Project 1: In Progress

CSI 310: Lecture 7

on 74-76.

Overloading > for the sorting and rank operations: Optional, and explained.

77-85: Will talk about today.

Overloading operator<< and operator>: Follow the pattern of pages.

are about 4 more projects.

If you don't finish the lecture, test paper will earn up to 65%, and there

(done).

lab is expected. (Not Visual C++ or Borldian then copy to UNIX when it's

Similarly for RCS... Just follow the instructions... development in a UNIX

I:30 today) in HW-25 to teach you as needed.

new things is EXPECTED in College I and TAs hold office hrs (beginning

Yes, this is new to many of you, but it is easy when explained, and done

Separate h, .cxx and compilation...

Project 1: In Progress

university at Albany Computer Science Dept. 1
development offices all over Silicon Valley, NYC, etc.

Pointer/reference/object diagrams are drawn on whiteboards in software.

Data referring to data instead of real things.

Both your old and new data structures with DDD.

Instead of the 10-element static array, test it with test fixture, and visualize this week's lab (DynPileLab): Make your pile class use a dynamic array.
ostream & operator>>(ostream & outs, int n)
{
    So, C++ recognizes the second case of >> as MEANING
    But the TYPE of cout is ostream and the TYPE of n is int.
    operator.
Second case: SYNTAXICALLY, >> is a two-operand, infix (i.e., binary)
    operator that shifts bits left.
First case: C++ recognizes >> means the built-in binary (two operand, infix)
    return:
Invariant: 128 has been printed.
    cout >> ans;
Ans == 128, since 32*2 = 128
Ans = 32 >> 2; // LEFT-SHIFT operator
int ans;
}
main()
using namespace std;
#include <iostream>
the 2nd argument of operator>>() is Ans
the 1st argument of operator>>() is count
In
operator>>() count, Ans
is synonymous with
count >> Ans;
The code
OWN functions get called! Symbols so when you write them applied to YOUR OWN types, then YOUR

Cool() { ++feature: You programmers can overload C++ operators

} //S. Chakraven2006

 university at Albany Computer Science Dept.
(4) outs REFERS to cout
{
  return outs;
outs >> source::minimark();
outs >> source::get-year();
" " >> ( ) source::get-denom();
outs >> source::get-dono();
}
std::ostream &operator>>(ostream &outs, Cout source)
{
  return outs;
}
(2) In operator>(cout, MyCout)
{
  cout >> MyCout " end!
  cout MyCout (5, 1970, M');
}
main()
What happens?
Eventually, a reference to count is returned, so $\gg \gg$ operations can be...
{  
  return outs;
  outs << source::minMarker;
  outs << source::year >> " ";
  outs << source::denom >> " ";
}

std::ostream& operator<<(ostream& outs, Config source) {
  //Private data members with code:
  access private data members with code

  //friend of the Config class. Then the body of operator<<( ) can directly
  operator<<(ostream& outs, Config source) {
  Alternative C++ usage: Make

  // }
changed.

Features of the variable, its VALUE (synonym: state) can be "cloned". Given a unique identity even after it is a person, the variable, shoebox, a variable has a unique identity itself. Like a

The NAME can be copied. The VARIABLE is the OBJECT itself. Like a

**VARIABLE**

4. (This is new...) a NAME can be the VALUE of a

conceptual entities (3 different things).

3. The value, the name, and the variable itself are three DIFFERENT

2. Each variable holds a value (or state).

1. Each variable (synonym: object) has (some kind of) name.

The main points...
Brandon = 98

Adam = 98

Charlie

We will use yellow to denote what has changed.
variables are comprised of different numbers of bytes. Don't worry about this yet...

One big house might cover several adjacent lots. Technically, different sizes of

current value.

of a memory location or byte is very different from the variables.
The addresses (called bytes). The byte numbers are called memory addresses. The addresses

A segment of computer memory consists of numbered memory locations.

house's current occupant.

A block of real estate consists of numbered houses. The house numbers are
called address. The address of a house is very different from the

name that C/C++ code can copy, use, store, etc., at runtime.

A pointer is the memory address of a variable. A pointer is the kind of
The value of this 57 integer variable points to a 4-byte address. The value of this variable is like an array; where addresses function as subscripts.
The pointer variable whose address is 987 is pointing to the above integer variable whose address is 992 and value is 57.

Here's a pointer variable whose value is 992.

Here's the integer variable whose address is 992. Its address is 57.

Real programmers write their addresses in hexadecimal because hex to binary conversion is very easy. You will see hex in Lab2.

I will denote pointer values by black dots because the numeric value is usually boring.

Here's a pointer variable whose address is 992.

Real programmers write their addresses in hexadecimal because hex to binary conversion is very easy. You will see hex in Lab2.

Here's the integer variable whose address is 992. Its address is 57.

Here's a pointer variable whose address is 992.

Real programmers write their addresses in hexadecimal because hex to binary conversion is very easy. You will see hex in Lab2.

You will see hex in Lab2.

Real programmers write their addresses in hexadecimal because hex to binary conversion is very easy. You will see hex in Lab2.

You will see hex in Lab2.

Real programmers write their addresses in hexadecimal because hex to binary conversion is very easy. You will see hex in Lab2.

You will see hex in Lab2.

Real programmers write their addresses in hexadecimal because hex to binary conversion is very easy. You will see hex in Lab2.

You will see hex in Lab2.

Real programmers write their addresses in hexadecimal because hex to binary conversion is very easy. You will see hex in Lab2.

You will see hex in Lab2.

Real programmers write their addresses in hexadecimal because hex to binary conversion is very easy. You will see hex in Lab2.

You will see hex in Lab2.

Real programmers write their addresses in hexadecimal because hex to binary conversion is very easy. You will see hex in Lab2.

You will see hex in Lab2.

Real programmers write their addresses in hexadecimal because hex to binary conversion is very easy. You will see hex in Lab2.

You will see hex in Lab2.

Real programmers write their addresses in hexadecimal because hex to binary conversion is very easy. You will see hex in Lab2.

You will see hex in Lab2.

Real programmers write their addresses in hexadecimal because hex to binary conversion is very easy. You will see hex in Lab2.

You will see hex in Lab2.

Real programmers write their addresses in hexadecimal because hex to binary conversion is very easy. You will see hex in Lab2.

You will see hex in Lab2.

Real programmers write their addresses in hexadecimal because hex to binary conversion is very easy. You will see hex in Lab2.

You will see hex in Lab2.

Real programmers write their addresses in hexadecimal because hex to binary conversion is very easy. You will see hex in Lab2.

You will see hex in Lab2.

Real programmers write their addresses in hexadecimal because hex to binary conversion is very easy. You will see hex in Lab2.

You will see hex in Lab2.

Real programmers write their addresses in hexadecimal because hex to binary conversion is very easy. You will see hex in Lab2.

You will see hex in Lab2.

Real programmers write their addresses in hexadecimal because hex to binary conversion is very easy. You will see hex in Lab2.

You will see hex in Lab2.
POINTER VALUES, and call them "reference".
Unlike C/C++, Java and Perl (except debuggers) hide all numeric
systems, sometimes within I/O devices (memory mapped ones).
The memory bus addresses usually locate data within cached RAM hardware
addresses and sometimes into page fault signals.

BUS or physical memory management units sometimes into hardware memory
Hardware-SW interface (CS1333).

Numerical (binary) virtual addresses are quickly translated by

POC/Workstation/Servers with Unix-like/WindowNT operating systems
a few background words... In late 20th century technology of
2. Initialized.
1. Uninitialized.

The array of 6 piles used in test.

2. Initialized.
1. Uninitialized.

Determination array used in letter—

3. Read in from cin.
2. Initialized.
1. Uninitialized.

One Coin: 

We now use DDD to visualize
MyCoin denomination = 65340
year = 0
mintmark = 0 '000'

MyCoin denomination = 5
year = 1970
mintmark = 0 '000'

MyCoin denomination = 65340
year = 1970
mintmark = 0 '000'

3. Code a constructor and member functions to REPLACE the current array

```cpp
};

int Numbse;
int Arraycapacity;
Coin *party;
}

class pile

2. THE DYNAMIC PILE has 3 data members:

1. SEPARATE OUT the array of Coins from the Pile structure.

Solution 2: Use dynamic allocation here.

belonging to most people.

(1) The huge fixed allocation make the software unusable on the smaller systems
datasets make the software unacceptable slow on most uses.
(2) The time overhead for allocating huge memory spaces for occasional large
Not acceptable for „Real“ applications.

Solution 1: Replace 10 by 100. Or 1,000. Or 1,000,000?

It imposes the limitation that a Pile may not hold more than 10 Coins.

Problem: This data structure has static size.
```
Such member function MUST copy the used coins from the old array when it replaces it with a new array.

With a NEW array that is bigger.
```cpp
#include <iostream>

using namespace std;

float x = 98.6;

int main()
{
    cout << "The VALUE of x is " << x << endl;
    cout << "The ADDRESS of x is " << (void*)x << endl;
    cout << "The ADDRESS of x is " << (void*)&x << endl;
    return 0;
}
```
The type of variable that it IS a pointer variable, and the declaration of a pointer variable specifies BOTH dereferencing operator...hmm...

(2) the TYPE of those variables it can point to.

(1) that it IS a pointer variable...

The type of Variable variable can hold the address of that the pointer variable.

The name of the newly declared pointer.

An asterisk (which, by coincidence, is also the C/C++ point to (the type of data it can point to)

The name of the newly declared pointer.

float * MyFirstPointer;

Learn how C++ pointer variables are declared.

Important for USING pointers!
Why? 2003 can change while MyP still points to MyInt.

MyP does NOT point to 2003.

Now we can say "MyP points to MyInt."
ALIASES.

General problem: names (different kinds).

Huh?! One variable has two different names (different kinds).

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

The * operator dereferences a pointer (or address value).

Important for USING pointers!
MyP

Before Multiplication

2003
MyP

After Multiplication

4006
MyP

\* MyP = \* MyP * 2;

\* MyP = \* MyP * 2;

4006
MyP

0xbe0fef04

count >> MyP >> end;

count >> MyP >> end;

count >> MyP >> end;

count >> MyP >> end;
This is a "CRASH": Computer tried to read memory at the illegal address 0x0.

Segmentation fault

count >> *myPtr >> endl;

0x0 0xbe0fef04

count >> *myPtr >> " " count >> *myPtr >> " "

4006 count >> *myPtr >> endl;

After Assignment:

```
    4006
    myInt
```

```
    0x0
    NULL
```

```
    myPtr = NULL;
```

Before Assignment:

```
    4006
    myInt
```

```
    0xbe0fef04
    myPtr
```

© S. Chakken 2006

University at Albany Computer Science Dept.
try {xxx: ? non-`value` in unary
    } (function, `x` main
    try {xxx: ?
        $ $ + - $ try
    } xxx

----------

{ }

cout >> `g(x)` { main
    int `x` = 3;
}

using namespace std;

#include <iostream>

Expressions, like `(x+3)`, are not variables, so:
A POINTER VARIABLE can store a pointer value.
Please simulate the execution of this by writing the values currently in each of

\[
\text{TEMP} = \text{TEMP} \times p_2 \\
\text{p}_2 = \text{p}_2 \\
\text{p}_1 = \text{p}_1 \\
\text{TEMP} = \text{TEMP} \\
\text{p}_1 = \text{p}_1 \\
\text{p}_2 = \text{p}_2 \\
\text{TEMP} = \text{TEMP} \\
\text{p}_1 = \text{p}_1 \\
\text{p}_2 = \text{p}_2
\]

How can you program swapping the values of i and j using pointers?

\[
i = 1928; \quad j = 2003 \\
i = i; \quad j = j
\]
assert (p1 == &I (p2 == &I))

1928 2003

cout >> J >> I >> p1 >> p2 >> endl;

2003 1928

cout >> p1 >> p2 >> endl;

p2 = TMPp;
p1 = p2;
TMPp = p1;
int *TMPp;

p1 = &I; p2 = &I;
int *p1; int *p2;
int J; int I; I = 1928; J = 2003;

Now, how is swapping the values of the pointer variables different?
myString = new String(...);
}

String *myString, NOT Java
is equivalent in C++ to:

myString = new String(...);
// copy return value to myString

Tip for the Trender: The following in Java:

What is a pointer value (synonyms: address, locator, Reference in Java)?

What is a variable (synonyms: object, memory location, cell, box for

Summary...
Java has only 2:  

1. Primitive type (roughly like C/C++'s but implementation dependent)  
2. Reference type, which are like C/C++ pointers (to class objects)
3. Pointer type (like * in C/C++)

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

- Variables/addresses
- Numerical pointer values/address
- Exposed numerical pointer

C/C++ hides machine language assembly