CSI33 Data Structures

Sharon Persinger

Fall 2019

Day 23 November 25

Topics

Errors in Dynamic Memory Allocation and Deallocation

C++ Linked Lists

Errors in Dynamic Memory Allocation with Linked Lists

Dynamic Memory Errors

Memory leak:

- A memory leak occurs when you allocate memory with the new operator but never deallocate it. If your program does this reputedly, the program will occupy more memory than it actually needs, maybe even more memory than is available. The disk will be used for swap space, causing the computer to slow down.

- Sometimes this is hard to detect since the new and delete methods aren't in the same function.
- Review your code to be sure you have deallocated all memory allocated with new. Be sure to write a destructor method for every class with dynamic memory allocation.

Example with memory leak: Where is it?

```
void f() {
  int * x;
  x= new int;
  *x = 3;
  x= new int;
  *x = 4;
  delete x;
  }
```

Dynamic Memory Errors

Accessing invalid memory:

Computer hardware splits memory into sections called pages, and assigns different pages to different programs running at the same time.

Checks are done at the hardware level to prevent one program from accessing the memory being used by another program. Attempting this access will crash the program.

A program may access memory in own of its own pages that was not allocated for that purpose and cause the program to behave erratically or crash.

Dynamic Memory Errors

Accessing invalid memory:

A program may access memory in own of its own pages that was not allocated for that purpose and cause the program to behave erratically or crash.

What happens when you access the memory positions that follow a C++ array using indexing?

What happens when you directly assign an address value to a pointer?

What happens when you assign a value using a pointer that has been declared but not initialized?

What happens when you assign a value using a pointer that was initialized but has been deleted?

C++ Linked Lists

First implement a class ListNode

Two private member variables:

Item variable must have a type specified

Link variable is a pointer to a ListNode object

Constructor is simple enough to be done inline.

Constructor can be called with 0, 1, or 2 parameters. What happens in each case?

```
class ListNode {
public:
 ListNode(int item = 0, ListNode* link=NULL);
 private:
int item_;
 ListNode *link_;};

inline ListNode::ListNode(int item, ListNode *link){
 item_ = item;
link_ = link;}
```

C++ Linked Lists

Before we implement the LinkedList class, look at a couple of details in ListNode.h

```
typedef int ItemType;
```

```
friend class LList;
```

C++ Linked List class

Continue with the translation of Llist into C++.

Look at Llist.h and compare it to the Llist implementation in Python

Data is private: size for number of elements, head a pointer to the first ListNode

Constructors, destructor, assignment operator

Size method, append method, insert method, pop method, indexing operator

Private methods copy, dealloc, _find, _delete to collect some common functionality for reuse

C++ Linked List class

Look at the implementation file.

These method are similar to the versions in Python:

_find, _delete, insert, and pop

Compare them. Notice we have to deallocate nodes when they are being removed.

Notice the use of the new operator in append. It returns a pointer to the new ListNode object.

Notice the use of -1 as index for the final item in the pop method.

Default constructor is easy.

C++ Linked List class

Look at more methods in the implementation file

Overload [] returns a reference to allow for assignment using []. The item on the left of an assignment operator must be an l-value or something that can be assigned to, i.e., a memory location.

C++ Linked List class

Methods necessary for class with dynamic memory:

Copy constructor for deep copy:

Work is done by the copy method. must iterate through the ListNode objects with the variable snode. Crete a new version of each in the variable node and attach it to the new Llist object.. Copy copies the source into the calling object

Assignment operator deallocates existing calling object first, then uses the copy method to make a copy to assign

Destructor calls dealloc. dealloc iterates through the list, deleting nodes as it goes.

Linked List errors

Be careful not to lose nodes. What happens here”

```
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); //error here
  }
  size_ +=1}
```