Linked lists
Tutorial 5b

Katja Mankinen

6 October 2017

Aim

Pointers are one of the most powerful tools in C++: with pointers, you can directly manipulate computer memory. However, at first glance they may be somewhat confusing and it may not look obvious why they are even needed.

The aim of the tutorial is to make you familiar with pointers, step by step, and provide you one very practical and useful example why pointers are needed using a data structure called **linked list**.

After this tutorial, you

- can declare and use pointers
- i.e. can manipulate computer memory directly!
- have build a linear data structure, linked list

You should preferably finish all the steps today. It may look like a lot of work, but it is actually very doable! You can immediately start doing the exercises at your own pace. Note: You will need pointers in future, so take your time and finish all the exercises yourself instead of just reading the solutions! If you have any problems or questions, just ask!

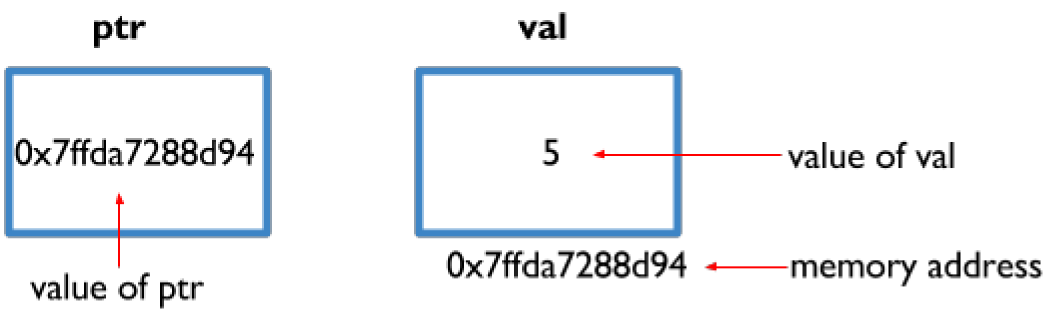
1. Warm up: Where are you pointing?

Let's start by refreshing our memory about the syntax of pointers:

- pointer points to another location of memory. Pointers hold addresses of memory locations.
- this address can be seen by using the address-of operator & in front of the variable

```
#include <iostream>

int main(){
  int val = 5;
  int* ptr; //pointer to int
  ptr = &val; // now ptr is equal to the address of variable val
  std::cout << val << " " << *ptr << " " << &val << std::endl;
  // 5 5 0x7ffda7288d94
}
```



Quick Quiz Time (correct answers at the end of this document)

A. How do you declare a pointer to an integer?

1. double* ptr;
2. int* ptr;
3. int& ptr;

B. When the pointer is declared, how do you set it to point to `int a = 5;`?

1. ptr = &a;
2. ptr = *a;
3. *ptr = &a;

C. How to deference this pointer?

1. &ptr = 10;
2. *ptr = 10;
3. &a = 10;

D. What does the variable i contain after the following code executes?

```
int i = 27;
int* p = &i;
*p = 102;
```

1. i = 27
2. i = 0
3. i = 102

2. Linked List

Now you are comfortable and familiar with the syntax of pointers, and are ready to put your knowledge to good use. Pointers are used mainly for managing data, accessing class member data and functions (you will understand what *classes* are in the next lecture), and passing variables by reference to functions.

Today we will play with a data structure called a **linked list**! Linked lists are the best example of a dynamic data structure that uses pointers for its implementation. Like arrays, a linked list is a linear data structure and its purpose is to store collections of data. However, unlike arrays, linked list elements are not stored at contiguous location. A linked list allocates space for each element in its own block of memory called *a node*. These elements are linked together using pointers. When using linked lists, elements must be accessed sequentially starting from the first node.

Each node contains two fields: a *data* field to store an element type, and a *next* field which is a pointer used to link one node to the next node. The entry point into a linked list is called the head of the list. It should be noted that head is not a separate node, but the reference to the first node. The first node contains a pointer to the second node. The second node contains a pointer to the third node, and so on. The last node in the list has its next field set to NULL to mark the end of the list.

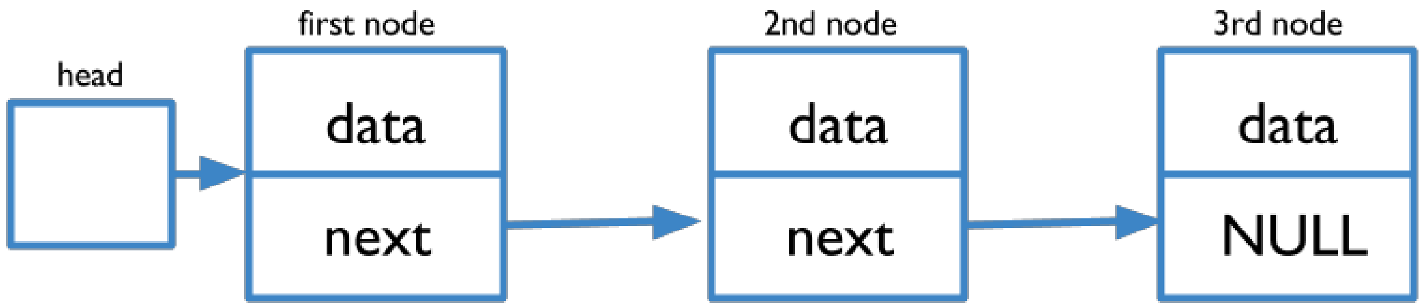


Figure 1: Schematic example of a linked list, consisting of three nodes

1. Exercise: Implement a node structure

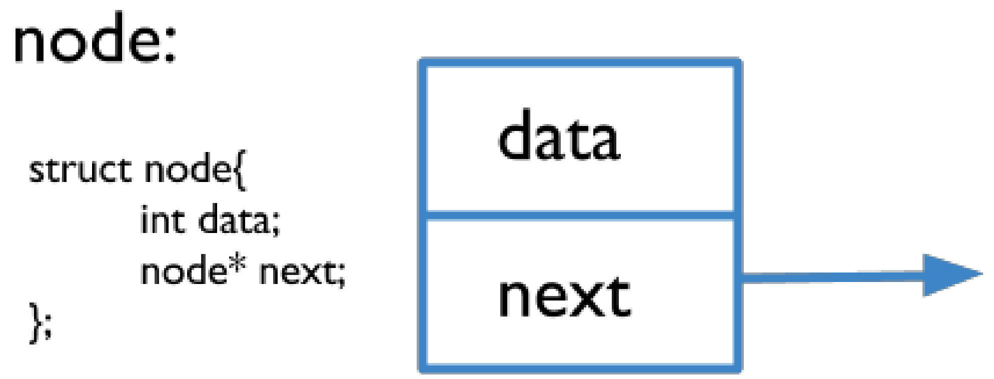
Finally, let's start our exercises after a long reading! Today you will implement a linked list data structure and some methods to it. Make sure that all program printouts make sense and are correct on your opinion. **If you get stuck at any point, just raise your hand and one of the teachers will come to help you!**

Make a new `.cpp` file, and include basic libraries and use namespace `std`. Then, first thing we want to cover is how to make one of those nodes. In C++, we can use `struct` to make a new structure type. Implement a typical node structure to your code as follows. Each node contains a single integer data element and a pointer to the next node in the list.

```
struct node{
    int data; //a single data element of a type int
    node* next; //creates a node pointer, called "next", to the next node
};
```

This declares a new structure type, called `node`, and defines it having two members: data and next. Now you can use `node` just like any other type (`int, double, char`...). [1]

Basically you just created a new struct that looks like the following:



Make a `main` function of a type `int` after your new node structure. In your main, declare a new node pointer:

```
node* linkedList;
```

If this seems too difficult, remember what `int* pointer` means. We have the same idea here, but now just instead of int, we have your newly made `node` pointer that can point to another node.

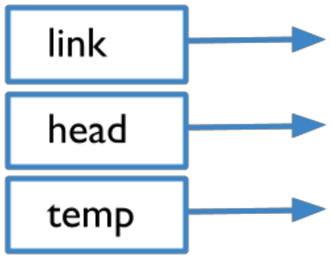
2. Exercise: Create a new node object

Let's start by creating new node pointers that can point another nodes.

[1] More information on structures in C++: http://www.cplusplus.com/doc/tutorial/structures/

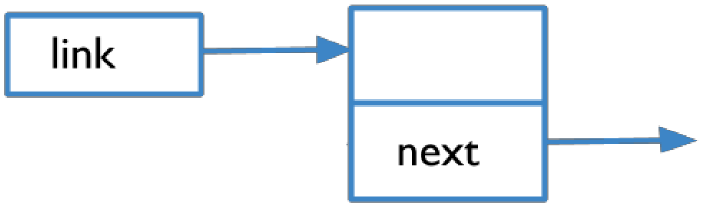
```
node* link;
node* head; //beginning of the list
node* temp; //temporary node pointer
```

It looks somewhat the following:



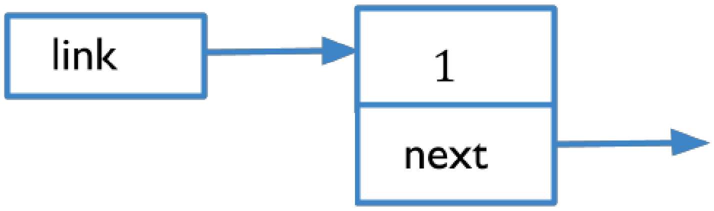
Now we've created our pointers. Let's use our `node* link` to point a new node:

```
link = new node; //it does not make "link" to be a new node, but it points to the newly
    created node
```



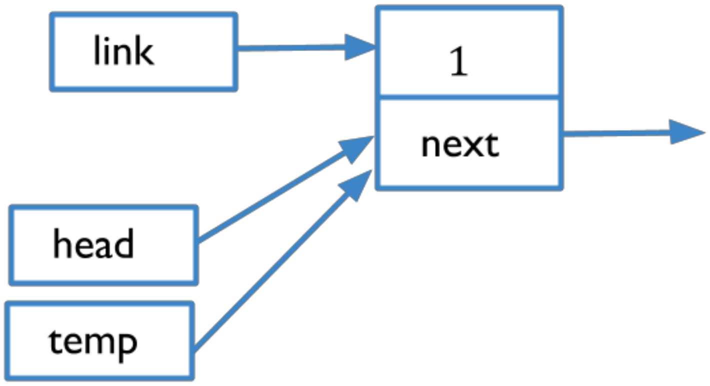
If we want to fill up our node with some data, we can just write

```
link->data = 1; //accessing a data member of this node, and setting its value to 1
```



In addition, we want our `temp` and `head` pointers to point the same node where `link` is pointing to:

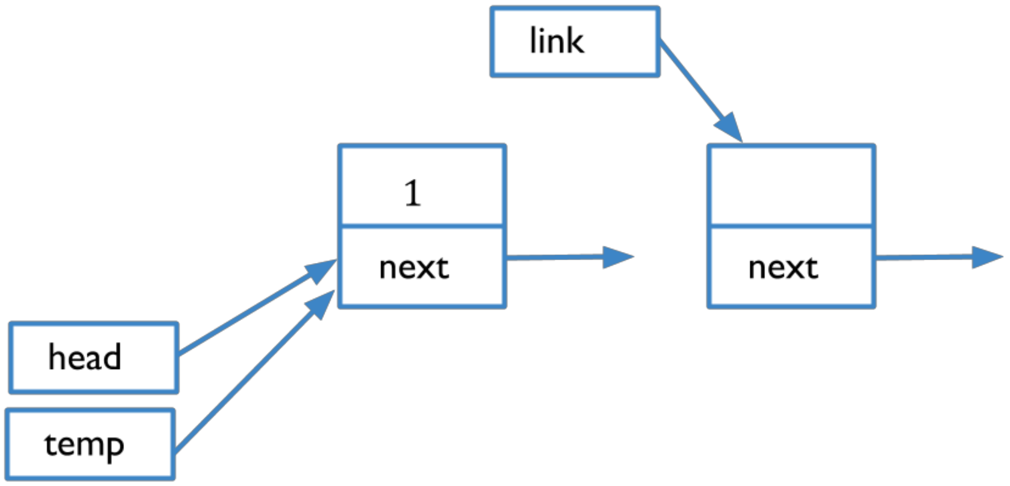
```
temp = link;
head = link;
```



In order to have access to our first node all the time, our **head** will stay here for the whole time. `temp` will follow `link`.

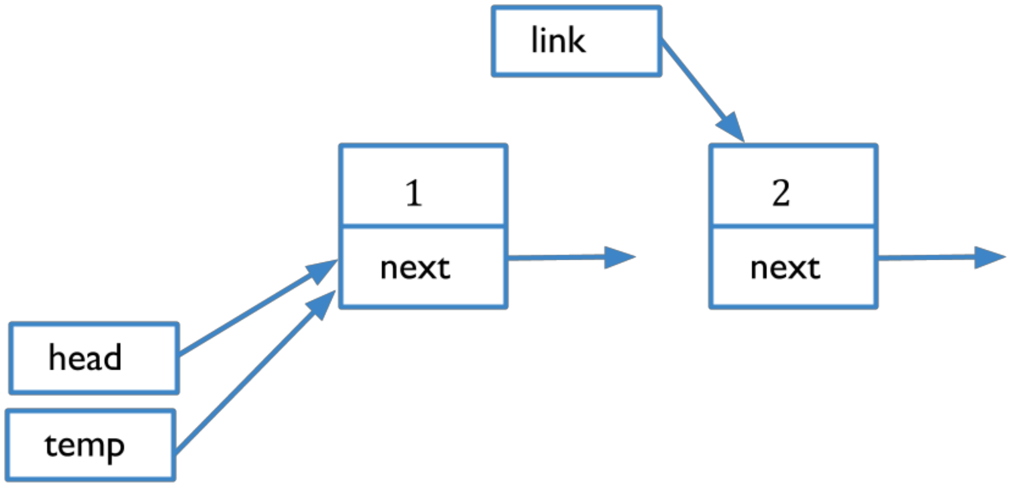
Next thing we want to do is to build our list. Let's change what our `link` pointer is pointing to. We want it to point to a new node.

```
link = new node; //again, this does not make "link" to be a new node, but it points to a new
    node
```



Now we have two nodes, but the list is not linked yet. We have a "link" pointing to the new node. That node has a "next" pointer, and we can fill some data there.

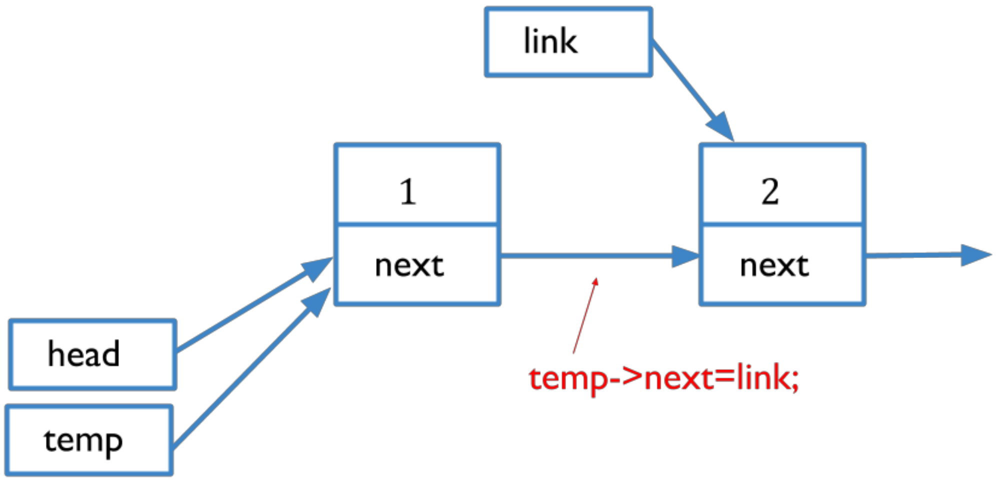
```
//Your implementation comes here. How to fill data to the new node?
//Access a data member of the new node, and set its value to 2.
```



How to link these two nodes together? We can use our `temp` pointer:

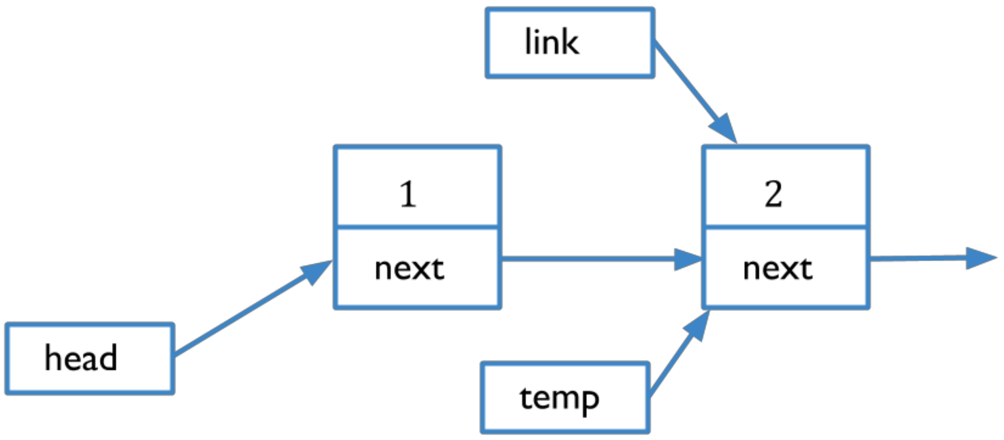
```
temp->next = link;
```

In human language that would read somewhat like "look at the pointer `temp` is pointing to, and access `next` data member inside that node, and make it pointing to whatever `link` is pointing to."



So basically we just extended the arrow between the nodes; now we're pointing to the second node. Then we want to move `temp` to point the same node as link.

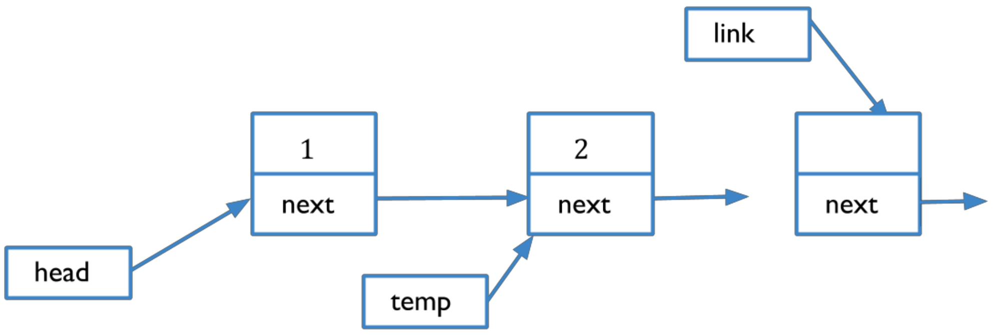
```
//Your implementation comes here. How to move temp to point to the same node as link?
```



Now the list consists of two nodes that are linked together.

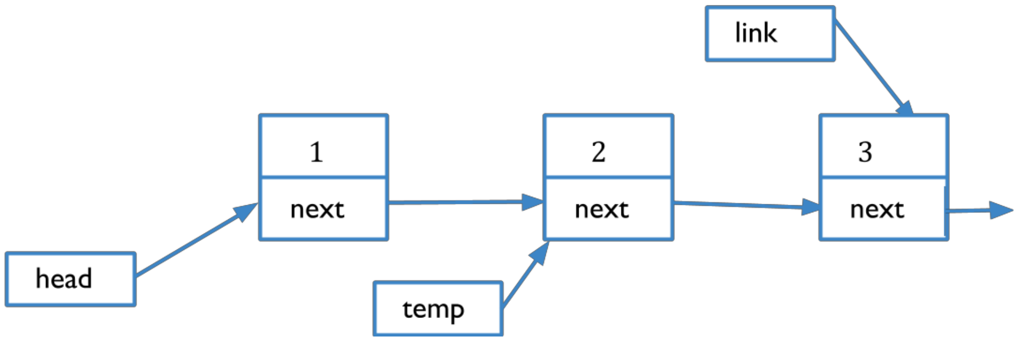
Let's make another node in a similar manner. Change `link` to point to a new node

```
//Your implementation comes here. How to make "link" to point to a new, third node?
```



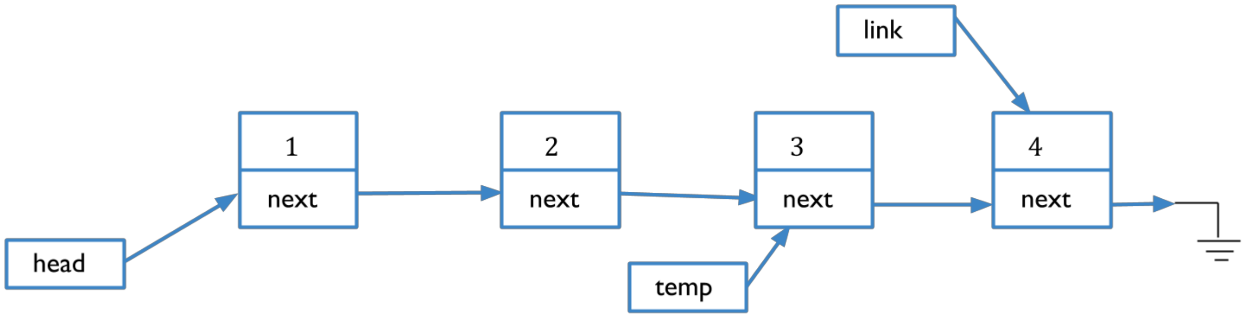
Fill the new node with some data, and connect the second node to the third node.

```
//Your implementation comes here.
//Fill the new node with data and set the value to 3.
//Connect the second node to the third one.
```



Finally, let's make our last node. Implement the structure implied by the following schematic graph.

```
//Your implementation comes here.
//Change "link" to point to a new, fourth node
//Fill the new node with data and set the value to 4
//Make "temp" point to the third node
//Connect the third node to the fourth one
```



We want to finish our list here. We can simply do it by saying

```
link->next = NULL; //now "next" does not point anything.
```

Congratulations! You've created a linked list that is connected by the nodes. We're keeping track with our `head` pointer, so we can always go back to the start.

3. Exercise: Functionalize it all!

Now you're really familiar with the concept and how everything works. However, when you're really working as a scientist and accessing maybe hundreds of nodes, you cannot write all the nodes and their contents by hand as we just did.

Go to Live@Lund and download `linkedList.cpp`. In this last exercise, you will implement some methods to a linked list data structure. The file `linkedList.cpp` contains a simple implementation of a linked list. It also contains main body of the program where functionality of linked list implementation is tested. Some functions are not fully implemented and it is your task to finish them! All tasks and questions marked with TODO keyword in comments to the program should be addressed.

Make sure that all program printouts make sense and are correct on your opinion.

```
g++ -o linkedList linkedList.cpp
./linkedList
```

Hints!

1. Don't panic!
2. Test your code after every change you make.
3. If you get stuck at any point, just raise your hand and one of the teachers will come to help you!

3. Summary

Linked list has a few advantages over arrays:

- The size of an array is fixed. There is no need to define an initial size for linked lists.
- Inserting a new element into an array is costly, because new elements need some space and to do so, existing elements have to be shifted. Using linked lists, items can be added or removed from the middle of the list

But why all this hassle about linked lists?

A linked list is a sequential data structure, where each element can be accessed only in particular order. We can use linked lists to implement more complex data structures such as *stacks, queues,* and *graphs.* From these structures we can go forward and extend our knowledge to *search trees* and *decision trees* and *machine learning* and so on, until the very edge of the computer science! Unfortunately, this is out of the scope of this course. Go ahead and google some examples of these fancy words!

You can also study more about linked lists, just use Google! There are many very good and educational videos about them on Youtube as well.

4. Answers to quiz

A2: When we declare a pointer, it does not contain the value it is assigned to, but the address of the value instead.

B1: To assign an address, you must add & to the variable whose address will be pointed to.

C2: Dereferencing takes the contents of ptr and sets the value in that location to 10.

D3: This was just to summarize the questions and answers above.