Specific Project 2 Questions.

More tips for Project 2.

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

Getting a Landmark.

CSI 310: Lecture II
constant_member functions for the point class: //

double get_x() const //

: postcondition: the value returned is the x coordinate of the point

void rotate90() //

: postcondition: the point has been rotated clockwise 90 degrees.

and by y-amount along the y axis. //

void shift(double x-amount, double y-amount) //

: postcondition: the point has been moved by x-amount along the x

and y-amount along the y axis.

point(double initial_x = 0.0, double initial_y = 0.0) //

: constructor for the point class: //

constant member functions for the point class: //

provided: point (an ADT) for a point on a two-dimensional plane

file: nespoint.h (replaces from point.h) in figure 2.9 on page 58

Main/savage C# nespoint.h

University at Albany Computer Science Dept.
Postcondition: The return value is true if p1 and p2 are not identical.

bool operator != (const Point p1, const Point p2) //

Postcondition: The return value is true if p1 and p2 are identical.

bool operator == (const Point p1, const Point p2) //

Postcondition: The sum of p1 and p2 is returned.

Point operator + (const Point p1, const Point p2) //

Postcondition: The point returned is halfway between p1 and p2.

Point midpoint(const Point p1, const Point p2) //

Postcondition: The value returned is the distance between p1 and p2.

double distance(const Point p1, const Point p2) //

Nonmember functions for the Point class:

Postcondition: The value returned is the Y coordinate of the Point.

double getY() const //

University at Albany Computer Science Dept.
class Point
{
    // namespace main-savitch-2b don't bother

    #include <iostream>
    #include main-savitch-newpoint
    #include main-savitch-newpoint

    // Assumptions and the copy constructor may be used with Point objects.
    // VALUE SEMANTICS for the Point class.
    //
    // read from ins. The return value is the stream ins.
    // postcondition: The x and y coordinates of target have been
    //                written to outs. The return value is the stream outs.
    //                (istream &ins, Point target)

    // outs to outs. The return value is the stream outs.
    // postcondition: The x and y coordinates of source have been
    //                written to outs. (ostream &outs, const Point source)

    University at Albany Computer Science Dept.
point middle(const point p1, const point p2);
double distance(const point p1, const point p2);

// NONMEMBER FUNCTIONS FOR THE POINT CLASS

{

double x, y; // x and y coordinates of this point
private:

friend std::istream &operator >> (std::istream &is, point T)

friend FUNCTION //
{
    double get-y() const { return y; }
    double get-x() const { return x; }

    friend MEMBER FUNCTIONS //
    void rotate90();

    void shift(double x=amount, double y=amount);

    MEMBERS FUNCTION //
    point (double initial-x = 0.0, double initital-y = 0.0);

    public:

    }
#end

// DON'T BOther wITH SAVITcH NAMEsPACES

#include <iostream>
#include <vector>

int main()
{
    std::cout << "Hello, World!\n";
    return 0;
}

};
void Point::shift(double xAmount, double yAmount) {
    y = y + yAmount;
    // Constructor sets point to a given position
    x = x + xAmount;
}

} // namespace main-salitch-2B

using namespace std;

#include "newpoint.h"
#include <math.h>
#include <iostream>

CLASS IMPLEMENTED: Point (see newpoint.h for documentation)
FILE: newpoint.cc

Main/SavAGE Ch2 newpoint.cc

University at Albany Computer Science Dept.


```cpp
return
}

bool operator ==(const point& pt1, const point pt2)
{
    y new-y;
x new-x;

    y new-y = y * times originat x, and the new x is the originat
    for a 90 degree clockwise rotation the new y is -

double new-y;
double new-x;
}

void point::rotate90()
{
    y y amount;
x amount;
    }
```
return sum;
    
    point sum(x-sum, y-sum);

    y-sum = (pt1.get-y() + pt2.get-y());
    x-sum = (pt1.get-x() + pt2.get-x());
    
    double x-sum, y-sum;

    }
}

point operator +(const point pt1, const point pt2)
{
    return (pt1 == pt2);
}

bool operator !=(const point pt1, const point pt2)
{
    (pt1.get-y() == pt2.get-y());
    (pt1.get-x() == pt2.get-x());
}


```c
int rotations_needed(point p)
{
    if (p.x > target.x && p.y > target.y)
        return 1;
    else
        return 0;
}

friend of point class

ostream operator >> (iostream ins, point target)
{
    return outs;
    outs << source.get_x() << " " << source.get_y();
}

Library facilities used: iostream
```

University at Albany Computer Science Dept.
// Library facilities used: cmath

double distance(const point p1, const point p2)
{
  p.rotate90();
  white (p.get-y() || p.get-x())
}

void rotate-to-upper-right(const point p)
{
  return answer;
  {
    ++answer;
    p.rotate90();
  }
  white (p.get-y() || p.get-x())
  answer = 0;
}

int answer;

x-middle = (p1.x - x) + p2.x;  
    Compute the x and y midpoints //

double x-middle, y-middle;
    }
}
}
point middle(const point p1, const point p2)
{
    return sqrt(c-square);  // sqrt calculates square root (from math.h)

    c-square = a*a + b*b;
    
Pythagorean Theorem to calculate square of distance between points //

    b = p1.y - p2.y;  // Difference in y coordinates
    a = p1.x - p2.x;  // Difference in x coordinates
    
    calculate differences in x and y coordinates //

    double a, b, c-square;
    
University at Albany Computer Science Dept.
Don't bother with switch namespaces.

```

point midpoint(x, y, midpoint, \y - midpoint)

// Construct a new point and return it

\y - midpoint = (pt.\y(\y) + p2.\y(\y)) / 2;
```
public:
    node * TAIL;
    node * HEAD;
private:
} class Pseudoence

{;
    point *data;
    node *fmd;
} struct node

#include "newpoint.h"

#include <cassert>
#include<iostream> // for cout, endl, etc.
#include <iostream> // for std::cout
#include <stdlib> // for NULL
#define chailken_Pseudoence-h
#include chailken_Pseudoence-h
HEAD = tmp;
tmp->next = HEAD;
tmp->data = pointer;
node *tmp = new node;
assert (pointer);
}

} // a future time

// Warning: Sequence may execute delete pointer; at

// equals the original sequence.
// and our sequence's suffix after the first item
// and our sequence's suffix after the first item in our sequence
// Post: pointer is the first data item in our sequence
// Pre: *pointer is a dynamically allocated pointer object
void new-first(pointer *pointer)
{
    else return HEAD->data;
    if (HEAD) return NULL;
}
```cpp
#endif

};
{
{

tmp=tmp->jwd;
std::cout >> "I just did it to " . std::endl;
#endif

} do
SOMETHING (tmp->data) ;
while (tmp)
node *tmp = HEAD;
}
void do-something-to-every-pointer()
{
```
{ return 0; } mySequence.do-something-to-every-point();

count >>= * mySequence.first() >> end;

mySequence.new-first( new-point(-572.0, 0.003) );

mySequence.new-first( propunt );

propunt*propunt = new-point(95.47,

mySequence mySequence;

} main( )

using namespace std;

<include<iostream>
#include <cstdlib>
#include <cstdio>
#include <sequence.h

Chaisken's sequenceDemo.cxx
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
pointer to (addr of) a char

pointer to (addr of) a node

Another cut

This line's been cut

First main list text line

And line

The third

The Third

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

NULL

N Thông tin core

Need to design any others cursor and pointer to (addr of) a char
Between two and Th3 ??

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 this one?
1. Determine char array length needed, i.e., for "dogs" .
2. Allocate char array, saving its address.
   `tpc=new char[5] ;`
3. Copy C-string into it.
   `strcpy(tpc, "dogs") ;`
4. Allocate a dnode, saving its address.
   `tpn=new dnode ;`
5. Copy char array's address into it.
   `tpn->data=tpc ;`
   `(tpn*).data=tpc ;
(since) which means `tpn->data=tpc ;`
Given the original data structure, how can you change it to this one???
Remember, we want to insert *two* dogs between the two circled nodes. We need the addresses of the 2 circled nodes so we can modify their values.
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 field (data member) names will be omitted in our data structure diagrams.
Alternatively, if we know the address of the circled node, we can program:

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

Therefore, the following code calculates the address of the right node:

\[
\text{tpleft} = \text{tpright} \rightarrow \text{pfront}!
\]

If \text{tpleft} contains the address of the left circled node,
But, how can we obtain ANY one of these addresses?

(1) Starting from value of HEAD or TAIL traverse

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

OR

or search the linked list forward or backward on the "cursor" variable.

dnode * tpleft;
dnode * tpright;

(C++ declarations for these variables)
Now a new dnode containing the addr. of the C-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:
to connect (which is the new dnode)

in               tpn−>fore=tpn;
tpleft−>pback=tpn;

(2) Set 1 link field in each of

in *tpn :    tpleft−>fore=tpn;

(1) Set 2 link fields

into the linked list:

(which is the new dnode) *tpn to connect *tpn

Only 2 more steps remain

Dogs

Two

Th3

NULL

One
The result of step 1 is

IS

tpn->fore=tpn

tpn->back=tpn
The result of step 2.

`tp_left->pfore = tpn;`

`tp_right->pback = tpn;`

**Volla!**

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

The big picture after the changes:

The temporary

The big picture after the changes:
A bigger picture after the changes:
Your job is to solve these programming problems, using the diagrams.

7. Removing a node; and moving it into the cut-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?

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?

Hm... 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:

```c
STT* ps = new STT;
```

Dynamic allocation by:

```c
STT g;
```

Automatic storage allocation by:

```c
{ struct STT int addr; STT* link; } struct definition:
```

selected by field (member) name `addr`.

allocated near each other,

nonuniform types defined by `STT`.

Collection of variables of `struct`.

What's a structure (class, type object)?

What's an array? A:

[Pa][7]

`assert(0 > 2003);`

Selection using array notation:

```c
define a = new etertype[?];
```

Dynamic allocation by:

```c
define a[2003];
```

Automatic storage allocation by:

selected by subscript value.

allocated contiguously,

uniform type,

Collection of variables of an `array`.

University at Albany Computer Science Dept.
struct STT{
    int ddF;
    STT* link;
};

STT G;
STT *pS;
pS = new STT;

STT *ps; ps = new STT[3];
pA = new eleType[3];

eleType A[5];
Calling the "Fake function members":

```
void fake(STT *ps)
{
    count>>dffc://ERROR: dffc not in scope
}

void fake(STT *ps)
{
    count>>ps->dffc;
    count>>(*ps)->dffc;
    count>>dffc;
}
```

What does calling a function member like `fun` really mean?

```
SST C; SST *ps; ps = new SST;
{
    count->fun();
}
```

Suppose struct SST

```
straight C: 
```
by setting a breakpoint on it (you must break main or equality and run first).

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

Tip: 1) If you step into a MESS, the object for which the

function was called.

When debugging inside a C++ function

command steps into functions when they are called, too.

Function calls are performed and return without stopping.

"this" is handy for debugging.

add command next performs line-by-line controlled execution of the current
4. Overloading the prefix ++ operator with a member function X:

\[ \text{prefix++ returns } X.X \]

\[
\text{application programs (\text{C\texttt{tn}} et al.) they were called for (\text{C\texttt{oded by C++ library designers, not}}}
\]

These member functions return a reference to the \text{iostream object (context)}:

\[
\text{iostream \text{iostream::operator}} (\ldots \text{I/O member functions of}}
\]

\[
\text{iostream \text{iostream::operator}} (\ldots \text{I/O member functions of}}
\]

3. Definition of \text{ostream} and \text{iostream}:

\section*{RECOMMENDED FOR BEGINNERS} (i.e., this course).

NOT advanced object oriented styles of data structure programming.

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

assignment operators: See Page 174 of DS0 for details.

1. Detection of self-assignment for proper implementation of overloaded

Some uses of \text{C++ "this" pointer value in programming}:

\section*{Computer Programming unlike skimming, etc.:} is unlikely to break legs, etc., on too difficult