4. Chap. 3 and 4 on some abstract data types.

3. Sec. 4.1 of DS0: POINTERS

2. Recap of pre- post-conditions and invariants.

1. The difference between a URL written in an HTML document, the information within that document delivered to the Web browser.

0. Lab 2 Assignment on Pointers was Here. BASIC IDEA.

CSI 310: Lecture 5
// Found Bill's demonstation.
// Remark: The copied data indicates the
// that serial number is copied to Bill.
// that serial number. The Bill closest to the top with
// Returns true if there is a Bill in the pile with
// with serial number equal to serialization in the pile.
// Post: Returns false if there isn't any Bill

bool Find(long unsigned int serialization, Bill bill)
{ }
{ }
  \text{\bf if} \ I == 0 \ \text{\bf return} \ \text{\bf false} ; \\
  \text{\bf else} \\
  \text{\bf if} \ I \text{\bf return} \ \text{\bf true} ; \\
  \text{\bf else} \\
  \text{\bf for} \ \text{\bf I} = 1 \text{\bf to} 2 \\
  \text{\bf then} \ \text{\bf that} \ \text{\bf BIII} \ \text{\bf is} \ \text{\bf somewhere} \ \text{\bf in} \ \text{\bf A} \[ \text{\bf 0} , \text{\bf I} - 2 \] \\
  \text{\bf if} \ \text{\bf there} \ \text{\bf is} \ \text{\bf a} \ \text{\bf BIII} \ \text{\bf in} \ \text{\bf this} \ \text{\bf pile} \ \text{\bf with} \ \text{\bf serial} \ = \ \text{\bf serial}^\text{\bf prefix} \text{\bf stamp} \\
  \text{\bf import} \\
  \text{\bf with} \ \text{\bf while} \ ( \text{\bf get-serial} ( \text{\bf A} [ \text{\bf I} - 1 ] ) > \text{\bf serial} = \text{\bf serial}^\text{\bf prefix} ) \\
  \text{\bf size} - \text{\bf if} \ \text{\bf I} \ \text{\bf used} ; \\
  \text{\bf end} \\
  \text{\bf bool} \ \text{\bf pile} : \text{\bf find}(\text{\bf long unsigned} \ \text{\bf int} \ \text{\bf serial} , \ \text{\bf BIII} , \ \text{\bf prefix} \text{\bf BIII} )
to that instance.

to that instance. FOR which the function is called. It is a C++ pointer.

Advanced notice: “this” is a reserved C++/Java keyword that “refers”

which find () was called. Here it is element 3 of the ptte array pt.

refer to the “a” and “used” data fields in that INSTANCE. FOR

The names “a” and “used” in the body of both ptte: find ( )

pt [3] : find (37822, p) :

bt1 b ?: ptte pt [6] ;

type is that class. Thus:

always called FOR a particular variable (synonym: instance) whose

An (instance) member function (synonym: method) of a class is
VALUE (synonym: state) can be changed.

Identity even after it is "cloned". Features of the variable, is itself. Like a person, an automobile, a shoebox, a variable has a unique name. The variable can be copied. The variable is the object.

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

3. The value, the name, and the variable itself are three different entities (3 different things).

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

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

The main points...
Brandon = 98;
One big house might cover several adjacent lots. Technically, different
sizes of variables are comprised of different numbers of bytes. Don’t
worry about this yet...

DIFERENT FROM the variables’ current VALUE.

DIFERENT FROM the house’s current OCCUPANTS.

A block of real estate consists of numbered houses. The house
number are called addresses. The address of a house IS VERY
A segment of computer memory consists of numbered memory
locations (called bytes). The byte numbers are called memory
addresses.

A pointer is the memory address of a variable. A pointer is
The value of this 57 variable is integer variable.

The value of this 57 variable is pointer value or points to a 4-byte address.

57
Pointer values, like any other numbers, can be stored in variables. Here's a pointer variable whose address is 992. The pointer variable whose address is 992 and value is 57 is pointing to the above integer variable whose address is 987 and value is 57.

Real programmers write their addresses in hexadecimal (base sixteen) because hex to binary (base 2) 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.

The pointer variable whose address is 987 is pointing to the above integer variable whose address is 992 and value is 57.
NUMERICAL "POINTER VALUES," and call them "references."

Unlike C/C++, Java and Perl (except debuggers) hide all

ones.

Hardware systems. Sometimes within I/O devices (memory mapped

The memory bus addresses usually locate data within cached RAM

signals.

memory bus or physical addresses and sometimes into page fault

Hardware memory management units sometimes into hardware

Numerical (binary) virtual addresses are quickly translated by

the hardware-software interface (GSI333).

Numerical pointer values are virtual addresses, and part of

systems:

PC/Workstation/Servers with Unix-Like/Windows

a few background words… In late 20th century technology of
$ 

The VALUE of X is 98.6

The ADDRESS of X is 0x35799b4a

d $ 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>

operand, which must be a variable.
The C/C++ % operator provides the ADDRESS OF its
The C/C++ compiler uses this formula for you. You won’t need it.

Formula: \( A[I] = (A[I][1]) = (A[I]) \cdot \text{base} + \text{size} \)

The computer can quickly compute the address of \( A[I] \).

1) the address of the first element of an array.
2) the size of each entity (number of bytes), and the element index.
3) the size of each entity (number of bytes), and the address of the element

So, given \( A[I] \), we can access to the \( I \)-th entry with \( A[I] \).

It is a variable, comprised of a sequence of entities varriables of \( A[I] \).

What is an array? [4] [3] [2] [1] [0]
An array element is a variable, its address

float x[3] = { -37.6, 15.3, 98.6 };
```
try: cxx: In function `main()

    $ E += E - E

----------------

    { 

        cout >> x3; 

    int x = 3;

} 

main

using namespace std;

#include <iostream>

Expressions like (x+3), are not variables, so:
```
A pointer variable can store a pointer value.
Learn how pointer variables are declared.

The type of variables it can point to.

1) that it IS a pointer variable, and
2) the type of those variables 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

variable
declared pointer
The name of the newly

float * MyFirstPointer;

The type of data it can hold the address of that the pointer variable

The type of variable variables are declared.
MyP does NOT point to 2003.

Now we can say "MyP POINTS TO MyInt."

2003
MyInt

MyP

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

MyP = &MyInt;

MyInt = 2003;

MyInt is the address of the int variable MyInt.

MyP = &MyInt;

MyInt = 2003;


Produces in the computer’s memory:

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

Means

Pointer variable MyP POINTS TO int variable MyInt.
0xbe0fef04

cout >> MyP >> endl;
cout >> *MyP >> endl;
cout >> MyInt >> endl;

MyInt

2003

MyP

0xbe0fef04

value.
orem the variable whose address is the dereferenced address.

dereferenced pointer

That means: Access the variable whose address is in the

dereference operator * or address value.

The
Before Multiplication

0x0be0f4

0x0be0f4

count >> Myp >> end;

4006

4006

count >> Myp >> end;

count >> Myp >> end;

Myp = *Myp * 2;

After Multiplication

4006

Myp

0x0be0f4

2003

Myp

0x0be0f4
This is a "CRASH": Computer tried to read memory at the illegal Segmentation fault

cout >> *Myp >> endl;
0x0 0xbe0fef04

cout >> "" >> Myp >> "" >> endl;
4006
cout >> Myp >> endl;

After Assignment:

MyInt

4006

NULL

Myp

0x0

Before Assignment:

MyInt

4006

NULL

Myp

0xbe0fef04

Myp = NULL;
... currently in each of the 3 variables, before and after each step.

Please simulate the execution of this by writing the values

```
*pt = TEMP;
*pt = *p2;
TEMP = *p1;
```

```
int TEMP;
p2 = &j;
p1 = &l;
int *p1; int *p2;
int *pt; int *pt;
```

Pointers?

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

```
I = 1928; J = 2003;
int I; int J;
```
assert (p1==q1) \&\& (p2==q2);

1928 2003
count >> I >> J >> end;

2003 1928
count >> *p1 >> *p2 >> end;

p2 = TMPp;
p1 = p2;
TMPp = p1;
int *TMPp;
p1 = q1; p2 = q2;
int *p1; int *p2;
int I, J; I = 1928; J = 2003;
different

Now, how is swapping the values of the pointer variables
The diagram shows an array access. The array is defined as follows:

```
[0] [1] [2] [3] [4]
```

- `p1[1] = p1[0] + 2;`

The code snippet given is:

```c
count >> p1[0] >> end.
printf "p1 = \%d\n", *p1;

int *p1;'`

If `p1` is a pointer variable whose value is the address of an int array, then `p1[0]` accesses the first element of the array.

Dereferencing a pointer `p1` using array notation:

```
p1[0]
```
cout >> PAT[3].FLOW() >> endl;

PAT[1].shitt(2);
PAT[0].shitt(1);

PAT = the address of some array of throttles;

throttle *PAT;

This works for an array of ANYTHING, e.g.
“Used”.

3. Its current length: How much of the “Capacity” is currently

2. Its maximum usable length “Capacity”.

1. Data array.

structure the

overcome the vulnerability of C-strings by combining in one data

The partially-而不 array type classes of DSO chapter 3 (and 4)
STL in 310 projects is forbidden until further notice...around after the midterm.

1. C++’s Standard Template Library includes such software structures/algorithms that truly knowledgeable professionals and computer scientists must understand.
2. Naive programmers will encounter enormous run time penalties on large data sets.
3. This course and DSO introduce how to use it, but scientific understanding has higher priority for us...too.

Java and Perl arrays resize themselves automatically when large enough subscripts are first used. But...
algorithm runs new to allocate an BIGGER array.
allocated partially filled array. When it fills up, the insertion
practically unlimited. The impl. data structure is a dynamically
"chapter 4 improves this bag class, so the number of items is
array of items.
The implementation data structure is a fixed size partially filled.
can hold up to a fixed number of items (constant multiplications).
"chapter 3 teaches a primitive bag class, for which each bag
Roadmap:
the program runs (malloc and free in C). Explicitly coded operation (new and delete) executed when
But a dynamic variable is created or destroyed only by an
returns.
function is called and are destroyed (storage recycled) when it
a local extant (C++ automatic) variables are created when a
Dynamic variables are created.
Unpredictable input data can determine if and how many
Dynamic variables are created during process execution.
Dynamic variables are not declared. They are not named by
identifiers. But they, like all variables, have memory addresses.
Dynamic variables allocated variables
(You need pointers to access them!)
Dynamically Allocated Variables
new is a keyword in C++. See DSO FIGURE 4.1

```c
PAT = new throttle[5];
throttle *PAT;
To allocate an array of 5 throttles:

PAT = new int[4];
array form for new

To allocate an array of say 4 ints, do the same except use the

PAT = new int;
and then make program execute:

e.g.: int *PAT;

a pointer variable to hold its address, 

To allocate ONE variable of any type, say int, make sure you have

Doing dynamic allocation: use new.
3. The total size of this dynamic array is the value of capacity.
   First element. It is a dynamic array where the value of data is the address of its
2. The actual items in the bag are stored in a partially filled array.
1. The number of items in the bag is the value of used.

Invariant for the Revised Bag Class

```java
;

size-type capacity; // current capacity of the bag.
size-type used; // how much of the array is used.
value-type *data; // pointer to dynamic array.

private:
   ... public:
}

class Bag
```
in ch. 3.

store the current position. (also
plus variable current-index to
SAVE as for bag!

Partially filled array:

```

// implementation:
```

a critical part of the class's
The invariant of the class is:
Key Design Concept:
```

class, invariant of the class is
```
```

```

...}

insert ()
advance ()
current ()
start ()
```

```

...}

operator+()=
insert ()
erase-one ()
```

```

// multiple
```

... 0 1 2 3

How many times does $x$ appear in $W$?

Given an item $x$,

MULTISET $M$ (another name for "bag")

YES

Has a well-defined answer:

"Is $x$ in $S$?"

Given an item $x$,

SET $S$

What is a bag? Mathematicians say "finite multiset"
<table>
<thead>
<tr>
<th>Item number</th>
<th>Name</th>
<th>Item</th>
<th>No. orders</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Grilled cheese sandwich</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>pot of hot &amp; sour soup</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Egg roll</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Szechuan chicken</td>
<td>72</td>
<td></td>
</tr>
</tbody>
</table>

Good example: What a Chinese restaurant serves:
Implementations (They often vary in efficiency).

The same abstract data type can have very different implementations.

**Abstract Data Type**

**Implementation**

An **ALTERNATIVE** implementation for various variants is the linked-list (to be taught) is a partially filled array.

```
  { ... }
```

A **keyed-bag** is of the form:

```
  { ... }
```

Partial exercises are stored in an array.

```
  { ... }
```

With the receipt stored in with bag-with-receipts, see ch. 3.

**Implementation**

More abstract data type/implementation combinations.