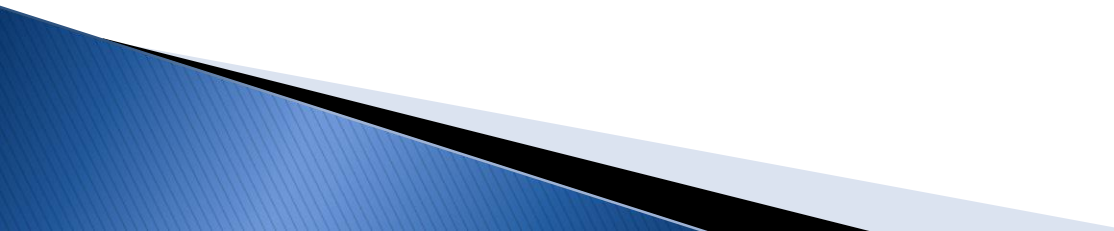


Data structures and algorithms – Lab 7

Iulia-Cristina Stanica
iulia.stanica@gmail.com

Roadmap – Lists

- ▶ Dynamic allocation
 - ▶ Studying the Linked List abstract data type (ADT) and its different variations
 - ▶ Applications with lists
- 

Dynamic allocation

- ▶ **Dynamic allocation:** memory allocation which happens at runtime.
- ▶ Operators: new and delete
- ▶ Use: when we do not know from the beginning how much memory we will need
- ▶ Syntax: new **data-type**;
- ▶ More info:
http://www.tutorialspoint.com/cplusplus/cpp_dynamic_memory.htm

Dynamic allocation – example

```
#include <iostream>
using namespace std;
int main ()
{
    int* pvalue = NULL; // Pointer initialised with NULL
    pvalue = new int;    // Ask memory for the variable
    *pvalue = 29495;     // Save the value at the allocated
address
    cout << "Value of pvalue : " << *pvalue << endl;
    delete pvalue;      // Free up memory
    return 0;
}
```

Dynamic allocation – arrays

```
#include <iostream>
using namespace std;
int main ()
{
    int n; cin>>n;
    int* a = NULL; // Pointer initialised with NULL
    a = new int [n]; // Ask memory for the variable
    for (int i=0; i<n; i++) {
        a[i] = 0; // Initialise elements of the array
    }
    delete [] a; // Free up memory
    return 0;
}
```

Dynamic allocation – objects

```
#include <iostream>
using namespace std;
```

```
int main()
{
    ClassName* myArray = new ClassName[4];
    delete [] myArray ; // Delete array

    return 0;
}
```

List

- ▶ Instance of an ADT
- ▶ In C++ (stl): `list` / `slist`.
- ▶ Ordered collection of entities.
- ▶ Content of each element from the list:
 - info
 - pointer(s) to one or more nodes from the list

Basic operations

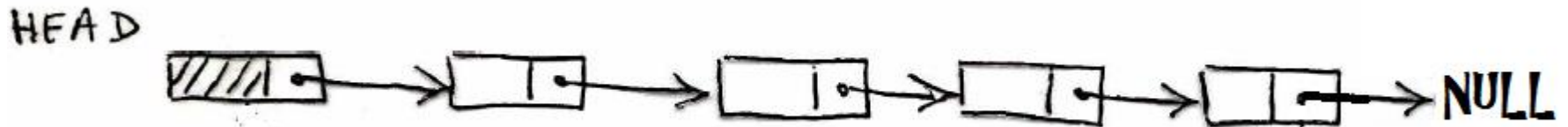
- **Add** – adds an element (entity) to the list: at the beginning, at the end or at an arbitrary position
- **Remove** – removes an element (entity) from the beginning/end of the list or taken into account its index/content
- **Get** – consults an element taken into account its index
- **Update** – updates the information/content of an element
- ▶ **OBS: list properties:**
 - Length (size) – the nr of elements in the list (in a function getSize())
 - Type – the type of elements in the list

Implementation

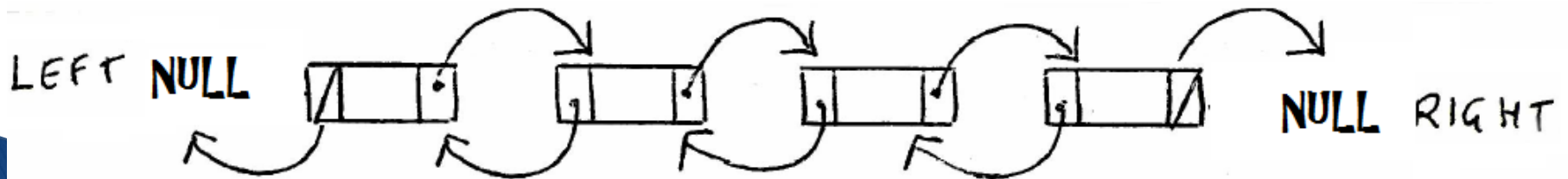
- Each node contains :
 - info (content)
 - link (pointer):
 - to its neighbours (doubly linked lists)
 - to the following element (singly linked lists)
 - last element linked to the first one (circular lists)
- Nodes are **dynamically allocated**, so we can obtain lists of a size limited only by the program memory
- More info about dynamic allocation:
http://www.tutorialspoint.com/cplusplus/cpp_dynamic_memory.htm

List types (1)

- singly-linked linear lists

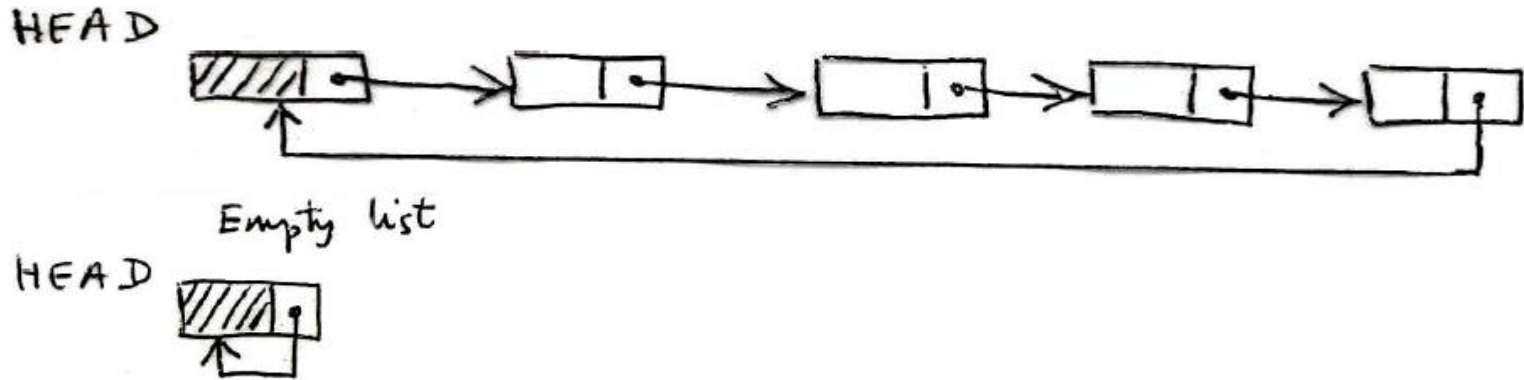


- doubly-linked linear lists

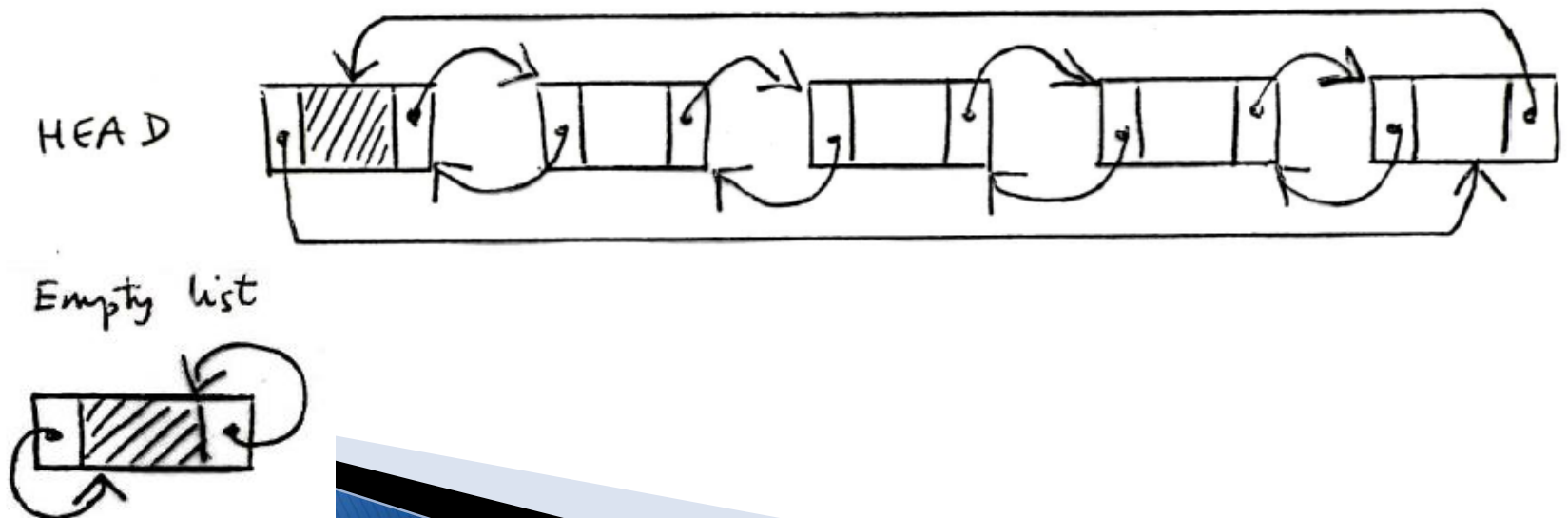


List types (2)

- ▶ singly-linked circular lists



- doubly-linked circular lists



Exercises

- Ex 1) Test the doubly linked list header (list1.h – fils.curs.pub.ro). Add a **printList()** method to the header in order to display the content of each node.
- Ex 2) Considering the linked list, make two lists such that the first one contains only the odd numbers which appear in the initial list and the second one contains the even numbers of it. Add these functions to the header. Display the two new lists
- Ex 3) Use the implementation from Ex. 1 and implement a queue using lists.
- Ex 4) Modify the header list1.h to obtain a doubly-linked circular list.
- 