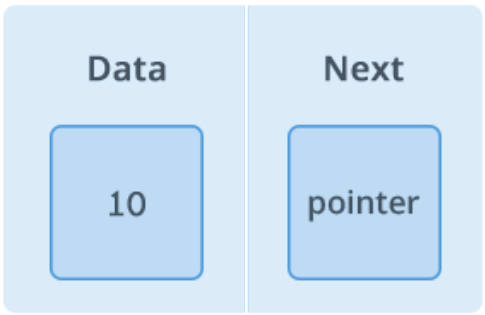
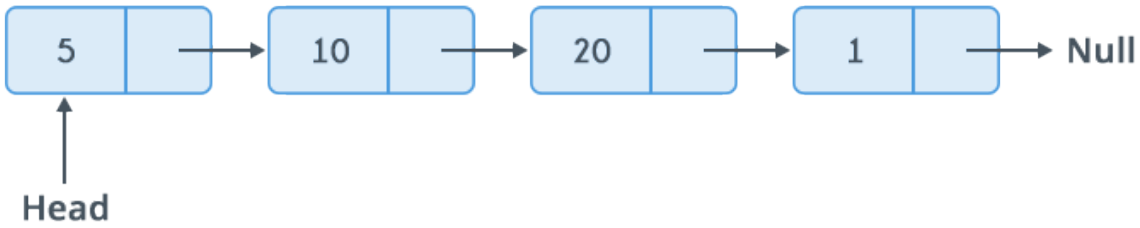
**Simple linked list:** Linked lists are the best and simplest example of a dynamic data structure that uses pointers for its implementation.




```c
/* LinkedList.c a simple linked list, modified – original
from "Wolf (2010) 'Grundkurs C'"*/


#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

// Globals
typedef struct Node {
    int data;
    struct Node *next;
} node_t;

typedef node_t *p_node_t;
p_node_t g_p_head = NULL;

// Function declaration
void new_node(void);
void insert_node(p_node_t new_node);
void remove_node_with_value(int val);
void list_nodes(void);
void search_node(int searched_data);
bool list_available(void);
```

```c
int main() {
    int choice, val;

    do {
        printf (" -1- Insert new node\n");
        printf (" -2- Remove node with value\n");
        printf (" -3- List all nodes\n");
        printf (" -4- Search node\n");
        printf (" -0- Exit\n");
        printf ("Choose an action: ");

        scanf ("%d", &choice);

        switch (choice) {
            case 1: new_node();
                break;

            case 2: printf ("Delete value: ");
                scanf ("%d", &val);
                remove_node_with_value ( val );
                break;

            case 3: list_nodes();
                break;

            case 4: printf ("Search value: ");
                scanf ("%d", &val);
                search_node ( val );
                break;

            default: if (choice !=0) {
                printf ("Wrong input - exit\n");
                choice = 0; // Exit program
            }
                break;
        }

    } while ( choice != 0 );

    printf ("Exit\n");

    return 0;
}
```

```c
// Function implementation
void insert_node(p_node_t new_node) {
    p_node_t p_help;

    if (g_p_head == NULL) {
        g_p_head = new_node;
        new_node->next = NULL;
    }
    else {
        p_help = g_p_head;
        while( p_help->next != NULL ) {
            p_help = p_help->next;
        }
        p_help->next = new_node;
        new_node->next = NULL;
    }
}
```

```c
void new_node(void) {
    p_node_t new_node = malloc(sizeof(node_t));

    if (new_node == NULL) {
        printf("No memory!?\n");
        return;
    }

    printf("Value for new node: ");
    do {
        scanf("%d", &new_node->data);
    }
    while( getchar() != '\n');

    insert_node(new_node);
}
```

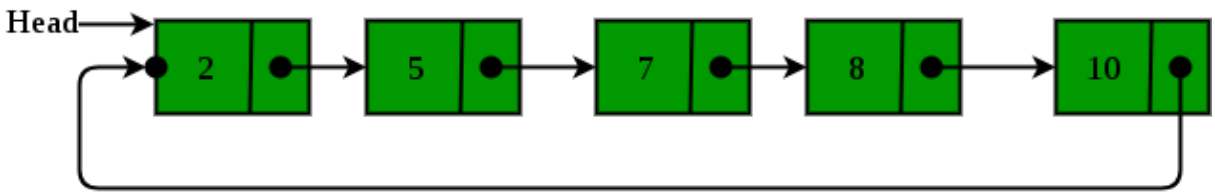
```c
void list_nodes() {
    // Check if there is a list
    if (list_available() == false) return;

    p_node_t p_help = g_p_head;

    while (p_help != NULL) {
        printf ("%d\n", p_help->data);
        p_help = p_help->next;
    }
}
```

```c
void remove_node_with_value(int val) {
    p_node_t p_help_1;
    p_node_t p_help_2;
    bool found = false;

    // Check if there is a list
    if (list_available() == true) {
        // Case if the searched data is in the head node
        if (g_p_head->data == val) {
            found = true;
            // Next node is the new head
            p_help_1 = g_p_head->next;
            g_p_head = p_help_1;

            return;
        }
        // Search the data in the rest of the list
        else
        {   // Start at the beginning
            p_help_1 = g_p_head;
            // Traverse the list till searched data is
found
            while (p_help_1->next != NULL) {
                p_help_2 = p_help_1->next;
                // Cut the node by connecting the previous
and the next node together
                if (p_help_2->data == val) {
                    found = true;
                    p_help_1->next = p_help_2->next;
                  // Free the memory of the removed node
                  free (p_help_2);
                    break;
                }
                p_help_1 = p_help_2;
            } // End while
        } // End else
    } // End if

    if (found == false) printf("Data %d not found!\n",
val);

}
```

```c
void search_node(int val) {

    // Check if there is a list
    if (list_available() == false) return;

    if (list_available()) {
    // Get the head node pointer
    p_node_t p_help = g_p_head;
    bool found = false;

    // Case the data is in the head node
    if (p_help->data == val) {
        printf ("Data %d found!\n", p_help->data);
        found = true;
        return;
    }

    // Traverse the list till data is found
    while (p_help->next != NULL) {
            p_help = p_help->next;

            if (p_help->data == val) {
                printf ("Data %d found!\n", p_help->data);
                found = true;
                return;
            }
    }

    if (found == false) printf("Data %d not found!\n", val);
    }
}
```

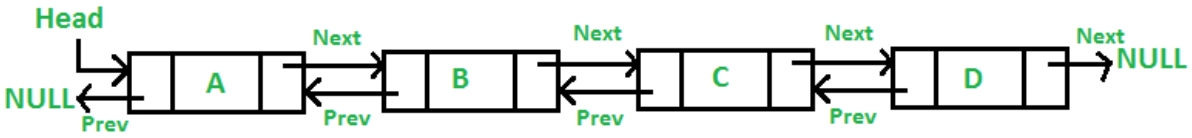
```c
bool list_available() {

    // Check if there is a list
    if (g_p_head == NULL) {
        printf("No list,\nyou have to create a node first!\n");
        return false;
    } else {
        return true;
    }
}
```

**Circular linked list:**



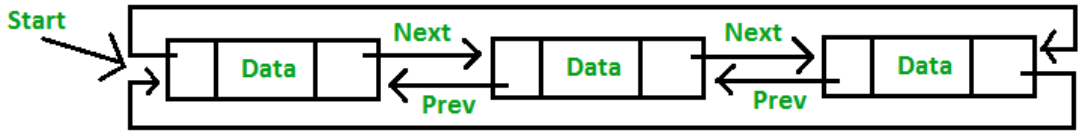

**Task:** Write a circular linked list with add node - , delete node - and search node functionality! Write a small user interface to let the user choose between these functionalities!

**Doubly linked lists:**




**Task:** Write a doubly linked list with add node at beginning - , delete node - , search node forward/backwards functionality! Write a small command-line interface to let the user choose between these features and print the output.

**Circular doubly linked list:**




**Task:** Write a circular doubly linked list with add node (after a given node) -, delete node -, search node forward/backwards functionality! Write a small command-line interface to let the user choose between these features and print the output.

Study and implement at least one of these cases. Use `typedef` for creating your node chains and take these node types for your function parameters and - outputs.