Climb at Linked Lists for Project 2.

Java comparisons and tradeoffs.

Dangers of deletion.

Review of scopes as used in the static bag.

Dynamically partially filled array (Ch. 4 of DSO)

Abstract data types, Bag, examples and implementation alternatives.

Array and dereferencing notation

Using pointers—3 steps

CSI 310: Lecture 7
Step 3: Dereference the pointer variable:

```cpp
*pointer = new Throttle(5);
```

Way 2 (Address of a dynamically allocated object (Lik in Java):

```cpp
pointer = & ThrottleTest;
```

Step 2: Obtain an address; store it in the pointer variable:

```cpp
ThrottleTest * pointer;
```

Step 1: Declare a pointer variable:

```cpp
ThrottleTest * pointer;
```
Library container classes for sequences.

Java DOESN'T do this. Java arrays are "first class" like 
++ Standard

Java arrays are:

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
4 & 5 & 6 & 7
\end{array}
\]

 pointers.

\[
p1 = \ast \text{the address of some integer array;}
\]

\[
\text{int } * \text{p1;}
\]

If p1 is a int pointer variable whose value is the address of entity 0 of an int array, then p1[0] accesses entity 0, p1[1] accesses entity 1, etc.

Deferencing a pointer p1 using array notation:

\[
p1 \rightarrow 7
\]
So \( \text{PART} = \text{NULL}; \text{PART}_- \text{shifit}(1); \) \text{CRASHES the process.}

For function member of the located object:

\[-\text{"hyphen less than" DEREFERENCE the pointer and SELECTS the data}\]

\[\text{PART}_- \text{shifit}(1); \] \text{is EQUIVALENT TO} \text{PART* \text{shifit}(1);} \]

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

\[\text{PART}_- \text{shifit}(1); \] \text{is EQUIVALENT TO} \text{PART[0].shifit(1);} \]

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

\[\text{count} \gg \text{PART}[3].\text{flow}() \gg \text{end};\]

\[\text{PART}[1].\text{shifit}(2);\]

\[\text{PART}[0].\text{shifit}(1);\]

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

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

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

University at Albany Computer Science Dept.
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.

1. Fill array: The details were easily adapted from DS0 Chapter 3.

2. Project 1: You implemented each feature using a fixed capacity (of 10) partially filled array, of chapters 3 and 4 of DS0.

3. The first data structure/algorithm used to implement certain container classes, clever, efficient data structure/algorithm implementation examples, will concentrate on container classes both as abstract data types and for the output and output are examples of concrete class, very helpful. CS1310
1. Data array

one data structure the

3. Its current length: How much of the "Capacity" is currently "Used".

2. Its maximum usable length "Capacity".

The classes based on partially-filled arrays of DSO chapter 3 (and 4) overcome

the vulnerabilities of NULT char terminated "C-string" arrays by combining in
midterm.

STL in 3.0 projects is forbidden until further notice. Around the
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. Naive 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
subscripts are not used. But

Java and Perl arrays resize themselves automatically when large enough
\((n) W\) is a function with domain and range \(W\).

Mathematical formalizations: Set \(S\) is finite means the sum of the \(W\) is finite.

\[
\begin{align*}
0 & \quad 1 \quad 2 \quad 3 \\
\text{No} & \quad \text{Yes}
\end{align*}
\]

Has a well-defined answer:

\(\text{Is } x \text{ in } S?\)

\(0, 1, 2, \ldots \) \(\{0, 1\} \leftarrow W \colon W\)

Given an item, \(x\),

Multiset \(W\) (another name for "bag"
<table>
<thead>
<tr>
<th>Item number</th>
<th>Name</th>
<th>No. orders</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Grilled cheese sandwich</td>
<td>99</td>
</tr>
<tr>
<td>3</td>
<td>Pot of 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 serves!
in ch. 3.

Also store the current position.

(Plus variable current-index to

SAME as for bag;

Partially filled array

"the order matters"

(Sequence

The invariant p.103-4.

(Study ch. 3 details; especially

used/first empty position.

Array of items; plus variable

Partially filled array

Implementation/Structure

Abstract/Type Data Structure

{ ... insert() advance current start

operator+() insert erase one() erase

} } "multi"
Implementations (They often vary in efficiency)
The same abstract data type can have very different alternatives

of bag

for various variants
an ALTERNATIVE
The Linked-List (to be taught) is
Partially-Linked Array.

each array element used.
With the receive stored in
Partially-Linked Array:

exercises
see ch. 3

Abstract Data Structure
Implementations/Implementation Combinations

More abstract data type/implementations
When it fills up, the insertion algorithm runs new to allocate an
initialized. The implemented data structure is a dynamically allocated partially
structure is a fixed size partially filled array of items.

Roadmap:

Chapter 4: Improves this bag class, so the number of items is practically
used have local meanings.

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

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

private:

NOTE: No other public data members!

static const size-type CAPACITY = 30;

typedef std::size_t size-type;

typedef int value-type;

public:

} }

class bag

...
static const size_type CAPACITY = 30;

NOTE: No other public data members!

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

class bag
{
    static const size_type CAPACITY = 30;

    static data member qualifier means "class variable", just ONE, named bag::CAPACITY,
    cannot be changed (directly).

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

    private:
    ...
class bag {
public:
  ... static const size_type CAPACITY = 30; ...
  NOTE: No other public data members!

  // some (public) member functions

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

  // Some (private) member functions

  // ... No other private data members!

  static const size_type CAPACITY = 30;

  public:
    ...}

class bag
Precondition for insert():

- size() + 2 \leq \text{CAPACITY}

Precondition for operator+(Bag b1, Bag b2):

- b1.size() + b2.size() > \text{CAPACITY}

We don't care about the rest of data[].

1. The number of items in the bag is the value of used.
2. For an empty bag, used = 0.
3. For a non-empty bag, the items are stored in data[0] through data[used-1].

Invariant for the Primitive Bag Class:

```java
private:
    private:
        public:
    class Bag

FILE: bag1.h
University at Albany Computer Science Dept.
```
...multitudes of Java libraries and language classes are dynamic!

 réalité (new and delete) executed when the program runs.

 coded operation (new and delete) executed or destroyed, or recycled, when it returns.

 called and destroyed (stored) recycled when it returns.

 3. Local external (C/C++ automatic) variables are created when a function is

 2. Dynamic variables are created during process execution, unpredictable.

 input data can determine if and how many dynamic variables are created.

 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
The improved bag uses a dynamically allocated array:

```java
public class MyBag2 {
    private int capacity; // size - type: capacity
    private int used; // size - type: used
    private int data[]; // value - type: data

    private MyBag1 bag1; // private: private const size - type: CAPACITY = 30;
    private MyBag2 bag2; // public: public;

    public MyBag2() {
        // constructor
    }
}
```
The (I) part is retained.

new value-type [brigger] 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.

Each bag consists of

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

Main and Savage, Figure 4.2

University at Albany Computer Science Dept.
<table>
<thead>
<tr>
<th>C++</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runs, Java when the garbage collector garbage collection, as they do in garbage collection are impossible.</td>
<td>Programmers don't worry about garbage pointer failures because garbage collection is automatic.</td>
</tr>
<tr>
<td>do not pause unpredictably for interactions in C++</td>
<td>C++ does not, but it gives us the delete operator.</td>
</tr>
<tr>
<td>Java has garbage collection: It automatically searches for inaccessible dynamic objects and recycles their space.</td>
<td>A Java vs. C++ language engineering tradeoff:</td>
</tr>
</tbody>
</table>
The computer can remove the top item and then re-heapsify the data structure.

You get closer to the root. The biggest data item is at the top. If a tree that holds data in an arrangement so that the values get bigger as few weeks.

Unfortunatly, the word "heap" has a very different meaning we will cover in a

Unfortunately, the word "heap" has a very different meaning we will cover in a

Textbooks (DSO, etc.) call it the heap.

Structurp's.

allocation is called the free store in professional C++ books (like

The memory or storeage "resource" (apartment building) for dynamic

A bit of terminology...
call, and C/C++ internals like \texttt{386, 0x3BD, 0x13 (which equals eleven)}, and

**Initializer in a declaration**: input from the user, return value from a function

**Ask yourself**: How do you program in C++ to "get" an integer value?

alternative array [] syntax.

- Declaring pointer variables or values: in an **EXPRESSION**, * Dereferencing pointer variables or values: in an **EXPRESSION**, *

- Declaring pointer variables: in a **DECLARATION**.

- Obtaining pointer (i.e., address) values: \texttt{new, new},

\texttt{new}$^*$ \texttt{array} fundamental:

\texttt{vector}() or other function that returns a pointer value.

\texttt{new} \texttt{array} fundamental:
Like this is a Core Learning Objective.

Understanding Data Structure Diagrams

Nothing happens... until user types 17 center:

\[ \text{cin} \rightarrow \text{PMyInteger}! \]

1. \text{PMyInteger = new int;  //Useless, time and space wasting operation.}
2. \text{garbage variable is created an anonymous integer}
3. \text{PMyInteger = new int;}
4. \text{garbage int \* PMyInteger;  //Useless.}

University at Albany Computer Science Dept.
node * tail-\texttt{ptr};
node * head-\texttt{ptr};
\ldots
\}

\text{\texttt{private:}}
value-\texttt{type} \texttt{data-field};
\ldots
\text{\texttt{public:}}
typed \text{\texttt{double value-type}};
\text{\texttt{constructor,}} \text{\texttt{manipulator}};
\ldots
\text{\texttt{public:}}\

\begin{itemize}
\item[\texttt{class node { \texttt{public:}}]
\end{itemize}

\begin{itemize}
\item[\texttt{class node {{\textit{STYLe}}\hfill}]
\end{itemize}

\begin{itemize}
\item[\texttt{OOP/Modern C++ STYLe}]
\end{itemize}

\begin{itemize}
\item[\texttt{Classic C STYLe}]
\end{itemize}

\text{\texttt{Known LOCATIONS}}

\text{\texttt{Linked List:}}\hspace{1em} \text{Concrete linked data structure} \text{\texttt{good for implementing}}

\text{\texttt{One-dimensional sequences, so entity insertion and deletion near entries at}}

\text{\texttt{known locations, so}}