PROGRAMMAZIONE PROCEDURALE

A.A. 2021/2022
LINKED LISTS
LINKED LIST

What are the problems with arrays?
✓ Size is fixed
✓ Array items are stored contiguously
✓ Insertions and deletions at particular positions is complex

Why linked lists?
✓ Size is not fixed
✓ Data can be stored at any place
✓ Insertions and deletions are simpler and faster
WHAT IS A LINKED LIST

A linked list is a collection of nodes with various fields:
- Data fields
- Link fields
REPRESENTATION AND OPERATIONS OF SINGLY LINKED LISTS
```c
struct Node {
    int info;
    struct Node* pNext;
}

typedef struct Node Node
```

```c
struct Node {
    int info1;
    char info2;
    struct data info3;
    struct Node* pNext;
}
```
int main() {
    Node *p = (Node*) malloc(sizeof(Node));
    ...
    p = 1000
    ...
}
int main() {
    Node *p = (Node*) malloc(sizeof(Node));
    p->info = 3;
    p->pNext = NULL;
    ...
}

heap

info = 3
pNext = NULL

stack

p = 1000
...
SIDE STRUCTURES

- One pointer to the first element of the list
- One pointer to the last element of the list (optional)

Node* pFirst

Node* pLast

10  1000

30  4000

5   NULL
MOST COMMON OPERATIONS

- Print all the elements

- Insertion
  - Head
  - Tail
  - At a given position

- Deletion
  - Head
  - Tail
  - At a given position
// As a parameter, it takes the pointer to the first node of a list
void print_list(Node* pFirst)
{
    if (pFirst == NULL) // No node in the list
    {
        printf("No node in the list!");
    }
    else
    {
        // New pointer used to scan the list.
        Node* pScan = pFirst;
        do
        {
            printf("Info: %d\n", pScan->info);

            // ptrScan is updated to point to the next node in the
            // list
            pScan = pScan->pNext;
        }while(pScan != NULL); //NULL when this was the last node
    
return;
}
HOW IT RUNS

Node* pFirst

pFirst == NULL? No
pScan = pFirst 7000
printf pScan->info 10
pScan = pScan->pNext 1000
pScan != NULL? Yes
printf pScan->info 30
pScan = pScan->pNext 4000
pScan != NULL? Yes
printf pScan->info 5
pScan = pScan->pNext NULL
pScan != NULL? No
return

Info: 10
Info: 30
Info: 5
HEAD INSERTION

// Node* pFirst is a pointer to the first node of a list (global)
void head_insertion(void)
{
    // Creation of a new node in the heap
    Node *pNew = (Node*) malloc(sizeof(Node));
    scanf("%d", &pNew->info);
    pNew->pNext = NULL;

    if(pFirst == NULL) // No node in the list
        pFirst = pNew;  // The first node is the newly created one
    else
    {
        // Else, there is already at least one node in the list
        pNew->pNext = pFirst; // the first node becomes the second one
        pFirst = pNew;        // The first node is the newly created one
    }

    return;
}
Node *pNew = (Node*) malloc(sizeof(Node));
scanf("%d", pNew->info);
pNew->pNext = NULL;

pFirst == NULL? Yes
pFirst = pNew 3500
Node *pNew = (Node*) malloc(sizeof(Node));
scanf("%d", pNew->info);
pNew->pNext = NULL;

Node* pFirst

pFirst == NULL? No
pNew -> pNext = pFirst
pFirst = pNew

3500
3 7000

10 1000
30 4000
5 NULL
HEAD DELETION

// Node* pFirst is a pointer to the first node of a list (global)
void head_deletion(void)
{
    if(pFirst == NULL) // No node in the list
    {
        printf("No node in the list!");
    }

    else
    {
        // Else, there is at least a node in the list

        // Remember the pointer to the second node, which will become
        // the first one
        Node* temp= pFirst->pNext;

        // Memory is deallocated (node canceled from memory)
        free(pFirst);

        // The first node becomes the former second node
        pFirst= temp;
    }

    return;
}

**HOW IT WORKS**

pFirst == NULL? No

temp = pFirst -> pNext 1000

free(pFirst) 7000

pFirst = temp 1000
ADDING AND REMOVING FROM TAIL

- When we need to add or remove a node from the tail, it is useful to also have a pointer to the last node of a list.
  - It simplifies writing these two functions.
- This pointer (pLast) needs to be updated at the end of these operations.
- Of course if we need to have add-or-remove from head and add-or-remove from tail, pLast needs to be updated also in add-or-remove from head (as pFirst in previous examples).
// Node* pFirst is a pointer to the first node and Node* pLast to the last node of
// a list (both global)
void tail_insertion(void)
{
    // Creation of a new node in the heap
    Node *pNew = (Node*) malloc(sizeof(Node));
    scanf("%d", &(pNew->info));
    pNew->pNext = NULL;

    if(pFirst == NULL) // No node in the list
        pFirst = pNew;  // The first node is the newly created one
        pLast = pNew;   // The last node is the newly created one
    else
    {
        // Else, there is already at least one node in the list
        pLast->pNext = pNew; // the last node becomes the second one
        pLast= pNew; // The last node is the newly created one
    }

    return;
}
HOW IT WORKS

Node* pFirst

Node* pLast

pFirst == NULL? No
pLast->pNext= pNew 500
pLast= pNew 500
TAIL DELETION

void tail_deletion() {

    if(pFirst == NULL)
        printf("No node in the list!\n");
    else {
        Node* pPrev = NULL;
        Node* pScan = pFirst;

        if(pScan->pNext == NULL) { // It means we only have one node in the list
            free(pScan); // Free memory
            pFirst = NULL; // Now the list is empty
        }
        else { // Otherwise, I need to scan the list until I find the last node (pLast)
            do{
                if((pScan->pNext) == pLast) { // Reached the node before the end
                    pPrev = pScan;
                    break;
                }
                else
                    pScan = pScan->pNext; // Otherwise, I need to iterate
            }while((pScan->pNext) != NULL);

            free(pPrev->pNext); // Free memory allocated to the last node
            pPrev->pNext = NULL; // pPrev becomes the last node (no node after it)
            pLast = pPrev; // pPrev becomes the last node
        }
    }
}
HOW IT RUNS

pFirst == NULL? No
pScan = pFirst 7000
pScan->pNext == NULL? No
pScan = pScan -> pNext 1000

Node* pFirst

Node* pLast

10 1000

30 NULL

5 NULL

(pScan->pNext) == pLast  Yes
pPrev= pScan 1000
free(pPrev->pNext) 4000
pPrev -> pNext= NULL
pLast = pPrev 1000
`void delete_if_equal(Node* pFirst, Node* pLast, int key){...}

e.g., 30

Update pFirst or pLast if instead you remove either the first or last node
ADD NODE IN SOME POSITION

Node* pFirst

Node* pLast

Homework!
DIFFERENT KINDS OF LISTS

- Singly linked lists
- Circular singly linked list
- Doubly linked lists
- Circular doubly linked list
SINGLY LINKED LIST

Pointer to the first element

- 7000
- 10, 1000
- 30, 4000
- 5, NULL

- Stored at memory address 7000
- Stored at memory address 1000
- Stored at memory address 4000
CIRCULAR SINGLY LINKED LIST

Pointer to the first element
DOUBLY LINKED LIST

null

7000

10

1000

7000

50

5000

7000

50

5000

1000

30

null

Stored at memory address 7000

Stored at memory address 1000

Stored at memory address 5000