

OUTLINE

1 CHAPTER 11: C++ LINKED STRUCTURES

- A C++ Linked Structure Class
- A C++ Linked List
- C++ Linked Dynamic Memory Errors
- In-class work

A LISTNODE CLASS

- To support a Linked List container class `LList`, a `ListNode` class is used for the individual nodes.
- A `ListNode` object has two attributes: `item` and `link`.
- Public access is allowed for these attributes– the only class using the `ListNode` class is the `LList` class.

A LISTNODE CLASS

In Python:

- Data in a node can be of any type—a linked list can be heterogeneous.
- All values are references; the link attribute need not be declared to be a pointer.
- A link with the **None** value is used to indicate the end of a list.

A LISTNODE CLASS

In C++:

- A **typedef** statement allows the type of data to be specified at compile time. (The *Linked List* will still be homogeneous, but at least a different version of the class for another type can be compiled for another program by simply changing the **typedef** statement).
- The **item** attribute must be declared to be a particular type (for now).
- The **link** attribute must be declared to be a pointer to another **ListNode**.
- A link with the **NULL** value (a C++ pointer set to 0) is used to indicate the end of a list.

A LISTNODE CLASS

In C++:

- Later, we will see that the C++ Standard Template Library allows homogeneous lists for different data types (say, a list for the `int` type and a list for the `Rational` type) to be written at the same time, using a `template` class.
- Homogeneous lists for different data types can then be declared and used in the same program.
- The linked lists of these types will still be homogeneous. A list of integers can coexist with a list of doubles, but there can be no list containing integers and doubles mixed together.

HEADER FILE: LISTNODE.H

```
typedef int ItemType;

class ListNode {

    friend class LList;

public:
    ListNode(ItemType item, ListNode* link=NULL);

private:
    ItemType item_;
    ListNode *link_;
};
```

HEADER FILE: LLIST.H

The main differences between writing Python and C++ linked structure classes:

- the need to write a **destructor**, **copy constructor**, and **assignment operator** for the class
- our C++ class must also explicitly deallocate memory (not required by Python)

HEADER FILE: LLIST.H

```
class LList {
public:
    LList();
    LList(const LList& source);
    ~LList();

    LList& operator=(const LList& source);
    int size() return size_;
    void append(ItemType x);
    void insert(size_t i, ItemType x);
    ItemType pop(int i=-1);
    ItemType& operator[](size_t position);
```


HEADER FILE: LLIST.H

```
private:  
    void copy(const LList &source);  
    void dealloc();  
    ListNode* _find(size_t position);  
    ItemType _delete(size_t position);  
    ListNode *head_;  
    int size_;  
};
```

_FIND METHOD

```
ListNode* LList::_find(size_t position)
{
    ListNode *node = head_;
    size_t i;
    for (i=0; i<position; i++) {
        node = node->link_;
    }
    return node;
}
```

_DELETE METHOD

```
ItemType LList::_delete(size_t position)
{
    ListNode *node, *dnode;
    ItemType item;
    if (position == 0) {
        dnode = head_;          head_ = head_->link_;
        item = dnode->item_;    delete dnode;    }
    else {
        node = _find(position - 1);
        if (node != NULL) {
            dnode = node->link_;  node->link_ = dnode->link_;
            item = dnode->item_;  delete dnode;    }
        }
    size_ -= 1;
    return item;    }
```

INSERT METHOD

```
void LList::insert(size_t i, ItemType x)
{
    ListNode *node;
    if (i == 0) {
        head_ = new ListNode(x, head_);
    }
    else {
        node = _find(i - 1);
        node->link_ = new ListNode(x, node->link_);
    }
    size_ += 1;
}
```

DESTRUCTOR

```
LList::~LList() {
    dealloc();
}
void LList::dealloc()
{
    ListNode *node, *dnode;
    node = head_;
    while (node) {
        dnode = node;
        node = node->link_;
        delete dnode;
    }
}
```

LINKED DYNAMIC MEMORY ERRORS - BREAKING LINKS

The integrity of a linked structure depends on the correct maintenance of all the links, since these are required to access the information in the structure. In our linked list class, if the `ListNode`'s `link_` attribute is set incorrectly, the resulting list will not be valid:

- If the link is incorrectly set to `NULL`, the list will be shortened, losing all data after that node. A memory leak will also occur, since there is no way to access the nodes to deallocate them.
- If the link is set to a node further along on the list, all nodes in between will be stranded: their data will be lost and their memory will not be deallocated.
- If the link is incorrectly set to a node earlier in the list, then a circular structure results (traversing the list becomes an infinite loop).

LINKED DYNAMIC MEMORY ERRORS - BREAKING LINKS

```
// this code is incorrect
void LList::insert(size_t i, ItemType x)
{
    ListNode *node;
    if (i == 0) {
        head_ = new ListNode(x, head_);
    }
    else {
        node = _find(i - 1);
        node->link_ = new ListNode(x); // incorrect
    }
    size_ += 1;
}
```

LINKED DYNAMIC MEMORY ERRORS - MORE ERRORS

Python doesn't allow to use a name that has *not been defined* or is value *None*.

Example: `node = None`, and we attempt to execute `node.link` or `node.item`, the Python interpreter will catch this problem and generate an exception and traceback (if you don't catch it).

In C++ if you try to dereference an *uninitialized pointer* or a pointer that *refers to a deallocated object*, the run-time environment will attempt to access the memory location, resulting in garbage data or a memory fault that crashes your program.

IN-CLASS WORK

As of now, we have two list classes defined in C++: **List** and **LList**.

- 1 Finish up the in-class **List** work from the previous lecture.
- 2 Define **cout** operation for the objects of type **LList**.

IN-CLASS WORK

3. Using **testLList.cpp** write the program that performs the same operations using **LList** class:

(I)

- 1) Creates an array of 20 elements,
- 2) fills it with $1^2, 2^2, \dots, (20)^2$,
- 3) displays it,
- 4) then adds all of them and displays the sum, then

(II) then

- 5) Define a friend function **cin** for **List** class,
- 6) Ask the user for a size of an array (now many values the user plans to enter),
- 7) Create an array of capacity = $10 * \text{size}$,
- 8) Get the numbers from the user to store in the array,
- 9) add 10 to each member of the array and display it.