Maybe more...

(32) Introduction to C++ delete operators

26-31 Dynamic Variables: Dynamic Bag

(22-25) Static Bags: Details and Sec. 4.3-4.4

Various Implementations.

Lecture 7:

17-21 Bag and a few other Abstract Data Types COMBINED WITH

14-16 Partially Filled Array

(2-13) Pointers continued ... Lab 2 supports this material

CSI 310: Lecture 6 (No Lab assignment: Use Lab time for TA help with Prof. 1.)
myObject = new MyObject();

*Tip for the Tryout: The following in Java:

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

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

Going back to pointers, again...
The values of $x[0..2]$ are

The addresses of $x[0..2]$ are

```c
int x[3] = {-37.6, 15.3, 98.6};
```

main() {  
  cout << "The values of x[0..2] are " << endl;  
  cout << "The addresses of x[0..2] are " << endl;
}
```
try::cxx:7: non-template in unary

:: (int main) {

try::cxx: In function 'int main'

try

$ 

$ + - 

$ + - 

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

{ 

cout >> g(x+3) << endl;

int x = 3;

}

main()

using namespace std;

#include <iostream>

Expressions, like (x+3), are not variables, so:

C.S. Chakravarthi

University at Albany Computer Science Dept.
A POINTER VARIABLE can store a pointer value.
(2) Java "Reference type" which are like C/C++ pointers (to class objects independent)

(1) Primitive type (roughly like C/C++'s but implementation

Java has only 2:

(3) Pointer type (pointer to type, int *pointer;)

(2) Struct/class type (struct type, int, char, float, etc.)

(1) Primitive type (int, char, float, etc.)

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

called Java reference variables, to hold and copy Java reference values.

Java HIDEs numerical pointer values/addresses and provides variables/values/addresses.

C/C++/assembly/machine languages EXPOSE numerical pointer
The declaration of a pointer variable specifies BOTH:

(1) that it IS a pointer variable, and
(2) the TYPE of those variables it can point to.

The type of variable that can hold the address of (the type of data it can point to)

The name of the newly declared pointer variable

MyFirstPointer = &float

(Which, by coincidence, is also the C++ dereferencing operator...hmm...)

an asterisk

(point to)

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

The name of the newly declared pointer variable

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

MyP does NOT point to 2003.

Now we can say "MyP points to MyInt."

```
2003
MyInt
```

```
MyP
```

When we declare:

```
int *MyP;
```

Computer's memory produces in the computer's memory:

```
MyP = &MyInt;
MyInt = 2003;
```

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

**MEANS**

```
MyInt = 2003;
MyP = &MyInt;
int MyP;  // Not int *MyP;
```
Huh? One variable address value (or is the * operator dereference or address value).

That means: Access the variable whose address is in the pointer (or its the address value). Important for *sing pointers!

General problem: ALIASES. Names (different kinds) has two different variable names (different kinds).

\[ \text{count} \rightarrow \text{MP} \]
After Multiplication

Before Multiplication

\[ \text{MyInt} = \text{MyInt} \times 2; \]

\[ \text{MyInt} = 4006; \]

\[ \text{MyInt} = 2003; \]
so it could print the int value located there.

This is a "CRASH": Computer tried to read memory at the illegal address 0x0,

Segmentation fault

cout >> *MyP << endl;
cout 0x000000004

cout >> " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " 

After Assignment:

        0x000000004
         NULL
         MyP
MyInt

After Assignment:

Before Assignment:

        0x000000004
         NULL
         MyP
MyInt

MyP = NULL;

cout >> MyInt << endl;

cout >> MyInt >> " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " 

©S. Chakken2004

University at Albany Computer Science Dept.
Library container classes for sequences.

Java Doesn't do this. Java arrays are "first class" like C++ standard.

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

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

cout >> p1[0] >> endl; // prints 7

p1 = p1 + 2;
int* p1 = p1;

int array[5] = {0, 1, 2, 3, 4};
int array_entity[10] = {0, 1, 2, 3, 4};

if (p1 is a int pointer variable whose value is the address of an int array, then p1[0] access entity 0, p2[1] access entity 1, etc.)

De-referencing a pointer p1 using array notation:
```
So \( \text{PAT} \rightarrow \text{null} \); \( \text{PAT} \rightarrow \text{shift}(1) \); CrashES the process.

or function member of the located object.

\( \rightarrow \) "hyphen less than" DEREFERENCE the pointer and SELECTS the data

\( \text{PAT} \rightarrow \text{shift}(1) \); is EQUVALENT TO \( \text{PAT} \ast \). shift(1).

Here is an alternative, very popular C++ syntactic synonym:

\( \text{PAT} \rightarrow \text{shift}(1) \); is EQUVALENT TO \( \text{PAT} \ast \). shift(1).

Note, the value of \( \text{PAT} \) is the address of the 0th Throttle, so

\( \text{count} \ll \text{PAT}[3].\text{low}() >> \text{end} \);

\( \text{PAT}[3].\text{shift}(2) \);

\( \text{PAT}[2].\text{shift}(1) \);

\( \text{PAT}[1].\text{shift}(1) \);

\( \text{PAT}[0].\text{shift}(1) \);

\( \text{PAT} = \) the address of some array of throttles;

\( \text{throttle} \ast \text{PAT} \);

This works for an array of ANYTHING, e.g.

\( \odot \text{S. Chakravarty} \)
they always begin at position 0, so you have to keep track of the end.

It's easy: THE SAME array holds different length sequences at different times.

Project 1: You implemented the deck and each pile using a fixed capacity (of

is the Partially Filled Array, of chapters 3 and 4 of DSO.
The first data structure/algorithm useful to implement certain container classes.

devour, efficient data structure/algorithm implementation examples.

will concentrate on container classes both as abstract data types and for

thought and point are examples of concrete classes, very helpful. CS1310

©.Chai ken 2004

University at Albany Computer Science Dept.
3. Its current length: How much of the "Capacity" is currently "Used".

2. Its maximum usable length "Capacity".

1. Data array: one data structure the
the vulnerabilities of NUL char terminated "C-string" arrays by combining in
the classes based on partially-stored arrays of DS0 chapter 3 (and 4) overcome

© S. Chakravorty 2004
midterm.

STL in 310 projects is forbidden until further notice. Avoid STL in 310 projects is forbidden until further notice. Avoid

Priority for us.

and DSO introduce how to use it, but scientific understanding has higher

3. C++'s Standard Template Library includes such software too. This course

data sets.

2. Native programmers will encounter enormous run time penalties on large

truly knowledgeable professionals and computer scientists must understand.

1. Their runtime environment software uses data structures/algorithms that

subjects are first used. But...

Java and Perl arrays resize themselves automatically when large enough
\((\mathcal{H}, \mathcal{A})\) says \(\mathcal{H} \mapsto \mathcal{A} \mapsto \mathcal{H}\). \(\mathcal{H} \mapsto \mathcal{A}\) is a function with domain and range \(\mathcal{H}\). \(\mathcal{H} \mapsto \mathcal{A}\) is finite means the sum of the \(n\) sum is finite.

Mathematical formalizations: Set \(\mathcal{H}\) but Multiset

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & \cdots & \\
\end{array}
\]

has a well-defined answer:

\[
W \mapsto W, \text{How many times does } x \text{ appear in } W?
\]

given an item \(x\),

Multiset \(W\) (another name for bag)

YES

NO

has a well-defined answer:

\[
\begin{array}{ccc}
\text{Is } x \text{ in } S? \\
\text{given an item } x, \\
\text{Set } S
\end{array}
\]

What is a bag? Mathematicians say "finite multiset"

(Lecture 07 starts here.)
<table>
<thead>
<tr>
<th>Item number</th>
<th>Name</th>
<th>No. orders</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Grilled cheese sandwich</td>
<td>99</td>
</tr>
<tr>
<td>3</td>
<td>Hot &amp; sour soup</td>
<td>27</td>
</tr>
<tr>
<td>8</td>
<td>Egg roll</td>
<td>19</td>
</tr>
<tr>
<td>5</td>
<td>Szechuan chicken</td>
<td>72</td>
</tr>
</tbody>
</table>

Good example: What a Chinese restaurant waiter serves:
to store the current position, also plus variable current-index to
SAME as for base:
Partially filled array:

(3.)

in ch. 3.

the invariant p. 103-4:

(4.)

the invariant p. 103-4:

Study ch. 3 details; especially
used first empty position.
Array of items; plus variable
Partially filled array:
Implimentation Data Structure

Sequence matters

Abstract Data Type
The same abstract data type can have very different alternative implementations. They often vary in efficiency.

<table>
<thead>
<tr>
<th>Bag of</th>
<th><code>a b</code></th>
<th><code>c</code></th>
<th><code>d</code></th>
<th><code>e g f</code></th>
<th><code>d h i j f</code></th>
<th><code>^</code></th>
<th><code>e l k m n</code></th>
<th><code>o l p d</code></th>
<th><code>a b</code></th>
<th><code>q r</code></th>
<th><code>^</code></th>
<th><code>a r</code></th>
<th><code>a s</code></th>
<th>`a o</th>
<th><code>t</code></th>
<th><code>r u q t k v f</code></th>
<th><code>w</code></th>
<th><code>^</code></th>
<th><code>x</code></th>
<th><code>y</code></th>
<th><code>y z</code></th>
</tr>
</thead>
</table>
| Array  | `f`   | `g h` | `i j` | `h k` | `j n o h` | `j x o y` | `o j o j o z` | `u x j n t i u i u h` | `z` | `} {` | `u v` | `u v` | `u n o j o k g n m ~` | `st u z` | `g f o u n i j h z k n u € x k u n x` | `zV"` | `'

The linked-list (to be taught) is

- Partially filled array.
- Partially filled array.
- Keyed bag

```
{ · · }
```

```
{ · · }
```

```
{ · · }
```

```
exercises
```

```
" // see ch. 3
```

```
• · · }
```

Abstract Data Structure

Implementation/Implementation combinations.

More abstract data type/implementation combinations.
Roadmap:

DSO chapter 3 teaches a primitive bag class, for which each bag can hold up to a fixed number of items (counting multiplicities). The implementation data structure is a fixed size partially filled array of items.

DSO chapter 4 improves this bag class, so the number of items is practically unlimited. The impl. data structure is a dynamically allocated partially filled array. When it fills up, the insertion algorithm runs new to allocate an BIGGER array.
used have [local] meanings.

In the scope of class bag, value-type, size-type, capacity, data and

```cpp
size-type used;
value-type data[capacity];
```

private:

```cpp
...;
```

NOTE: No other public data members!

```cpp
static const size-type capacity = 30;
```

```cpp
typedef std::size_t size-t;
typedef int value-t;
```

```cpp
public:
```

```cpp
class bag
```

```cpp
...;
```

```cpp
#include <cstdlib>
```

```cpp
#include <cassert>
```

```cpp
FILE: bag.h
```

© S. Chakken 2004

University at Albany Computer Science Dept.
Instances of class bag

private:
    value_type data[CAPACITY]; // The array to store items
    size_type used;            // How much of array is used
};

NOTE: No other public data members!

static const size_type CAPACITY = 30;

initialized only ONE, initialized at startup.

class bag
{
    static const size_type CAPACITY = 30;
    static const size_type size_type data[CAPACITY];
    static const size_type used; // The array is used

    private:
        ...;

    public:
        ...;

    private:
        ...;

    public:
        ...;

    };
Outside the scope of class bag, you must code the scope qualifier to refer to \texttt{bag::CAPACITY}. Within the scope of class bag, \texttt{bag::fun(...)} refers to our class (static) variable, \texttt{CAPACITY}.

Outside the scope of class bag, say, within a member function, \texttt{ret-type bag::fun(...)}:

\begin{verbatim}

private:

    size-type used; // How much of array is used
    value-type data[CAPACITY]; // The array to store items

public:

    \ldots

\end{verbatim}
size > bag::CAPACITY
 precondition for insert():
\[\text{size} + \text{size} = \text{CAPACITY}\]
precondition for operator+(bag b1, bag b2):
\[\text{size} + \text{size} = \text{CAPACITY}\]
data [0] through data [used-1]. (We don’t care about the rest of data[].)
2. For an empty bag, used = 0. For a non-empty bag, the items are stored in
the number of items in the bag is the value of used.
Invariant for the Primitive Bag Class:

; size-type used;
  // How much of the array is used.
value-type data[CAPACITY]; // The array to store items

private:
  ...
public:
}
class Bag
FILE: bag1.h //
multitudes of Java Library and Language Classes are dynamic.

FYI: All of the objects (variables, instances) of the interesting and full featured

( matrix and free in C).

coded operation (new and delete) executed when the program runs.

But, a dynamic variable is created or destroyed only by an explicitly
called and are destroyed (storage recycled) when it returns.

3. Local extant (C++ automatic) variables are created when a function is
input data can determine if and how many dynamic variables are created.

2. Dynamic variables are created during process execution. Unpredictable

But they, like all variables, have memory addresses.

1. Dynamic variables are not declared. They are not named by identifiers.

(You need pointers to access them!) In Java you need „references“.

Dynamically Allocated Variables
new is a keyword in C++. See DS0 Figure 4.1

\[
\text{PART} = \text{new throttle}[5];
\]

\text{throttle* \text{PART};}

To allocate an array of 5 throttles:

\[
\text{pi} = \text{new int}[4];
\]

new

new

To allocate an array of say 4 int’s do the same except use the array form for

\[
\text{pi} = \text{new int}[	ext{4}];
\]

and then make program execute:

\text{e.g.: int* \text{pi};}

variable to hold its address.

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

Doing dynamic allocation: use \text{new}.
executed.

Remark: That length was determined when new value=type[capacity] was
3. The length (#) elements of this dynamic array is the value of capacity.
dynamic array, where the value of data is the address of its first element.
2. The actual items in the bag are stored in a partially filled array. It is a
1. The number of items in the bag is the value of used.

Invariant for the Reversed Bag Class

private:

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.

public:

class bag
3. The total size of the dynamic array is the value of capacity.

dynamic array, where the value of data is the address of its first element.

2. The actual items in the bag are stored in a partially filled array. It is a

Invariant for the Reversed Bag Class

{ size_t type capacity; // Current capacity of the bag.
size_t type used; // How much of the array is used.
value_t *data; // Pointer to dynamic array.

private:
...;

public:
}

class Bag

FILE: bag2.h //
The improved bag uses a dynamically allocated array:

```
{  
    size_type capacity;
    size_type used;
    value_type *data;

    public:  
      static const size_type CAPACITY = 30;
      public:
      class Bag2 
        
      public:  
      class Bag1 
        
      public:  
      private:  
      
      public:  
      private:  
    
    Bag2 MyBag2;  
    Bag1 MyBag1;  
  }
```

Lots of Garbage

−9

30

The improved bag uses a dynamically allocated array:
The (1) part is retained.

(2) New value-type [bigger] to allocate a bigger array to replace (2).

When a bag is full, the insert and operator methods run

(2) a dynamically allocated dynamically-sized array to actually hold the items.
(1) a statically-sized structure, and

Each bag consists of a statically-sized data structure only. Its capacity is

Limited to 30 items. Memory is wasted when the bag has few items.

Each bag consists of a statically-sized data structure only. Its capacity is
Advice: Don't rely on case 1 always happening!!! NEVER rese the pointer.

After calling delete()

(2) The new tenants will be very annoyed by what your friends like to do.

or people in what they think is your apartment.

These consequences might occur: (1) Your friends will find wrong things

Case 2: Somebody else DID move in. There will be trouble!! Either or both

Case 1: Nobody else moved in. Nobody will know the difference!

What if you send your friends a copy of your old address and invite them to a

party there AFTER you gave your apartment up?

Now, what if a pointer to a "deleted" variable is dereferenced?

Future calls of the new operator will obtain memory space from the free store.

as recyciling of variables.

Also think of it

as putting a preassigned unused apartment or dorm room back on a

realtor's list of AVAILABLE SPACES, called the free store. Also think of it

(Main and Savage, p. 151)

unprocessed for dynamic variables.

"The delete operator frees memory that has been use for dynamic variables."

S. Clark, 2004

University at Albany Computer Science Dept.

32
First swapping...

EXTRA STUFF
Please simulate the execution of this by writing the values currently in each of
the 3 variables, before and after each step.

TIMP;
I = J;
J = I;
TIMP = I;

That cannot work!!

Try first J = I; or first I = J; How can you program swapping the values of I and J?

J = 2003;
I = 1928;
J = J;
I = I;
the 3 variables, before and after each step...

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

\[ \text{temp} = \text{temp} \]
\[ \text{p1} \]
\[ \text{p2} \]

int temp;
int p2 = 42;
int p1 = 41;
int * p1; int * p2;

How can you program swapping the values of 1 and 2 using pointers?

\[ i = 1928; \quad j = 2003; \]
\[ i = j; \quad j = i; \]
assert (p1==q1) (p2==q2)
1928 2003
cout >> I >> J >> end;
cout >> *p1 >> *p2 >> end;
p2 = TEMP
p1 = p2;
TEMP = p1;
int *TEMP
p1 = q1;
int *p1; int *p2;
int J; int J; I = 1928; J = 2003;

Now, how is swapping the values of the pointer variables different?
Perhaps we should always use the word "address" for "pointer value".

done.

Type which determines what values it can hold and what operations can be

Technically, "pointer" and "int" describe C/++ types. Each variable has a
```cpp
{ return 0;

// while function failed or empty string was read.
while (true) {
    // if sorted string was printed.
    cout << A >> endl;
    // a sorted string was printed.
}

// now, A[0..n-1] is sorted

// now, A[t] has the smallest char from A[t..n-1]

{ }

// code to "swap" A[t], A[f]

if (A[f] < A[t]) {
    for (; t < f + 1; t++) { } 
    for (; f > t; f--) { } 
}

while (cin >> gettine(A, &size) && A[0] != 0) { } } 
const int size = 100; int nch = A[0]; char A[size]; 

main()

using namespace std;

#include<iostream>

#include<iostream>

using namespace std;

// Selection sort demo: processes characters within the array A.
Save a copy to help you begin future projects.

WHEN you type more than 11 characters before pressing „enter“. Everybody ASAP: Write a program like this, and see WHAT HAPPENS.

textbook. To get more details right now, read pages 183-187 of Main and Satch’s This program manipulates chars as if they were numbers.
of the variable its VALUE=STATE can be changed.
automobile, shebox, has a unique identity even after it is "cloned". Features

The NAME can be copied, the VARIABLE=OBJECT itself, like a person,

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 has (some kind of) name.

The main points...
We will use yellow to denote what has changed.

Brandon = 98

Adam

Brandon

Charlie

Brandon

Charlie

Brandon

Adam
is it the value of a pointer variable, that is, an address.

So, when you or others say "pointer", think hard: It is a pointer variable, or

But most everyone, we and DSO, say, for short, "PIVAR is a pointer."

Illegal value.

C/C++ int variable, or else it might have the NULL value, or else some

C/C++ type variable. The variable named PIVAR might store an address of a
pointer type variable. The variable named PIVAR is really (the name of)
we and DSO said "a pointer is an address". PIVAR is really (the name of)

int *PIVAR; What is PIVAR? Is it a "pointer"?

int PIVAR; What is PIVAR? Is it a "pointer"?

int *ivar; What is ivar? It's an integer.

Most say "It is an integer!" but, really, ivar is (the name of) an variable

A linguistic pitfall—try not to fall into it!
Perhaps we should always use the word "address" for "pointer value".

done.

Type which determines what values it can hold and what operations can be used with it. Each variable has a type, and "pointers in C/C++ types. Each variable has a type which determines what values it can hold and what operations can be used with it. Each variable has a type, and "pointers in C/C++