Doubly Linked List Programming, needed for Proj 2.

Singly Linked List Programming: DS0 Ch. 5

OOP style compared to classical C style...C style OK for this course.

Enrichment: Java/ C++ compared.

Project 2 Planning

CSI 310: Lecture 11-12
passing to the editorcore.

5. For the `input` command, also input the DATA LINE and prepare it for

4. Call one or more editorcore functions to perform editorcore operations.

to editorcore functions.

3. Gather a command argument (for later commands) and prepare to pass it

2. Determine legality and Kind of command.

1. Read command Lines from `cin`.

Purposes of command driver/user interface module:

2. Two kinds of input lines: (a) Commands (p) Data

1. Input is (old fashioned, not GUI) line-by-line.

Important application concepts

1. one-by-one.

2. Have an environment for adding and testing editor functionalities

I. Can compile and test right away

Start HERE, so you

Command driver or user interface module.

University at Albany Computer Science Dept.
in the main() function. Use helper functions if you wish.

OK and encountered: Implement the user interface or command driver module

```plaintext
{
  if print error message;
  
  exit if it is the quit command;
}
(except for about and quit):
Call the right editorcore function(s)
read and prepare data input if any;
gather any argument;
}

if (it is legal)

Determine what kind of command it is:
read a command;
}

(2) Repeat forever:

(1) Construct one editorcore object.
Command driver or user interface pseudo-code:

University at Albany Computer Science Dept.
```
Express the module separation between the user

"cursor", i.e., current line (an application concept.

Other variables: a pointer (and maybe an integer too) to implement the

cutHEAD and cutTAIL pointers to access the cut buffer.

HEAD and TAIL pointers to access the main buffer.

MYCore is the structure that holds these 4 pointer variables:

instance = object = variable

{
  ...

  editcore MYCore;
}

main()

(2) The program will instantiate ("have") ONE instance of editcore

defines a (data) type plus functions

Class definition: Data members, Function members (methods)

EDITCORCE CLASS

University at Albany Computer Science Dept.
command study. Cxx illustrates how to do this.

2. Use operator>> functions on cin to command the C++ input library

...also libraries for user editing, command histories, lexical scanning

This approach scales better for more complicated command

Scan and detect tokens (character sequences separated by whitespace or other

1. Read the whole command line into a character array buffer

2. Solutions

    a) User interface: Read, analyze, and act on command lines one-by-one.

    1) Insertion: Read line into the editor’s main buffer,

    2) Reading one new data line

    3) Inserting a new line into the editor

Programming problems:

Editor core by a C++ class

Implementing the interface and the editor core by

University at Albany Computer Science Dept.
Java Enrichment—Heavy cream and Sugar
What is a pointer value? (synonyms: address, locator, "reference" in Java)

What is a variable? (synonyms: object, memory location, "cell", "box for data")

Summary...
equivalent, \( T \) \text{ as a C++ pointer}

\[ \text{Tip for the Trendy: The following in Java:} \]

```
class Thing { public void void memberFun() { ... } }
```

\[ \text{Tip for the Trendy: The following in Java:} \]

```
class Thing { public void void memberFun() { ... } }
```

\[ \text{Tip for the Trendy: The following in Java:} \]

```
class Thing { public void void memberFun() { ... } }
```

\[ \text{Tip for the Trendy: The following in Java:} \]

```
class Thing { public void void memberFun() { ... } }
```

\[ \text{Tip for the Trendy: The following in Java:} \]

```
class Thing { public void void memberFun() { ... } }
```

\[ \text{Tip for the Trendy: The following in Java:} \]

```
class Thing { public void void memberFun() { ... } }
```

\[ \text{Tip for the Trendy: The following in Java:} \]

```
class Thing { public void void memberFun() { ... } }
```

\[ \text{Tip for the Trendy: The following in Java:} \]

```
class Thing { public void void memberFun() { ... } }
```

\[ \text{Tip for the Trendy: The following in Java:} \]

```
class Thing { public void void memberFun() { ... } }
```

\[ \text{Tip for the Trendy: The following in Java:} \]

```
class Thing { public void void memberFun() { ... } }
```

\[ \text{Tip for the Trendy: The following in Java:} \]

```
class Thing { public void void memberFun() { ... } }
```

\[ \text{Tip for the Trendy: The following in Java:} \]

```
class Thing { public void void memberFun() { ... } }
```

\[ \text{Tip for the Trendy: The following in Java:} \]

```
class Thing { public void void memberFun() { ... } }
```

\[ \text{Tip for the Trendy: The following in Java:} \]

```
class Thing { public void void memberFun() { ... } }
```

\[ \text{Tip for the Trendy: The following in Java:} \]

```
```
Java has only 2:

1. **Primitive types (int, char, float, etc.)**
   - C/C++ has 3 kinds of named variable and array types:

2. **Structure/Class types (immutable, implementable, etc.)**
   - C/C++ objects or pointers, which are like C/C++ pointers to class objects.

3. **Pointer type (int* point, etc.)**
   - C/C++ has 3 kinds of named variable and array types.

Java hides numerical pointer values/addresses and provides variables/values/addresses.

**EXPOSE** numerical pointer

---

(2) Java, "Reference types" which are like C/C++ pointers to class objects (independent)

(1) Primitive type (roughly like C/C++'s but implementation)

(2) Pointer type (int* point, etc)

(3) Pointer type (C/C++ has only 2):
Back to Proj 2 data structures...
2. Structure node for the linked list of char * data.

1. C-strings for text lines.

Pro! 2 Dynamic objects:
The dynamic memory used shrinks as well as grows as needed, dynamically.

Partially filled array.

dynamically allocated, DSO Ch. 4 covers the dynamically allocated
such data structures can be virtually unlimited in size if the objects are

objects.

that contain some pointer type fields that hold addresses of structure type
A linked data structure consists of some structure type objects (variables)
multitudes of Java Library and Language Classes are 
dynamic.

FYI: All of the objects (variables, instances) of the Intereters and full 
ụll

(matrice and free in C).

coded operation (new and delete) executed when the program runs.

But, a dynamic variable is created or destroyed only by an explicitly 
function is called and are destroyed (storage recycled) when it returns.

3. CS201 Local extern (C/C++ automatic) variables are created when a 
import data can determine if and how many dynamic variables are created.

2. Dynamic variables are created during process execution. Unpredictable

1. Dynamic variables are not declared. They are not named by identities.

You need pointers to access them! In Java you need “references”.

Dynamically Allocated Variables

University at Albany Computer Science Dept.
\[ p_i = \text{new int}[4]; \]

new

to allocate an array of say 4 ints, do the same except use the array form for

\[ p_i = \text{new int}; \]

and then make program execute:

\textit{e.g.:} \texttt{int *p}\_

\texttt{p} to hold its address.

To allocate ONE variable of any type, say \texttt{int}, make sure you have a pointer

Doing dynamic allocation: \texttt{use new}.
It consists of nodes, a Head pointer, and (usually) a Tail pointer. It has known locations for insertion and deletion near entries at one-dimensional sequences, so entity insertion and deletion near entries at

Linked List: Concrete linked data structure good for implementing

```
node * tail ptr;
node * head ptr;
...

{ }
node * link ptr;
value-typ data-field;

private:
and access methods
... constructor, manipulator
typecast double value-type;

public:
```

Classical C Style

```cpp
class node { public:
void SetLink(node *link) { this->link = link; }

Object/Modern C++ Style
```
Building a 1-item Linked List

```c
typedef struct node {
    double data;
} node;

class node {
    public:
    node * HEAD;
    node * TAIL;

    node * HEAD, TAIL;
    node * TAIL = NULL;
    HEAD->data = 23.6;
    node * TAIL = new node;
    node * HEAD;
    node * TAIL;

    ...;

    };

    node * TAIL,
    val-tp data;

    private:
    ...;

    {.
        data = L;
        TAIL = new node
        p = NULL;
        node * p = NULL;

        node(const val-tp A = val-tp(),

        typedef double val-tp;

        public:

    };

    node * HEAD;

    };

    node * HEAD;

    // How is your wheel if you forget
    HEAD->LINK = NULL;
    HEAD->data = 23.6;
    HEAD=TAIL=new node;

    node * TAIL;
    node * HEAD;

    ...;

    {.
        node * TAIL,
        val-tp data;

        private:
        ...;

        {.
            data = L;
            TAIL = new node
            p = NULL;
            node * p = NULL;

            node(const val-tp A = val-tp(),

            typedef double val-tp;

            public:

        };

        node * HEAD;

        };

        node * HEAD;
```
With our classical style code, and (2) are executed before and (3) (4) (5)

1. Copy "ditto" into automatic variable HEAD.
2. Copy the address (pointer to the new node) into automatic variable TAIL.
3. Copy 0 (NULL) into the TINK field of TAIL.
4. Copy (3) 23.6 into the data field of the new node variable.
5. Copy (1)ächst of the new node variable.
6. Run the dynamic memory allocator "new" of the C++ support library.

For both styles, the computer does the same things with the same results.
if (TAIL == NULL) TAIL = HEAD;

// HEAD before copying it
if (size == 0) if you lose your
  HEAD = tp;
  tp->next = HEAD;
  tp->data = 14.7;
  tp = new node;
  node * tp;

struct node

Classic C Style

class node

OOP/Modern C++ Style

Adding an item to the front:
23.6; it was NOT 23.6.

The old value of HEAD was the ADDRESS of the node "housing"

(1) RUN the dynamic memory allocator "new" of the C++ support library (I)

(2) Copy (short) 14.7 into the data held of the new node variable (2)

(3) Copy pointer value from HEAD into the link held of the new node (3)

(4) Copy the address of the new node (the one containing 14.7) into automatic variable HEAD. (This operation OVERWRITES the previous value in HEAD, so it must be done AFTER (3)).

(5) If TAIL==NULL, the original list was empty, so TAIL=HEAD gives TAIL its correct value!

(old value before step 4)
meantime, plus practice solving problems!
understanding of variables, data and (C/C++) language
Have patience; Speed and elegance come from precise
these steps.
If it fails for boundary cases, try to fix it and repeat

(If and 1-item list)
Then, check it for any boundary case(s). (E.g. Empty
and draft code if not.
Check that it works in the general case. Re-do design
idea in the general case.
Advice: Try to design and draft code for an algorithm
Otherwise, it's good.

This last C++ statement works when the list was empty, and does nothing.
public class Edtorcorer {
    private node *TAIL;
    private node *HEAD;

define edtorcorer entity //
    {
        char *data;
        node *prev;
        node *next;
    }

struct node {
    we USE completed public class\class/struct node:

#include <assert.h> for NULL
#define edtorcorer-struct
#define edtorcorer-struct
#endif edtorcorer.h

simplicity:

Alternative: Do this if you are having trouble, haven't started (really?) or love
Main/Seave: "Linked List Toolkit" is optional.

editorCore(); { HEAD=TAIL=NULL; /* etc */ }
private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:

private:


```c
void edittorcore::insertFront(char * pch) {

    Node * tp = new Node;
    if (TAIL == NULL) {
        TAIL = tp;
        HEAD = tp;
    } else {
        tp->next = TAIL; // HEAD is a class edittorcore DATA MEMBER!
        tp->prev = NULL; // NULL is a standard macro.
        TAIL = tp;
    }
    tp->data = pch; // LOCAL VARIABLE!
    node * tp = node * temp--node-pointer;
}

#include <stdlib.h> // for NULL
#include "edittorcore.h"
#include "edittorcore.hxx"
(i.e., executed return; or falls out.)

Automatically deallocated WHEN

tp is a local storage extent variable—NOT the variable named tp

POINTS TO.

delete tp; deletes the OBJECT/STORAGE/VARIABLE the value of tp

Main and Savage warn "Don't code delete tp;"
if (HEAD == TAIL)
    TAIL = t;

if (TAIL) TAIL->next = t;

if (TAIL) TAIL = NULL;
t->next = TAIL;
t->data = 33.3;
node * t = new node;

void set_TK(node * p)
    t->next = NUL;

classical C Style

private:

typed double val->tp;

public:

class node { class node :}

//OO/OOP/Modern C++ Style

Adding an item to the end: Requires modifying last node
TAIL before dereferencing it, to set the last link!

// Moe is you if you lose your
2. Rule of C++: The NULL pointer value is converted to boolean false.

When isn’t there a last node?

MUST be conditional on whether there is a last node.

Since the last node of the original list must be modified (to replace the

Remarks:
if (HEAD != NULL) {
    TAIL->set lk(t);
    TAIL = t;
}

if (HEAD == NULL) {
    if (TAIL == NULL) {
        node *t = new node(33.3);
        TAIL = t;
        TAIL->set lk(t);
        TAIL = t;
    } else {
        node *t = new node(33.3);
        TAIL = t;
        TAIL->set lk(t);
    }
}

node *t = new node(33.3);
TAIL = t;
{
  HEAD = t;
  if (list_is_empty(HHEAD));
}
esthe
{
  TAIL->LINK = t;
  if (TAIL); /* list is not empty */ assert(HHEAD);
}

Alternative (nicer) code:
between `Two` and `The`?

How can you insert the C-string containing `Dogs`?

Given this doubly linked list of 3 C-string addresses,
Given the original data structure, how can you change it to

Dogs

THREE
NULL

TWO
NULL

ONE
NULL

HEAD

TAIL
1. Determine char array length needed is 5, for "dogs"

2. Allocate char array (for C-string) saving its address.

3. Copy C-string into it.

4. Allocate a dnode, saving its address.

5. Copy char array's address into it.

6. Which means (tpn*) .data = tpc!

7. Copy C-string into it.
Given the original data structure, how can you change it to this one???

Dogs

TH3

6

One

TWO

NULL

NULL

TAIL

HEAD

University at Albany Computer Science Dept.
Remember, we want to insert dogs between the 2 circled nodes. We need the addresses of two and three. So we can modify their values.

One
NULL

Two

TIP

NULL

TAIL

HEAD

Two

TIP

NULL

TAIL

HEAD

NULL
The field (data member) names will be omitted in our data structure diagrams.

```
struct dnode {
    dnode* pfore; // pfore==addr of the next dnode if any.
    dnode* pback; // pback==addr of the previous dnode if any.
    char *data; // data==addr of a C-string.

    // data!=NULL.
};
```
Alternatively, if we know the address of the circled node, we can program:

\[
\text{tpLeft} = \text{tpRight} \rightarrow \text{pBack};
\]

If \( \text{tpLeft} \) contains the address of the circled node, the following code calculates the address of the right node:

\[
\text{tpRight} \rightarrow \text{pBack} = \text{tpRight} \rightarrow \text{pFore}.
\]
Two

One

NULL

NULL

(1) Starting from value of Head or Tail Traverse.

(2) Use the value in a "cursor" variable, or search the linked list forward or backward.

But, how can we obtain ANY one of these addresses?

(C++ declarations for these variables:

```cpp
struct node {
    int value;
    node *next;
};
```
Now a new dnode containing the addr. of the Dogs string is accessible via $tpn$ and 2 variables hold the addresses of dnodes we will put it between. How can we connect it?

Relevant data str. is circled:
**Step 1:** Set the link fields in each of the existing nodes.

```
*tpn = *tpn->fore = *tpn->pback = *tpn;
```

**Step 2:** Set the link fields for the new node to connect into the linked list.

```
*tpn = *tpn->fore = *tpn->pback = *tpn;
```

Only 2 more steps remain.
The result of step 1:
The result of step 2.

Voila!
The temporary pointer variables are not needed any more. (out of the picture)

The big picture after the changes:

The temporary

Dogs

HEAD

TAIL

NULL

NULL

NULL

three

one
two
Another cut

This line's been cut

Dogs

A bigger picture after the changes:

Possible cursor feature

Possible cursor feature

Solution
array alloc, and C-stmngs, man struct and previous study.

explanation, analyses, etc. Given here, in chapter 5, chapter 4 (on dynamic

Your job is to solve the programming problem, using the directory:

2. Removing a node; and moving it into the call-list?

OTHER pointers must also be set.

Hint: Whenever a node is inserted, its own 2 pointers must be set, AND 2

6. Inserting a node into the empty list?

2. Inserting a node at the beginning?

3. Inserting a node after the last?

implemented.

3. Moving the cursor up or down. Easy, depends on how the cursor is

puzzle for you.

2. Making the cursor work to indicate the end of list? Hmm... that's a

1. Implementing a cursor in front of a line? Easy! use a node pointer.

What about

University at Albany Computer Science Dept.