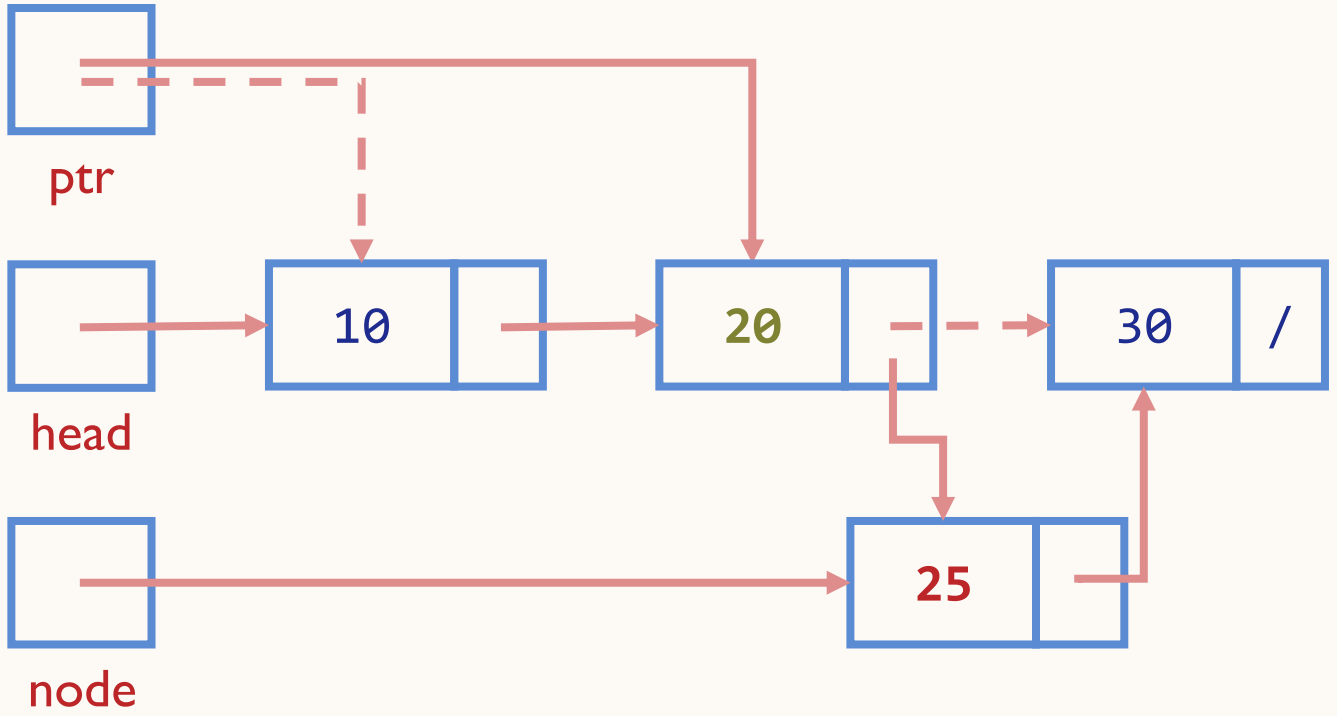


INSERTION

AFTER A GIVEN NODE IN A LINKED LIST

```
scanf("%d", &number); //25
node = (struct tnode *) malloc
       (sizeof(struct tnode));
node->data = number;
node->next = NULL;

ptr = head;
while(ptr->data != 20){
    ptr = ptr->next;
}

node->next = ptr->next;
ptr->next = node;
```

1 2 3 4



INSERTION

BEFORE A GIVEN NODE IN A LINKED LIST

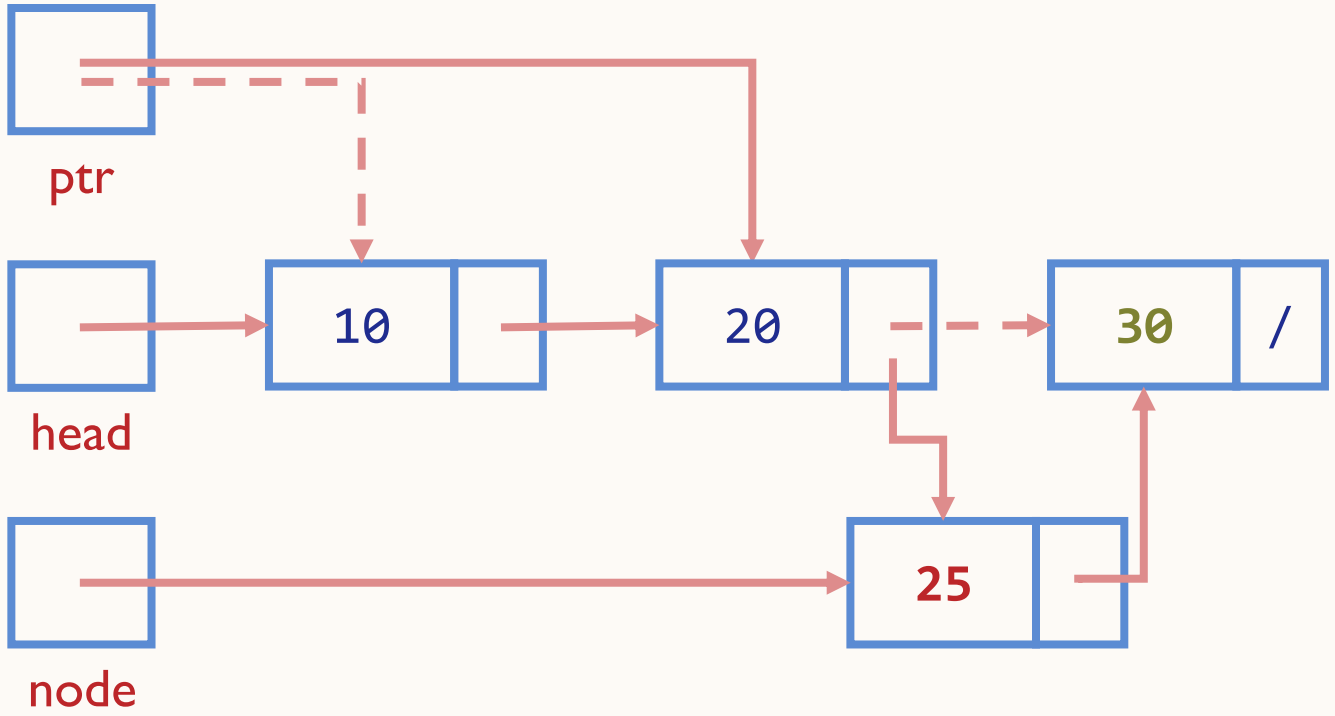
```
scanf("%d", &number); //25
node = (struct tnode *) malloc
       (sizeof(struct tnode));
node->data = number;
node->next = NULL;

ptr = head;
while(ptr->next->data != 30){
    ptr = ptr->next;
}

node->next = ptr->next;
ptr->next = node;
```

1

2

3

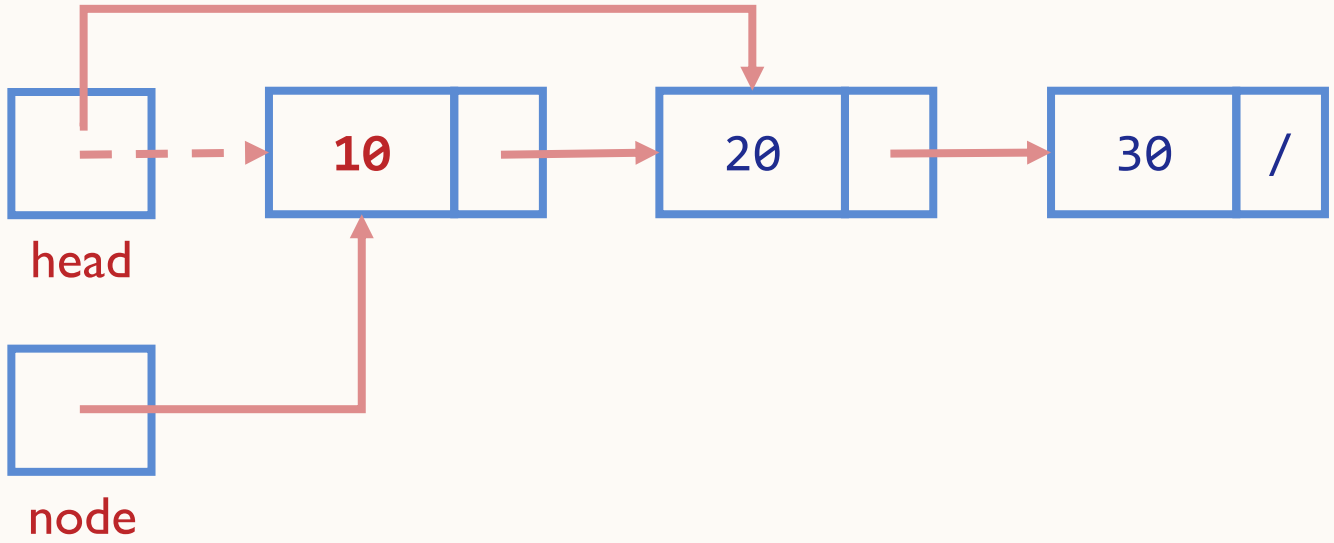
4



DELETION

DELETING THE FIRST NODE FROM A LINKED LIST

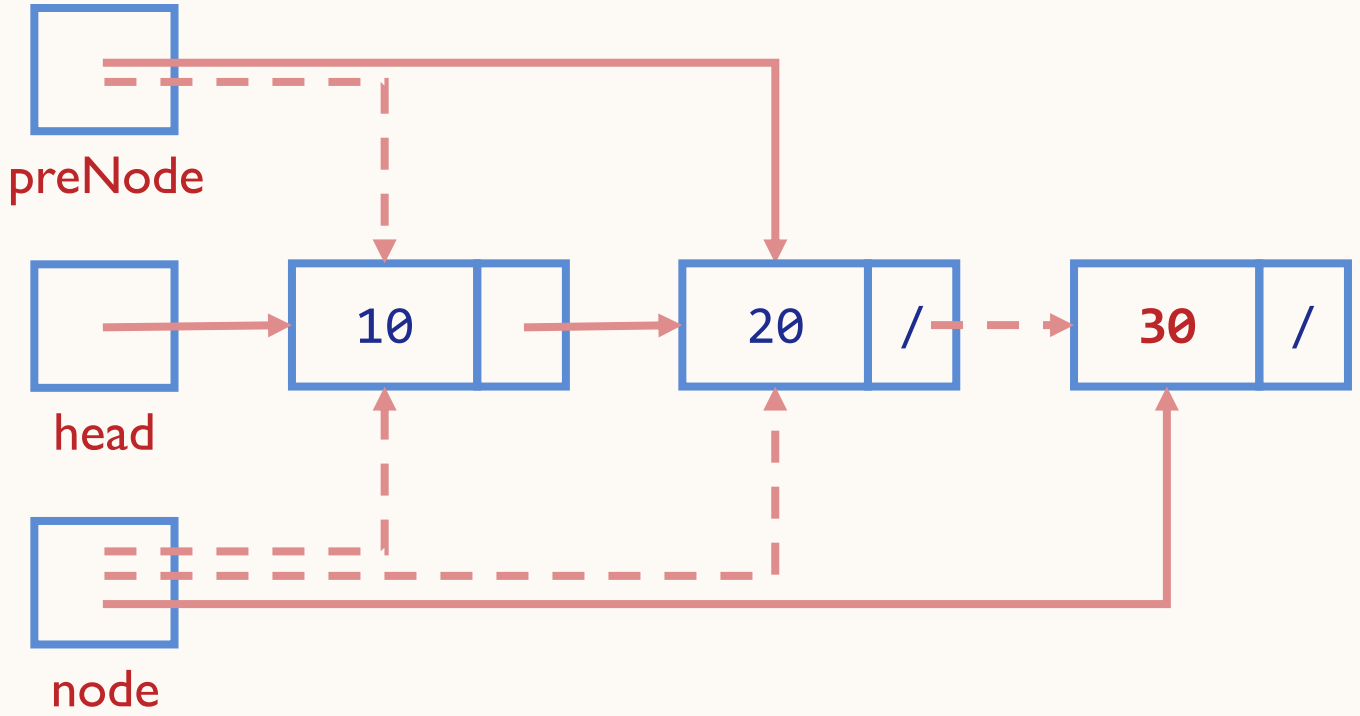
```
node = head;
head = head->next;
node->next = NULL; //optional
free(node);
```



DELETION

DELETING THE LAST NODE FROM A LINKED LIST

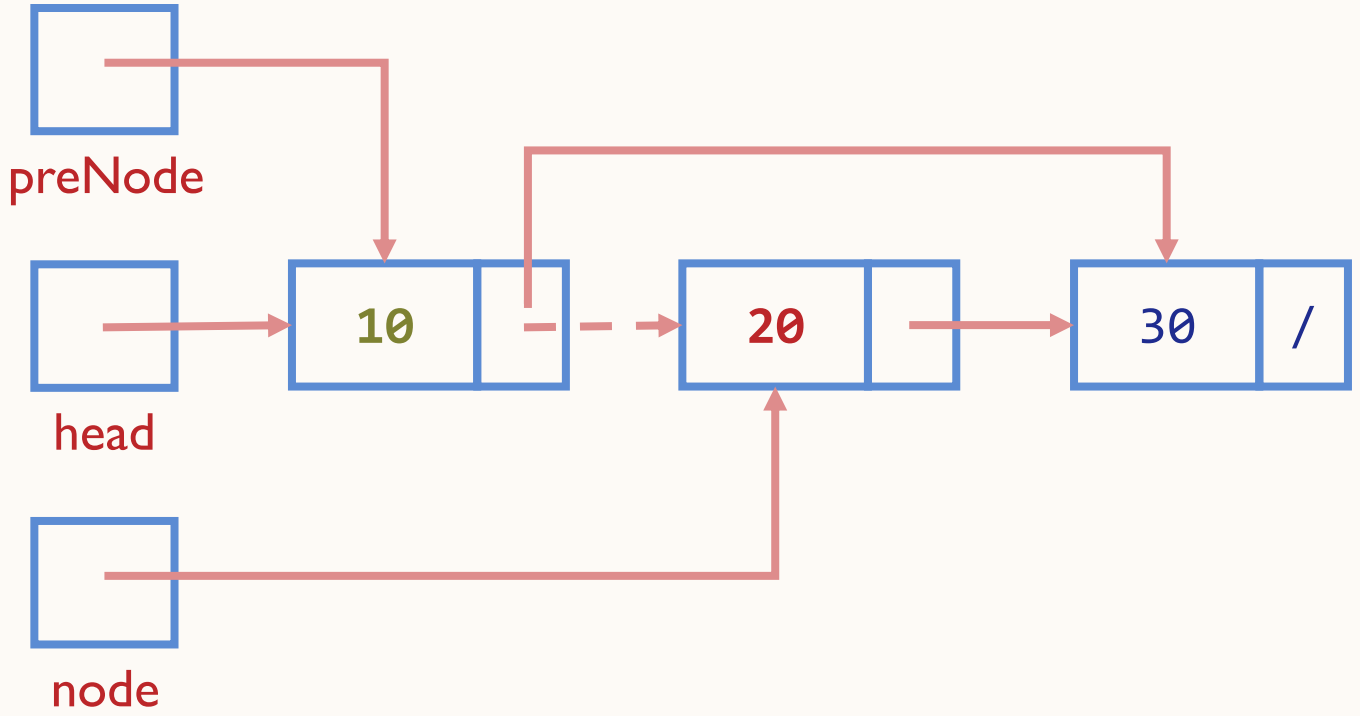
```
node = preNode = head;

while(node->next != NULL)
    node = node->next;

while(preNode->next != node)
    preNode = preNode->next;

preNode->next = NULL;
free(node);
```



DELETION

DELETING THE NODE AFTER A GIVEN NODE IN A LINKED LIST

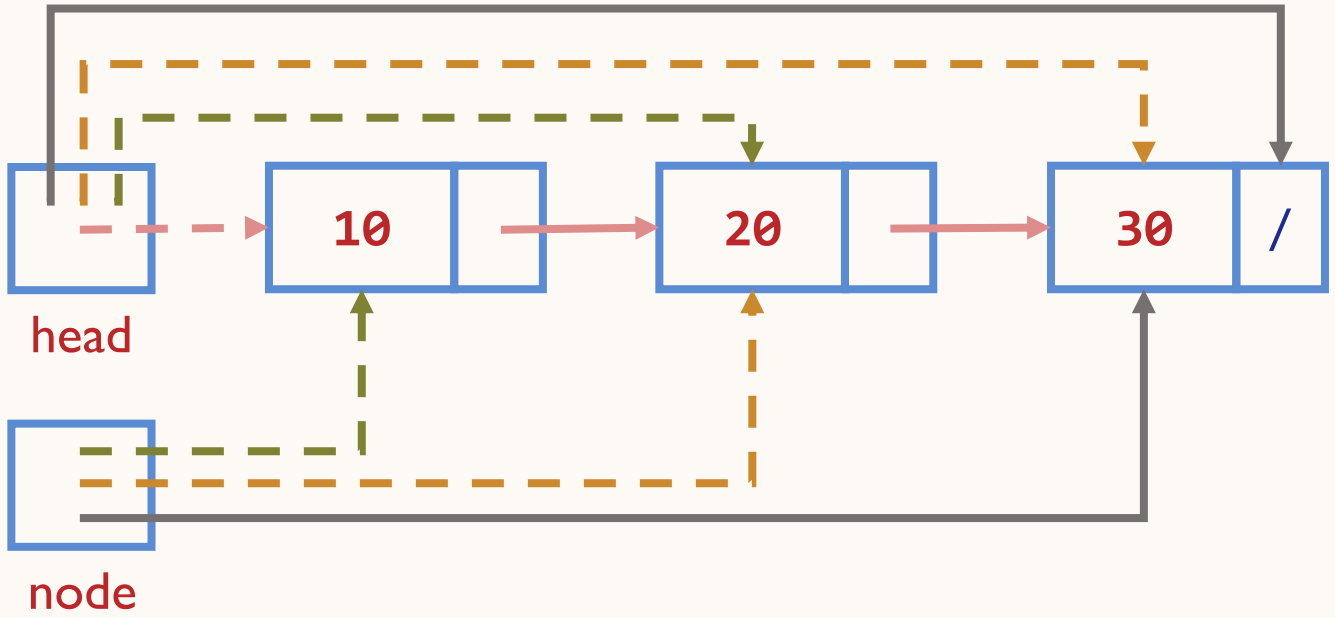
```
preNode = head;

while(preNode->data != 10)
    preNode = preNode->next;

node = preNode->next;
preNode->next = node->next;
node->next = NULL; //optional
free(node);
```



DELETION

DELETING THE ENTIRE LIST

```
while(head != NULL){
    node = head;
    head = head->next;
    free(node);
}
```



IMPLEMENTATION EXAMPLES

USING C FUNCTIONS

```
struct tnode * createList()
{
    struct tnode *head, *node;
    int number;
    head = NULL;
    while(scanf("%d", &number) == 1){
        node = (struct tnode *) malloc(sizeof(struct tnode));
        node->data = number;
        if(head == NULL)
            node->next = NULL;
        else
            node->next = head;
        head = node;
    }
    return head;
}
```

```
struct tnode *head;
head = createList();
```

IMPLEMENTATION EXAMPLES

USING C FUNCTIONS

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

```
struct tnode *head;
...
printList(head);
```

```
void deleteList(struct tnode **head)
{
    struct tnode *node;
    while(*head != NULL){
        node = *head;
        *head = (*head)->next;
        free(node);
    }
}
```

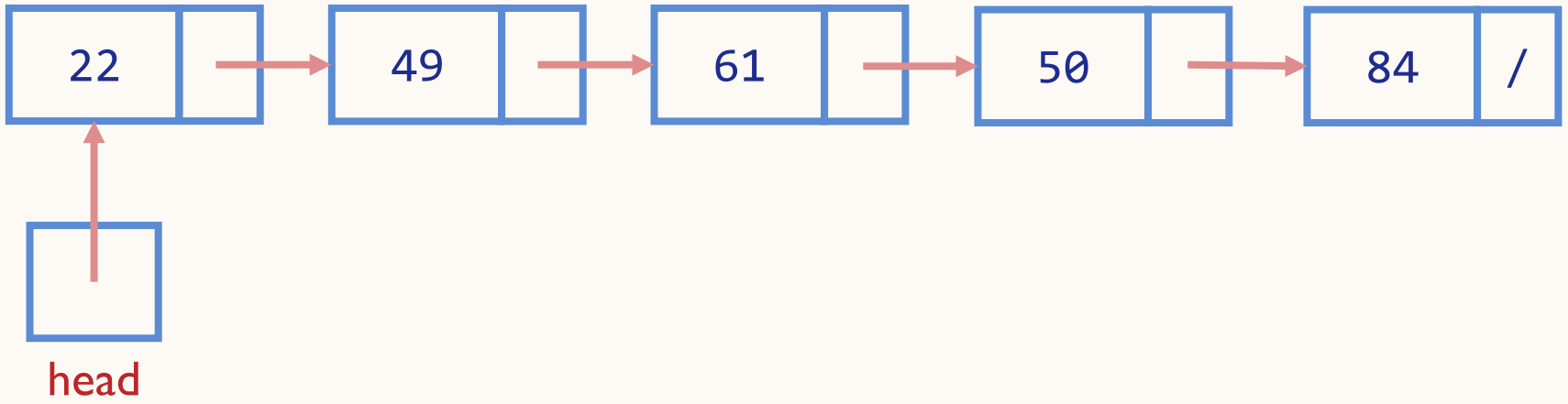
```
struct tnode *head;
...
deleteList(&head);
```

PRACTICE

EXERCISES

1. Write a program to sum the values of the nodes of a linked list and then calculate the mean.

2. Write a program that prints whether the given linked list is sorted (in ascending order) or not.

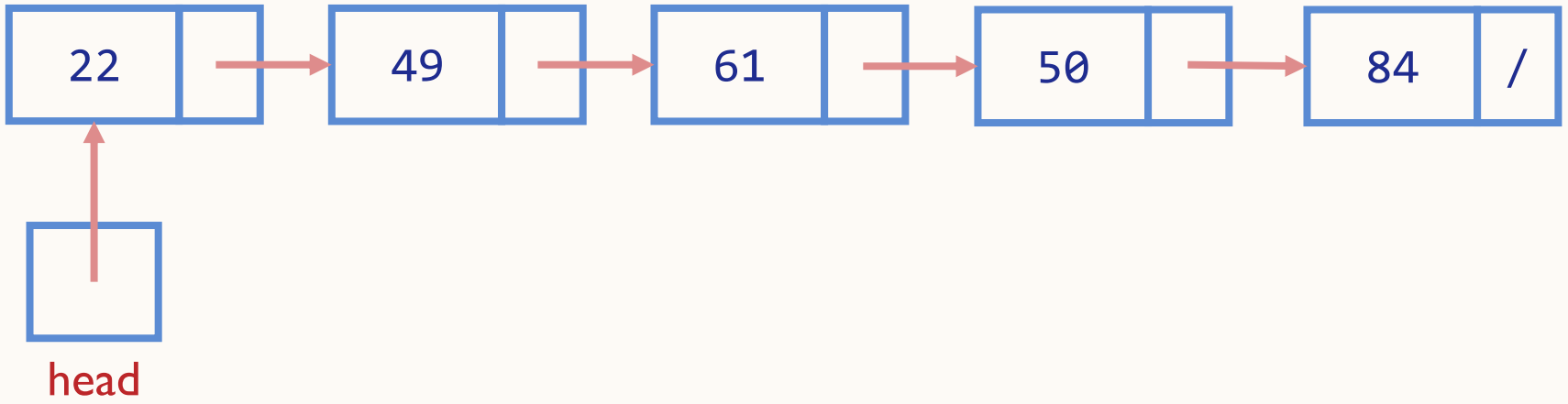


EXERCISES

3. What is displayed by the following code?

```
printf("%d",head->next->next->next->data);
```

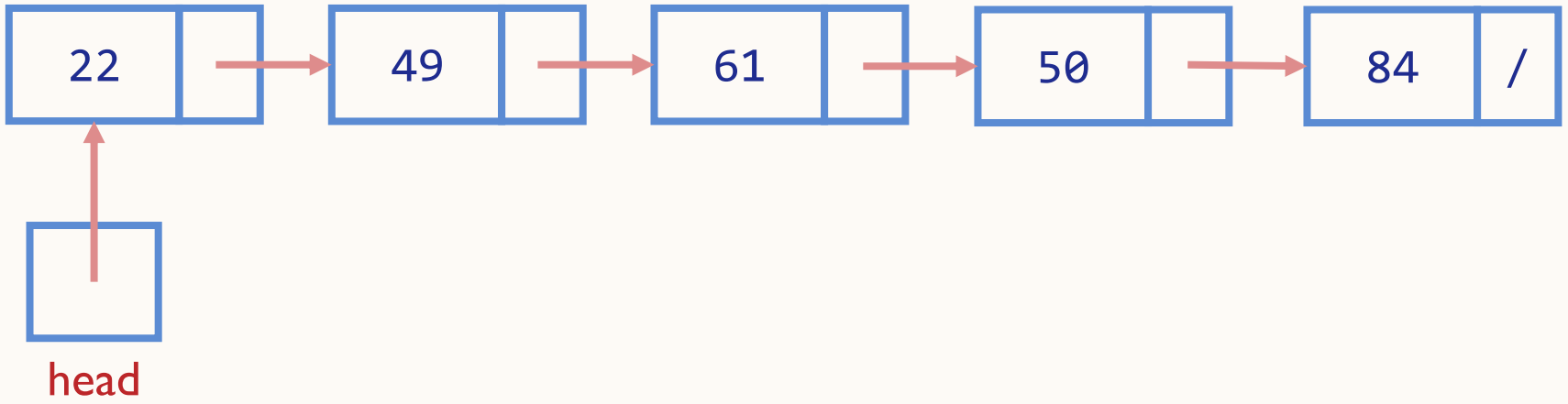
4. Write a program that prints minimum and maximum values in the linked list.



EXERCISES

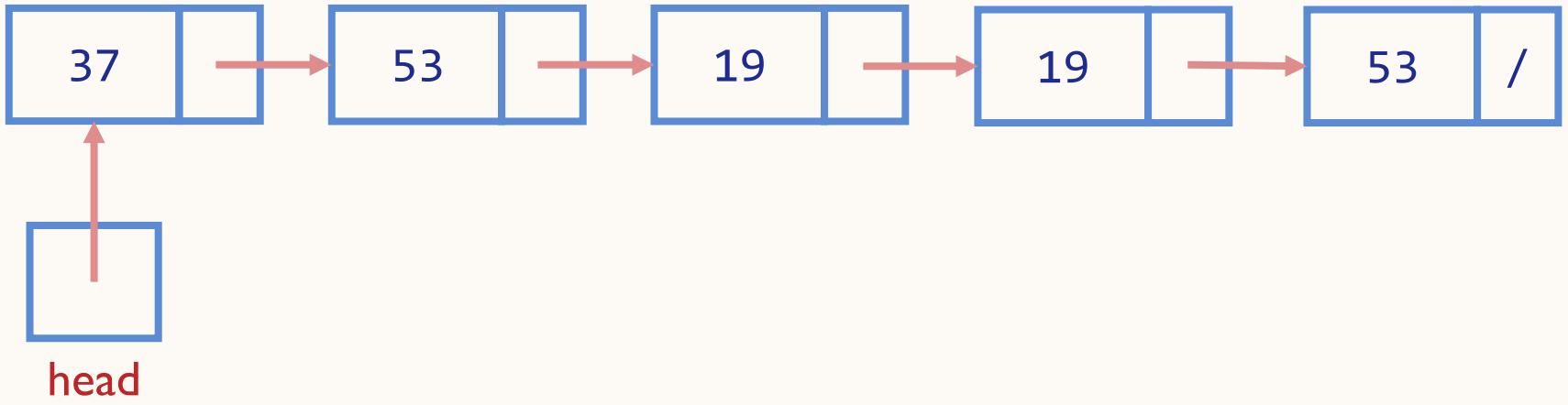
5. Write a program to make the first element of singly linked list as the last element of the list.

6. Write a program to reverse the list by interchanging the links and not the data.



EXERCISES

7. Write a program to delete the last occurrence of a given value.
8. Write a program that removes all nodes that have duplicate information.



REFERENCES

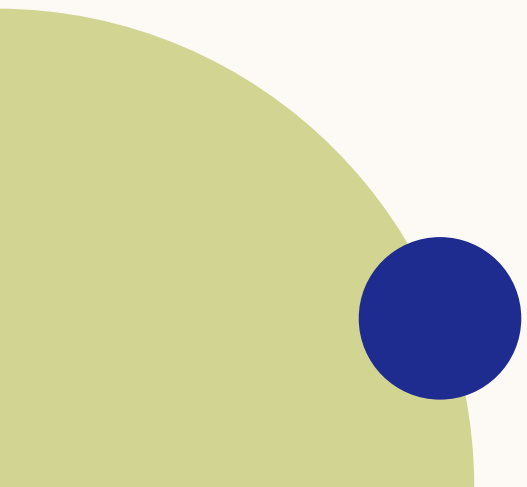
- Deitel, P. and Harvey Deitel (2022), C How to Program (9th Edition), Pearson Education.
- Thareja, R. (2014), Data Structures Using C (2nd Edition), India: Oxford University Press.

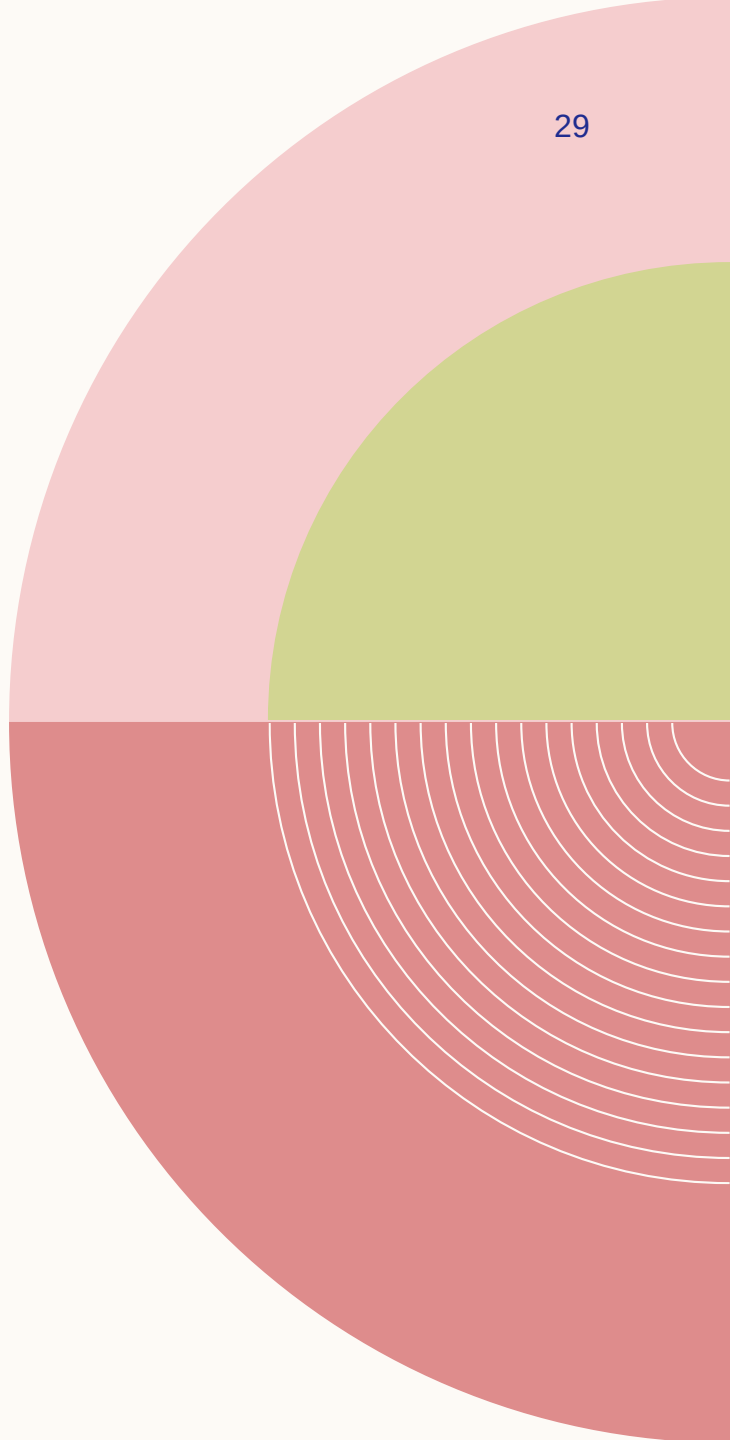
NEXT

Linked Lists:

Double Linked Lists

Circular Linked Lists





VISION

To become an **outstanding** undergraduate Computer Science program that produces **international-minded** graduates who are **competent** in software engineering and have **entrepreneurial spirit** and **noble character**.

MISSION

1. To conduct studies with the best technology and curriculum, supported by professional lecturer
2. To conduct research in Informatics to promote science and technology
3. To deliver science-and-technology-based society services to implement science and technology

Without hard work,
nothing grows but weeds.

UMN
UNIVERSITAS MULTIMEDIA NUSANTARA

INFORMATIKA UMN

Have patience.
All things are difficult before they become easy.