If there's time, what is a function member called, really? What is "the object for which it is called?"

Specific Project 2 questions.

More tips for Project 2.

Steps for inserting into the middle of a doubly-linked-list for Project 2.

CSI 310: Lecture 11
pointer to (addr of) a char
pointer to (addr of) a node

Another cut
This line's been cut
First main list/Text line
2nd line
The third

pointer to (addr of) a char
pointer to (addr of) a node

NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL

This is my editor core
You design any others
cursor and
my editor core
pointer to (addr of) a char

pointer to (addr of) a node

Another cut

This line's been cut

First main list text line

2nd line

The third

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

Another cut

First main list text line

2nd line

The third
how can you insert the C-string containing "Dogs" between "Two" and "Th3"?

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

Dogs

TH3

NULL

TWO

One

NULL

TAIL

HEAD
1. Determine char array length needed is 5, for "dogs".
2. Allocate char array (for C-string) saving its address.
   `tpc=new char[5];`
3. Copy C-string into it.
   `strcpy(tpc, "dogs");`
4. Allocate a node, saving its address.
   `tpn=new dnode;`
5. Copy char array's address into it. (which means)
   `tpn->data=tpc;`
   `(tpn->).data=tpc;`
Given the original data structure, how can you change it to this one??

Dogs

Given the original data structure, how can you change it to this one??

Dogs
so we can modify their values.

the 2 circled nodes
We NEED the addresses of
Two and Th3
insert dogs between
Remember, we want to

Remember, we want to

insert          between

We NEED the addresses of
Two and Th3
insert dogs between
Remember, we want to

Remember, we want to
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.
};

// the first dnode.
// pback==NULL if this is the first dnode.
// pfore==NULL if this is the last dnode.
// pfore==addr of the next dnode if any.
// pback==addr of the previous dnode if any.
Alternatively, if we know the address of the left node, we can program:

\[
\text{tpleft} = \text{tpright} \rightarrow \text{pback};
\]

The following code calculates the address of the right node:

\[
\text{tpright} = \text{tpright} \rightarrow \text{pfore};
\]

If tpleft contains the address of the left circled node, the address of the right circled node is calculated.

HEAD

TAIL

NULL
(1) Starting from value of HEAD or TAIL traverse or search the linked list forward or backward.

(2) Use the value in a "cursor" variable.

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

The variables: node * tpleft; node * tpright; (C++ declarations for these variables)
Now a new dnode containing the addr. of the dog's C-string is accessible via \texttt{tpn}.

AND 2 variables hold the addresses of dnodes we will put it between. Relevant data str. is circled:

\[ \]
to connect (which is the new dnode)

in:

\[\text{tpn} \rightarrow \text{fore} = \text{tpn}!\]

\[\text{tpn} \rightarrow \text{pback} = \text{tpn}!\]

\[\text{tpn} \rightarrow \text{fore} = \text{tpn}!\]

\[\text{tpn} \rightarrow \text{pback} = \text{tpn}!\]

(2) Set 1 link field in each of \(\text{tpn}\):

(1) Set 2 link fields into the linked list:

(Which is the new dnode)

Only 2 more steps remain

\[\text{tpn}^!\]
The result of step 1.

1.

\texttt{tpn->fore=tpleft!}

\texttt{tpn->back=tpleft!}

University at Albany Computer Science Dept.
The result of step 2:

\[
\text{tpnt} = \text{nul}
\]

\[
\text{tpleft} -> \text{pfront} = \text{tpn};
\]

\[
\text{tpright} -> \text{pback} = \text{tpn};
\]

\text{Volta!}
The temporary pointer variables are not needed any more. (out of the picture)

The temporary pointer variables

One

Two

THREE

NULL

NULL

Dogs

HEAD

TAIL

The big picture after the changes:
allocate and C-string(s), man start and previous study.

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

Your job is to solve these programming problems using this datastructure.

7. Removing a node; and moving it into the cell-list.

OTHER pointers must also be set.

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

6. Inserting a node into the EMPTY list?

5. Inserting a node AT THE BEGINNING?

4. Inserting a node AFTER THE LAST?

implemented.

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

for you.

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

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

What about
What are member functions, really?

Pointers to structs and accessing data members AKA fields.

CSI 310: Lecture 10.5
Selection using field name:

\[
\text{STL}_* \text{ps} = \text{new STL';}
\]

Dynamic allocation by:

\[
\text{STL}_* \text{p} = \text{ps} \
\]

Automatic storage allocation by:

\[
\begin{array}{l}
\text{struct STL} \\
\text{ \\ } \text{int addr; STL* link; }
\end{array}
\]

Selection using subscript notation:

\[
\text{STL}[\text{p}];
\]

Dynamic allocation by:

\[
\begin{array}{l}
\text{STL}_* \text{p} = \text{ps} \
\text{STL}_* \text{p}[\text{ps}];
\end{array}
\]

Automatic storage allocation by:

\[
\begin{array}{l}
\text{struct STL} \\
\text{ \\ } \text{int addr; STL* link; }
\end{array}
\]

selected by subscript value;

allocated contiguously;

uniform type;

Collection of variables, of

What is an array of

What is a structure (class type object) of

nonuniform types defined by you,

allocated near each other,

selected by field (member) name

dff

C. dff

Selection using field name:

\[
\text{STL}_* \text{ps} = \text{new STL';}
\]

Dynamic allocation by:

\[
\text{STL}_* \text{p} = \text{ps} \
\]

Automatic storage allocation by:

\[
\begin{array}{l}
\text{struct STL} \\
\text{ \\ } \text{int addr; STL* link; }
\end{array}
\]

Selection using subscript notation:

\[
\text{STL}[\text{p}];
\]

Dynamic allocation by:

\[
\begin{array}{l}
\text{STL}_* \text{p} = \text{ps} \
\text{STL}_* \text{p}[\text{ps}];
\end{array}
\]

Automatic storage allocation by:

\[
\begin{array}{l}
\text{struct STL} \\
\text{ \\ } \text{int addr; STL* link; }
\end{array}
\]

selected by subscript value;

allocated contiguously;

uniform type;

Collection of variables, of

What is an array of

What is a structure (class type object) of

nonuniform types defined by you,

allocated near each other,

selected by field (member) name

dff

C. dff

Selection using field name:

\[
\text{STL}_* \text{ps} = \text{new STL';}
\]

Dynamic allocation by:

\[
\text{STL}_* \text{p} = \text{ps} \
\]

Automatic storage allocation by:

\[
\begin{array}{l}
\text{struct STL} \\
\text{ \\ } \text{int addr; STL* link; }
\end{array}
\]

Selection using subscript notation:

\[
\text{STL}[\text{p}];
\]

Dynamic allocation by:

\[
\begin{array}{l}
\text{STL}_* \text{p} = \text{ps} \
\text{STL}_* \text{p}[\text{ps}];
\end{array}
\]

Automatic storage allocation by:

\[
\begin{array}{l}
\text{struct STL} \\
\text{ \\ } \text{int addr; STL* link; }
\end{array}
\]

selected by subscript value;

allocated contiguously;

uniform type;

Collection of variables, of

What is an array of

What is a structure (class type object) of

nonuniform types defined by you,

allocated near each other,
pA = new eleType[3];

struct STT{
    int ddF;
    STT* link;
};

STT G;
STT* ps;
ps = new STT;
G.link = ps;

struct STT t[5];
\begin{verbatim}
    void fake(SSt *);
    void fake(GS);
    
    Calling the "Fake function member":
    
    { 
        cout >> "ERROR: dtt not in scope"
    
        // Prints the same thing!
        print >> "PRESSIONS"
        cout >> p_dtt;
        cout >> p_dtt * p_dtt;
        cout >> p_dtt - p_dtt;
        cout >> p_dtt (p_dtt * p_dtt); 
    }

    void fake(SSt *)( ) ()
    
    // Called. Also it is in the class's scope and can access private members.
    
    called the address of the object for which it was called holding the address of the object for which it was called (value parameter) (which you can access via the keyword this) within the value parameter (which you can access via the keyword this) within the value parameter.
    
    A fun. member is like an ordinary fun. except it has an extra (implicit) what does calling a function member like fun really mean?
    
    ... SSt & c; SSt * pS; pS = new SSt;
    }

Suppose struct SSt { void fun ( ) { int dtt; 
\end{verbatim}
by setting a breakpoint on it (you must "break main" or equvil. and run first).

(2) You can also debug a function

debugger out of it with thefinish command. You can also debug a function

Tips: (1) If you step into a MESS (library function, etc.) you can pull the

**FUNCTION WAS CALLED.**

**FUNCTION WILL PRINT THE OBJECT FOR WHICH THE**

**PRINT */ THIS**

**ACTIVATION:**

**WHEN DEBUGGING INSIDE A C++ FUNCTION**

**STEP STEPS INTO FUNCTION WHEN THEY ARE CALLED, TOO.**

**FUNCTION:** Function calls are performed and return WITHOUT STOPPING.

**ADD COMMAND next:** Performed line-by-line controlled execution of the current

**"this" is handy for debugging.**
Computer Programming, unlike sketching, etc., is unlikely to break legs, etc., on too difficult slopes, so try anything you're curious about.

4. Overloading prefix ++ operator with a member function (X returns X).

4. Application programs.

4. Overloading prefix ++ operator with a member function.

3. Defining of ostream::operator<< (i.e., this course).

RECOMMENDED FOR BEGINNERS. *I.e., this course.

1. Detection of self-assignment for proper implementation of overloaded assignment operators. See page 174 of DSO for details.

2. Some member functions will store the address of the current object into the current object.

Some uses of C++ "this" pointer value in programming:

University at Albany Computer Science Dept.