Reference parameters, Constant parameters.

Invariant examples.

ADTS building and testing practices.

Filled arrays, arrays of objects, CS1310 software structuring.

PROJECT 1: One concrete class and one container class, partially

CSI 310: Lecture 5
Example of SW engineering principle: Separation of Concerns.

storeage of data (state) are implemented.

operations that do the work, AND NOT HOW the operation and

It emphasises WHAT work is done and how to INVOKED the

implemented.

operations available. The definition OMITS how the type is

A data type defined only in terms of names and properties of

What is an abstract data type?
in ch. 3.

store the current position. (also
plus variable current-index to
same as for base

Partially filled array:

\textit{The invariant} (p. 109-10):

(Study ch. 3 details; especially
used=first empty position.
able
item array, fixed size; plus vari-

Partially filled array:

Implementation Data Structure

Abstract Data Type

\textit{the order matters}

...;

\texttt{insert}();

\texttt{advance}();

\texttt{current}();

\texttt{start}();

\texttt{operator-+}();

\texttt{insert}();

\texttt{erase-one}();

\texttt{erase}();

\texttt{multi-}
The same abstract data type can have very different implementation:

| Implementation | Alternative implementation
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Linked-list (to be taught) is partially filled array.</td>
<td>The linked-list variant is an alternative implementation for various variants</td>
</tr>
<tr>
<td>Each array element used.</td>
<td>The $\texttt{Linked-list}$ variant is an alternative implementation for various variants</td>
</tr>
<tr>
<td>With the element stored in which</td>
<td>Pseudo-Code:</td>
</tr>
</tbody>
</table>
| Partially filled array: | `data-structure = \{ `}
| Implement alternative data structures. | `keyed-bag` |
| More abstract data type/implementation combinations. | `\texttt{Link}\ldots` |
private:

    int count (const DishTarget & target) const {
        int size = 0;
        return used;
    }

    void operator += (const BagOfDishes & added) {
        void insert (const Dish entity) {
            void erase one (const DishTarget & target) {
                int erase (const DishTarget & target) {
                    bool empty = 0;
                    bag = 0;
                    return used = 0;
                } int CAPACITY = 30;
            }
        }
    }

public:

    class BagOfDishes {
        #include "Dish.h"
        #include BagOfDishes.h
        #include BagOfDishes-heap.h
    }

    Simplified Main/Save the Bag of Dishes.
#endIt
}

INV: How much of array is used

INV: Array to store the dishes

Dish data [CAPACITY];
{
    +used;
    data[used] = entry;

    assert (size() > CAPACITY);
}

void bagOfDishes::insert(const Dish entry)

using namespace std;

#include <assert>

#include "bagOfDishes.h"

2 parts of the IMPLEMENTATION (file bagOfDishes.cpp)
{ return answer; 
answer++; 
if (target == data[i] ] 
{ for (i = 0; i < used; ++i) 
answer = 0; 
int i;
int answer;
}
int bagOfDish: count (cost Dish target ) cost

```
return answer;

for i in range(0, used-1):
    if target in data[0..(used-i)]
        answer = number of appearances
        answer++;

for i in range(0, used-1)
    if target == data[i]
    if target in data[0..(i-1)]
        answer = number of appearances
        answer++;

for i in range(0, used-1)
    if target in data[0..(i-1)]
        answer = number of appearances
        answer++;
```

Make sure it's right with INVARIANTS

int bags徙動().\count (\const Dijkstra target) \const
Implement a container class for coins using a partially filled array.

Parts of Project 1:

INVIARLANTS/PR/E/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 = entry; // assignment statement
  . . . // } 

  
  void bagOfDish::insert ( const Dish entry )
  { 
    data_used [ index ];
  } 

  // (2) const means insert will NOT CHANGE variable
  // be copied when order. insert ( yumyum ) is called.
  // (1) Reference parameter ( x ) means the value of yumyum will NOT
  // be copied when order. insert ( const Dish entry )
  
  void bagOfDish::insert ( const order )
  { . . . ; yumyum ; }

  . . .

  bagOfDish order;

  . . .

  Misc Techniques:
is called. The value of M doesn't change.

\( n \in \text{ a value parameter: \texttt{A SEPARATE (temporary) variable}} \)

\[
\begin{cases}
\text{return } & \text{\texttt{\( +n \)}} \text{ if } \texttt{\( n \)} \neq 0 \\
\text{int kick(int n) } & \text{\texttt{\( +n \)}} \\
\end{cases}
\]

printf 39 38

\[
\begin{cases}
\text{cout } & \text{\texttt{\( \gg M \)}} \text{ if } \texttt{\( M \)} \neq 0 \\
\text{cout } & \text{\texttt{\( \gg \texttt{kick(M)} \)}} \\
\text{int M; } & \text{\texttt{M = 38}} \\
\end{cases}
\]
Now n is a reference parameter. The n in kick refers to

```cpp
    {
        return ++n ?
    }

    int kick ( int n )

    n is a reference parameter (note the & in

    prints 39 39

    {
        cout >> M;
        cout >> kick ( M );
        cout >> kick ( M );
        cout >> kick ( M );
        int M; M = 38;
    }
```

the argument variable M. So, ++n increments M.
(primitive) user interface.

Part 2: Develop the teller application program, which includes a
test driver.

Part 1: Develop the Pile class, including TESTING thereof with a

Project 1 discussion:
makes a mistake.
So, the teller application (ideally) should never crash when the user
requests a withdrawal.

Preconditions:
Only application BUGS (programming errors) cause violation of
informative messages and allow the user to recover and continue.
If the user makes a mistake, the application should give an
error message.

---

If the preconditions are violated, the program must fail the test to try
violating the preconditions.

assert();

assert();