Partially Filled Array Implementation

Container classes

Public and Private (brief)

Using a class.

Why were classes invented?

Concrete classes, container classes

Data and function members

Classes/Structs, primitive and compound types

CSI 310: Lecture 4
Compound types: Arrays, and struct/classes

AND Pointers (called References in Java)

Primitive types: int, short, long, char, float, double, bool

Primitives.

An array can hold struct or class type variables as well as

how to call methods.

WITH one variable of THAT class or struct type. (Explore in lab

functions (synonym: methods) that usually operate ON or

in C++/Java, a class or struct ALSO can contain member

different types. (Such combinations are sometimes called records.

Unlike an array, a class or struct type variable combines variables of
named top-position and position.

Each throttle variable is composed of two subvariables,

```c
    {
        int position;
        int top-position;
    ...
    }
    //
```

class throttle

The data members of a throttle:
The function members of methods of a throttle:

```cpp
class throttle {
    public:
        // The body of throttle is in throttle.cxx
        void shift(int amount) {
            int position = 0;
        }
        void shift(String position = top) {
            int position = top;
        }
        bool is-on() const {
            return position > 0;
        }
    private:
        const double flow = 10;
        const double top = position;
}
```
structure/algorithms implementation examples.
abstract data types and for clever, efficient data
helpful. CS1310 will concentrate on container classes both as
the crate and point are examples of concrete classes, very
ALWAYS INITIALIZED PROPERLY.
So, your class design needs can guarantee your class variables are
whenever a class/struct variable is ALLOCATED.
The computer automatically calls a constructor you have written
structure/algorithms implementation examples.
in ch. 3,
store the current position; also plus variable current-index to
SAVE as for base;

Partially filled array:

```
{ ... }
insert() ;
advance() ;
current() ;
start() ;
```

```
the invariant p. 103-4.
{ ... }
```

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

```
item array, fixed size; plus var-
```

```
Abstract Data Type
Implementation Data Structure
```

```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
value of top-position.

avoid making the value of position become negative, or to exceed the
positive non-zero integer. Finally, to shift the throttle, make code that
like position = 0. Also, remember to assign top-position to some

Dear Programmer,

probably won't bother. (Something like this:

They, you might write comments or other documentation (but you

int position;
int top-position;
C/C++ code you write is

An old-fashioned style for implementing a “throttle”:

The ONLY
Disadvantages: position and top-position are global variables:

```c
{
    position = 0;
    (0 > position
    position = top-position;
    (position < top-position
    position = position + x);
}

void shuffle(int x);
{
    position = 0;
    position = x;
}

void initialize(int x);
```

Variables by calling functions like these:

and warn everybody that they should only access the two throttle

A small improvement is to code "throttle maintenance functions"
multiple threads.

It is very clumsy if you want to upgrade your software to have
functions used for different program objects.
The names of the functions will clash with initialization and other
be negative

that puts the throttle in an illegal state. The position should never

Notice nothing prevents an errant programmer from writing code

throttle: throttle; position = -38;

written like:

code that defines a separate throttle and then accesses it can be

};

int top;

struct throttle

In a header file, declare the throttle type as a structure:

Here’s a way to overcome the 3rd disadvantage:
In the throttle class, declare the throttle type:

Here's the object oriented way:
inline { position = 0; }
    top = 1;
}

throttle::throttle()
{
    position = 0;
    (position > 0)
        position = top;
    (top < position)
        position = position + x;
}

void throttle::shift(int x)
{ /* Things */

   // the issues
   // with formally specifying
   // what you'd like to do
   // with private member functions
   // and why you'd like to do it
   // is the implementation
   // of throttle::throttle,
   // carefully code private member functions to do the
   // right things
   // in the throttle::xx implementation.
{ return top < 0; }

bool throttle::is-on() const
{
    return position / double(top);
}

double throttle::flow() const

Implementation of data extraction functions:
class
outside the body of a function member belonging to the throttle

REUSE to compile a private member access like mytr.top = 9;
The private member protection rules of C++ make the compiler

....

mytr.shit(1); // Now mytr is ON.
because the default constructor was called.

// mytr is a property initialized throttle

throttle mytr;

....

//inside some functions

}....

#include "throttle.h"

Here's how you USE a throttle:
class point from Chapter 2 an an example.

Part of your project 1 job: Implement the concrete class Cotton; use

that the programmer coded the throttle class he designed.

C++ and Java ENFORCE the rules for working with a throttle

(similar features).

all functions to access and manipulate it) in one place. (Java has

encapsulate everything (variables to store the throttle’s state plus

Object Oriented Programming features of C++ to

What we surveyed, and MOS details in Chap. 2, is how to use
Technology for Speed

• Clever data structures and algorithms

• Abstract Data Types

CS2 "Data Structures" Subject:

• Using arrays

They and students must know internals first

C++ professionals use Standard Template Library

Directly useful in application programs

• Intro. to container classes

Chap. 3 of DS0
... 0 1 2 3

How many times does x appear in M?

Given an item x, "Multiplet M" (another name for "bag")

Yes

No

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>Description</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Grilled cheese sandwich</td>
<td>99</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Pot of hot &amp; sour soup</td>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Egg Roll</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Szechuan chicken</td>
<td>72</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

No. orders: What a Chinese restaurant waiter serves:
private:

    int count(const Dish target) const;
    { int size() const { return used; };
    void operator + (const Bag& disgard address);
    void insert(const Dish entry);
    bool erase-one(const Dish target);'
    int erase(const Dish target);'
    { bag() used = 0; };
    static const int CAPACITY = 30;

public:

    class BagOfDish

#include "Dish.h"

#define BagOfDish

#undef BagOfDish

Simulated Main/SaveBar of Dishe:
```java
#include

Dish data[CAPACITY];
```

```cpp
{
    ++used;
    data[used] = entry;

    assert (size( ) > CAPACITY);
}

void bag0fDishes::insert (const Dishes entry)
const int bag0fDishes::CAPACITY = 30;
using namespace std;

#include <assert>

#include "bag0fDishes.h"

(2 parts of the IMPLEMENTATION file bag0fDishes.cxx)
```
{ }

return answer;

answer++;

if (target == data[i])
{
for (i = 0; i < used; ++i)
    answer = 0;

int i;
int answer;

int bagOfDishes::count (const Dish target) const

```c
{ return answer;
    if (target in data[0].used-1) // INV: answer==number of appearances
        answer++;
    if (target == data[t]) [ ]
        if (target in data[0].(t-1)) // INV: answer==number of appearances
            answer++;
    for (t = 0; t < used; t++)
        int answer; int t; answer = 0;
}

int bagofdishes::count (const Dish target) const

Making sure it's right with INVARIANTS
```
Implement a container class for Coins using a partially filled array.

Implement a CoIn class.

**Parts of Project 1:**

**INVARIANTS/PREREQUISITES/POST-CONDITIONS.**

algorithms (like search, insertion, removal), and

Pay close attention to discussions of algorithms, coding of

Skip over value-type, size-type and namespace stuff, but

Your job: Study Chapter 3
{ 
    // COPIES one Dish value
    data[used] = entity; // assignment statement
    ...
    // }
    void bagOfDish::insert( const Dish entity )
    { // const means insert will NOT CHANGE variable YumYum
        (2) const means insert when order. insert( YumYum ) is called.
        (1) Reference parameter ( ) means the value of YumYum will NOT
        be copied when order. insert( YumYum ) is called.
    }
    void bagOfDish::insert( const Dish entity )
    { // Misc. Techniques
        ...
        order. insert( YumYum );
        ...
        bagOfDish order;
        Dish YumYum;
        ... }
The same abstract data type can have very different alternative implementations. They often vary in efficiency.

<table>
<thead>
<tr>
<th>bag</th>
<th>for various variants</th>
</tr>
</thead>
<tbody>
<tr>
<td>an ALTERNATIVE Imple.</td>
<td>The Linked-list (to be taught) is partially filled array.</td>
</tr>
</tbody>
</table>

Each array element used. With the receipt stored in with partially filled array:

```c
// exercises
// see ch. 3
...
```

**Implementation Structure**

**Abstract Data Type**

**More abstract data type/implementation combinations.**