Project 2 Planning and Problems.

Project 2 Data Structures.

CSI 310: Lecture 10
Another cut

This line's been cut

First main list text line

2nd line

The third

The pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL

pointer to (addr of) a node

pointer to (addr of) a char

NULL
p = \&[0](p[0])

*p denotes the array element with index 1.

++ peculiarities: p[0] is a SYNONYM for *

*p changes the first character to A.

\texttt{count} \rightarrow \ast p \texttt{prints the first character}.

\texttt{p denotes THE ACTUAL array element}.

\texttt{value of p equals THE ADDRESS OF the array element holding the first character}.

Accessed with \texttt{char \* p which "points to" the first char in the array}.

marked by the NULL char.

1. Dynamic C-strings: Dynamically allocated char arrays where the end is
class Node
{
  char *data;
  Node *prev;
  Node *next;
}

class Node
{
  Node *cutTAIL;
  Node *cutHEAD;

  if (cursor == NULL) cursor = last; // insert
  if (cursor == NULL) cursor = first; // representational invariant

  Node *cursor;

  Node *TAIL; // addr of Node for last line as in DS0
  Node *HEAD; // addr of Node for first line as in DS0
}

class editorcore

members/fields:

1. class strict variable/observer instance that CONTAINS pointer data
2. class strict variable/observer instance that CONTAINS pointer data

©S. Chakravarthy 2006 University at Albany Computer Science Dept.
is its first element. (Address of the first element.)

\texttt{C/C++ peculiarity: An array name like \texttt{inp} \texttt{buf} \texttt{DENOTES} the pointer to \texttt{arr} of chars.

"\texttt{strcpy} library fun. \texttt{strcpy} requires TWO pointers, both pointing to the \texttt{new} char array."

\texttt{\texttt{strcpy} (\texttt{inp}, \texttt{tempbuf}) ;}

\texttt{\texttt{strlen} in \texttt{input} has been copied into \texttt{one} new char array has been dynamically allocated.}

\texttt{\texttt{new char} \texttt{[length + 1] ;} // MUST ADD 1 char \texttt{* tempbuf ;}

\texttt{\texttt{new char} \texttt{[length] ;}}

\texttt{...}

\texttt{Read data line into \texttt{inp} ;}

\texttt{...}

\texttt{char \texttt{\texttt{inpbuf} [INBUF\texttt{SIZE}]} ;}

\texttt{}}
whose ADDRESS is in p.
The TYPE OF *p is int, so this prints the INTEGER value in the variable.

\[ \text{cout } \gg \ *p; \]

The library function for PRINTING addresses is called.

\[ \text{cout } \gg p; \]
\[ \text{print the ADDRESS OF an integer variable.} \]

\[ \text{BUT } \text{int } *p; \]
\[ \text{print the ADDRESS of a string is called.} \]

\[ \text{p.} \]
\[ \text{cout } \gg p; \]
\[ \text{print the C-STRING in the array pointed to by the value of} \]
\[ \text{char } *p; \]
\[ \text{contenets.} \]

Another peculiarity, obvious and convenient but leads to confusion in other

\[ \text{CS. Chapin 2006} \]

University at Albany Computer Science Dept.
Here is the dynamic allocation of a new Node:

```c
void edittorcore::insert(char* char_array)
{
    ...

    MyCore::insert(p);
    ... the new C-string.
    ... and the line data is copied into
    ... its address is stored in p,
    ... a new C-string is allocated,
    ... char *p;
    ...

    edittorcore MyCore;
}  

main()

Here is the CONTEXT:

4. Dynamic allocation of a Node:
tpNode->pdata = argep; // equitv. (tpNode).pdata=argep;

member is
for "Dereference a pointer-to-class/structure then select a data or function

C++ pecularity: The more popular ALTERNATIVE SYNTAX

/

(ftpNode).pdata = argep;

// a new node was dynamically allocated.

// A new node was dynamically allocated.
The value of x is 98.6
The address of x is 0x0001999A

\$ a.out

{ }

cout << "The value of x is " << x << endl;
cout << "The address of x is " << &x << endl;
float x = 98.6;
}

main()

using namespace std;

#include <iostream>

which must be a variable.

The C/C++ & operator provides the address of its operand,
values, pointer variables, getting addresses, dereferencing, etc.

limitations and performance in Java, you have to know about pointer/address

To use dynamic data structures in C/C++, and understand their nature,

© S. Chakravarti 2006

University at Albany Computer Science Dept.
The type of variable can hold the address of (the type of data it can point to)

The declaration of a pointer variable specifies BOTH
dereferencing operator...hmm...

which, by coincidence, is also the C/C++
an asterisk

point to (the type of data it can hold the address of)
that the pointer variable
The name of the newly declared pointer variable
The name of the newly declared pointer variable!

float * MyFirstName;

Learn how C/C++ pointer variables are declared.

Important for USING pointers!

(2) the type of those variables it can point to,
(1) that it IS a pointer variable, and
MyP does NOT point to 2003!

Now we can say "MyP points to MyInt".

Why? 2003 can change while MyP still points to MyInt.

2003

MyInt

MyP

MyP = &MyInt;

MyInt = 2003;

int MyInt;

int *MyP;

computer's memory:

produces in the

The value of MyP is the address of the int variable MyInt

MEANS

pointer variable MyP points to int variable MyInt
_ALIASES.

General problem: names (different kinds).

One variable has two different names (different kinds).

Huh? One variable address value.

That means: Access the variable whose address is in the pointer (or is the pointer a pointer to the variable (or address value).

Important for USING POINTERS!
Before Multiplication:

\[
\begin{align*}
\text{MyP} &: 0xbe0fef04 \\
\text{MyInt} &: 2003
\end{align*}
\]

After Multiplication:

\[
\begin{align*}
\text{MyP} &: 0xbe0fef04 \\
\text{MyInt} &: 4006
\end{align*}
\]

\[
\text{MyInt} = \text{MyP} \times 2;
\]
This is a "CRASH": Computer tried to read memory at the illegal address 0x0,

Segmentation fault

cout << MPF << endl;
cout << ox04006;
cout << MPF << " " << &MPF << endl;
cout << MPF << endl;
cout << &MPF << endl;

\[ \text{After Assignment!} \]

Before Assignment:

MPF = NULL;

\[ \text{NULL} \]

MPF

\[ \text{0x0} \]

MPF

\[ \text{ox04006} \]

MPF
try: cx = non-void in unary

try: cx = In function, int main()

-----

{ }

cout >> (x+3)

int x = 3;

}

main()

using namespace std;

<iostream>

// Expressions, like (x+3), are not variables, so:
but x equals 98.6
myStropeur equals 0xbd1e9bc
$ a.out
--------------
{
  cout << " x equals "
x >> myStropeur;
  cout << " x equals "
xX >> myStropeur;
  cout << " x equals "
xX >> myStropeur;
  myStropeur = xx;
  float x = 98.6;
  float *myStropeur;
} main()
the 3 variables, before and after each step:

Please simulate the execution of this by writing the values currently in each of

```c
int TEMP;
int p2 = 62;
p1 = &p2;
TEMP = &p1;
```

How can you program swapping the values of `I` and `J` using pointers?

```c
I = 1928; J = 2003;
int I; int J;
```
assert (pi==XI) (p2==XI) ;

1928 2003

count >> J >> I >> J end;

2003 1928

count >> *p1 >> *p2 >> J end;

p2 = TEMPp;
p1 = p2;
TEMPp = p1;
int *TEMPp;

p1 = XI; p2 = XI;
int *p1; int *p2;
int J; int J; int J;

Now, how is swapping the values of the pointer variables different?
What is a pointer value (synonyms: address, locator, "Reference" in Java)?

What is a variable (synonyms: object, memory location, "cell", "box for data")?
equivlent (\*member\*): \*member\*;

\*member\*;

\*member\*;

\*member\*;

\*member\*;

\*member\*;

\*member\*;

\*member\*;

\*member\*;

\*member\*;

\*member\*;

\*member\*;

\*member\*;

\*member\*;

\*member\*;

\*member\*;

\*member\*;

\*member\*;

\*member\*;

\*member\*;

\*member\*;

\*member\*;

\*member\*;

\*member\*;

\*member\*;
Java has only:

- \( \text{primitive type (} \text{int, char, float, etc.)} \)
- \( \text{pointer type (} \text{int} * \text{pointer}, \text{int \* \text{pointer})} \)
- \( \text{struct/class type (} \text{int, int\*int, \\text{int\*int\*int, etc.})} \)

C/C++ has 3 kinds of named variable and array types:

- \( \text{C/C++ has 3 kinds of named variable and array types: } \)
- \( \text{C/C++ has 3 kinds of named variable and array types: } \)
- \( \text{C/C++ has 3 kinds of named variable and array types: } \)

- \( \text{Java hides numerical pointer values/address and provides variables } \)
- \( \text{Java hides numerical pointer values/address and provides variables } \)
- \( \text{Java hides numerical pointer values/address and provides variables } \)

- \( \text{C/C++ exposes assembly/machine language numerical pointer } \)
- \( \text{C/C++ exposes assembly/machine language numerical pointer } \)
- \( \text{C/C++ exposes assembly/machine language numerical pointer } \)

except you cannot convert them to/from integers.

(2) Java „reference type” which are like \( \text{C/C++ pointers (to class objects) independent} \)

(1) Primitive type (roughly like \( \text{C/C++'s but implementation} \)

\( \text{C/C++ has only:} \)

- \( \text{pointer type (} \text{int, int\*int, int\*int\*int, etc.)} \)
- \( \text{struct/class type (} \text{int, int\*int, int\*int\*int, etc.)} \)
- \( \text{primitive type (} \text{int, char, float, etc.)} \)
passing to the editor core.  
5. For the INPUT command, also input the DATA LINE and prepare it for  
4. Call one or more editor core functions to perform editor core operations. 

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. 

Purpose of command driver/user interface module:  
2. Two kinds of input lines: (a) Commands (p) Data  
1. Input is (old fashioned, not GLUT) line-by-line. 

Important application concepts:  
one-by-one.  
2. Have an environment for adding and testing editor functionalities  
1. Can compile and test right away;  

Start HERE, so you  

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

{ }
else { 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;
collect 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:

© S. Chaiken 2006
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

(1) Class definition: Data members, function members (methods)
(2) The program will instantiate ("have") ONE instance of editCore

Define a (data) type plus functions

© S. Chaiken 2006

University at Albany Computer Science Dept.
command study C++ illustrate how to do this.

scan and decode tokens

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

and parse are available.

Syntax also libraries for user editing command histories, lexical scanning

This approach SCALES BETTER for more complicated command

punctuation (using array operations)

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

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

2 solutions

1) user interface: Read, analyze, and act on command lines one-by-one.

2) Inerting a new line into the editor’s main buffer.

3) Inerting a new line into the editor’s main buffer.

(2) Reading one new data line

1) User interface

Programming problems:

editor core by a C++ class

interface and the editor core by IMPLEMENTING the