Clever data structures/algorithmic technology for speed.

IMPLEMENTATIONS

Abstract Data Types Separate USE, INTERFACE from C S2 “Data Structures” subject:

Using arrays:

They and students must know internals first. They and professionals use Standard Template Library. Directly useful in application programs.

Intro to container classes:

1. DS O CH. 2 on concrete classes. Fast.
2. Sec. 4.1 of DS O: POINTERS
   - Lab I Assignment on Pointers is Here. BASIC IDEA.
3. CHAP. 3 of DS O, Reference for Project 1.

CSI 310: Lecture 4
changed.

"clone", features of the variable, its VALUE=STATE can be a person, automobile, shoebox, has a unique identity even after it is like. The NAME can be copied. The VARIABLE=OBJECT itself, like

**VARIABLE**

<table>
<thead>
<tr>
<th>4. (This is new)</th>
<th>a NAME can be the VALUE of a</th>
</tr>
</thead>
</table>

**DIFFERENT conceptual entities (3 different things)**

| 3. The value, the name, and the variable itself are three |

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

| 1. Each variable has (some kind of) name. |

The main points...
Brandon = 98
Well,
a little dangerous, requires a good knowledge of data structures to use
Perl is like a Swiss Army Knife...very useful, a bit clumsy and inefficient,

(at this point, I typed control-d, and the computer answered.)

print $brandon . "\n"
$adam = 98;
$adam = "brandon";
perl ~CSI310/lect/104 $ perl -t

variable...but not C/C++/Java.
store a STRING value in one variable and use it to NAME another
Some programming languages, such as Perl or Lisp allow you to
Worry about this yet...

sizes of variables are composed of different numbers of bytes. Don't

One big house might cover several adjacent lots. Technically, different

DIFFERENT FROM the variables' current VALUE.

addresses. The address of a memory location or byte is very

locations (called 'bytes'). The byte numbers are called memory

A segment of computer memory consists of numbered memory

DIFFERENT FROM the house's current OCCUPANTS.

numbers are called 'addresses. The address of a house is very

A block of real estate consists of numbered houses. The house

runtime.

the kind of name that C/C++ code can copy, use, store, etc, at

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

Pointer value or points to a 4-byte integer variable.

Address 992
Pointer values, like any other numbers, can be stored in variables, like integers. Here's a pointer variable whose address is 987 and value is 57.

The pointer variable whose address is 992 is pointing to the above integer variable whose address is 992 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.

The above integer variable is pointing to the above integer variable whose address is 992 and value is 57.
NUMERICAL POINTERS VALUES, and call them “references”.
Unlike C/C++, Java and Perl (except debuggers) hide all hardware systems. Sometimes within I/O devices.
The memory bus addresses usually locate data within cached RAM
Bus or physical addresses.
Hardware memory management units into hardware memory
Numerical (binary) virtual addresses are quickly translated by
the hardware-software interface (CS1333).
Numerical pointer values are virtual addresses, and part of
systems:
PC/Workstation/Servers with UNIX-like/WINDOWS NT
operating
a few background words... In late 20th century technology of
$\$

The VALUE of X is 98.6.
The ADDRESS of X is 0x8ffffff4

$ a.out $ 

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

{ 

cout << "The VALUE of X is " << x >> x.endl;
cout << "The ADDRESS of X is " << &x >> x.endl;
float x = 98.6;
}

main()

using namespace std;

#include <iostream>

The C/C++ % operator provides the ADDRESS OF its operand, which must be a variable.
And now for some concrete class motivation...
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
with the throttle, make sure you initialized it by executing a statement
Kingly, please be nice and follow these tricky rules. Be & do anything

Dear Programmer,

something like this:

The boss might ask for documentation (but you probably won’t

int position;
int top-position;

ONLY C/C++ code you write is
control for the speed of a simulated car/plane/rocket/etc. The
An old-fashioned style for implementing a “throttle” (user interface
```java
{
    0 = y
    (0 > y) &&
    y = top-position
    (y < top-position)
    x + y = position } =
    void shut(int x)
    {
        0 = position
        x = position
        } =
    void initialize(x)

    variables by calling functions like these:
and warn everybody that they should only access the two throttle
and warn everybody that they should only access the two throttle
A small improvement is to code "throttle maintenance functions"
variable, compared to that of B [366]:

Remember the 366 birthday count

• multiple throttles. (Remember the 366 birthday count

3. It is very clumsy if you want to upgrade your software to have

other functions used for different program objects.

2. The names of the functions will clash with initialization and

protect them from external access.

1. Position and top-postition are global variables, nothing

Disadvantages:
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; throttle.position = -38;
written like:
Code that defines a separate throttle and then accesses it can be
;
}
int top;
} int position;
struct throttle
In a header file, declare the throttle type as a structure:
Here's a way to overcome the 3rd disadvantage:
class Throttle

In the throttle, in header file, declare the throttle type:

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

throttle: throttle()
{

    position = 0;
    (position > 0) ? position = top :
        (top < position) ? position = position + x :

    (void throttle::shutter(tnt x)

right things:

In the throttle::cxx implementation file, you can implement
class
outside the body of a function member belonging to the throttle
REUSE
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"

//Separate C++ Title race-car.cxx

Code to defines and use a throttle:
that the programmer coded the throttle class had designed. C++ and Java enforce the rules for working with a throttle (similar features) and has 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 DSO details in Chap. 2, is how to use
Going back to pointers, again...
The C/C++ compiler uses this formula for you. You won’t need it.

Formula: \[ a[I] = \text{base} + \text{size} \times I \]

So, given (1) the address of the \( a[I] \).

We can access to the \( I \)th entry with \( a[I] \).

It is a variable, comprised of a sequence of entity variables of

What is an array? [0] [1] [2] [3] [4]

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
$-37.6 \ 15.3\ 98.6$

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

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

`$ a.out

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

{
  // The VALUES of x[0..2] are ""
  count >> "The ADDRESSES of x[0..2] are ""
}

float x[3] = {-37.6, 15.3, 98.6};

} main()

An array element is a variable, so & produces its address:
try {  
    cout "\n    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.

```c
main()
{
    float *my1stPointer;
    my1stPointer = &X;
    cout << "my1stPointer equals \&X;" << endl;
    cout << X << endl;
    cout << "\&X << endl;";
}
```

$ a.out
my1stPointer equals 0xbffff9b0
But X equals 98.6

0xbffff9b0

•–
— ˜™
š
œ
•
ž
y

x
s
u
x
w
y

us'
z
¡
The type of variable it can hold the address of (the type of data 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...)

(whch, by coincidence, is also the C/C++)

an asterisk

variable

declared pointer

The name of the newly declared pointer

float * MyFirstPointer;

Learn how pointer variables are declared.
MyP does NOT point to 2003!

Now we can say "MyP POINTS TO MyInt".

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

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

pointer variable MyP POINTS TO int variable MyInt

computer's memory: produces in the

```
MyP
```

```
2003
```
The * operator \textbf{dereferences} a pointer (or address value).
That means: Access the variable whose address is in the pointer (or is the address value).

\begin{center}
\begin{tikzpicture}
\node[draw] (MyP) at (0,0) {MyP};
\node[draw] (MyInt) at (1,0) {MyInt};
\draw[->, thick] (MyP) -- (MyInt);
\node[anchor=west] at (MyP) {0xbe0fef04};
\node[anchor=west] at (MyInt) {2003};
\end{tikzpicture}
\end{center}

\begin{verbatim}
cout << MyInt << endl;
cout << *MyP << endl;
cout << MyP << endl;
\end{verbatim}
0xbe0fef04
MyP MyInt 2003

After Multiplication

4006

count >> MyP

4006

count >> MyP

4006

count >> MyInt end;

Myp = *Myp * 2;

Before Multiplication

MyP MyInt

0xbe0fef04

MyP MyInt

0xbe0fef04

MyP MyInt

0xbe0fef04
This is a "CRASH": Computer tried to read memory at the illegal address 0x0, so it could print the int value located there.

Segmentation Fault

cout >> MyP
        0x0 0xbe0fef04
        cout >> MyP
        4006 cout >> MyP
        cout >> MyP

After Assignment:

Before Assignment:
pi[1] = pi[0] + 2;

cout "pi[0] > end!", prints 7;

// pi[0] is the address of some integer array

int *pi;

\[ \text{\texttt{print}} \]

If pi[0] is a int pointer variable whose value is the address of entity 0

 Dereferencing a pointer pi using array notation:
cout << PAT[3][FLOW()] >> endl;

PAT[1][SHIT] (2);

PAT[0][SHIT] (1);

"PAT = the address of some array of throttles;"

throttle *PAT;

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

...
Depending on how you tell where the end is.

... holds strings up to 99 or 100 chars long.

char mystring[100];

Of char, declared:

One way to store/process strings data is to use a C/C++ array

This is called string data.

... page, what you see "RIGHT HERE": R, I, C, H, T, etc.

contents of a word processed text file, text on a Web

(Somethings NEW?) Non-numerical "text" data, such as the
track of the end.
different times, they always begin at position 0, so you have to keep
It's easy: The SAME array holds different length sequences at
adapted from DS0 Chapter 3.
fixed capacity (of 52) partially filled array. The details are easily
Project 1: You must implement the deck and each P1.P using a
4 of DS0.
container classes is the Partially Filled Array, of chapters 3 and
The first data structure/algorithim used to implement certain
structure/algorithim implementation examples.
abstract data types and for clever, efficient data
helpful. CS1310 will concentrate on container classes both as
theortical and point are examples of concrete classes; very
then W, o, x, T, and finally d.

prints the characters in order, H, then e, two I's, an o, a space,
count >> "Hello World"

is easier to think about than
prints the string Hello World
count >> "Hello World"

It is useful to think that an array of char
variables is a single variable that holds ONE C-string.
C-strings are different from C++ strings you get from #include <string>

In C/C++, the char "A" is coded \x01, not A (char W[]) with:

The null char is coded \0, the char "A" is coded \x01, the null char

In C/C++, unprintable value called the "null char".

One way to tell where the end of a string is: Just after the last
cout << "MyChar$text >> endt;
Printing what you typed:

#include "hello_world"

Reading up to 11 characters you type on one input line:

holds a C-string with length up to 11
char MyChar$text[12];
C-string:

REQUIRED in CSI10: Declaring a variable that can hold a

like: cout << "Hello World"
C-strings are very easy to use. You have used them in CSI20 code

using namespace std;
#include <iostream>
(i) their design choices were rational.

was valuable.

But they did think that every last microsecond of computer time

imagine that some nasty people will make that happen on purpose.

The earliest C/UNIX/Internet/DOS/Windows designers did not

so-called BUFFER OVERFLOW!

when you copy a C-string without counting the characters, so that

VIRUS AND OTHER SOFTWARE EXPLOITATION

VULNERABILITIES THAT ENABLES PEOPLE TO WRITE

C-strings are probably the leading

gee, that's wonderful. Wow!
USED

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

2. Its maximum usable length "Capacity"

1. Data array.

I overcome this vulnerability by combining in one data structure the

The partially filled array type classes of DSO chapter 3 (and 4)
the midterm.

STL in 310 projects is forbidden until further notice...around after

understanding has higher priority for us.

too...this course and DS0 introduce how to use it; but scientific

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

on large data sets.

2. Naive programmers will encounter enormous run time penalties

and computer scientists must understand.

structures/algorithms that truly knowledgeable professionals

I. Their runtime environment software uses data

enough subscripts are first used. But...

Java and Perl arrays resize themselves automatically when large
... 0 1 2 3

`How many times does x appear in W?`

Given an item `x`,

`Multiset 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>No. orders</th>
<th>Item number</th>
<th>Name</th>
<th>Item</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>99</td>
<td>Grilled cheese sandwich</td>
<td></td>
<td>6.99</td>
</tr>
<tr>
<td>3</td>
<td>27</td>
<td>Pot of hot &amp; sour soup</td>
<td></td>
<td>2.79</td>
</tr>
<tr>
<td>8</td>
<td>19</td>
<td>Egg roll</td>
<td></td>
<td>1.99</td>
</tr>
<tr>
<td>5</td>
<td>72</td>
<td>Szechuan chicken</td>
<td></td>
<td>7.29</td>
</tr>
</tbody>
</table>

Good example: What a Chinese restaurant serves.

Bad example: Would like some chicken please.

University at Albany Computer Science Dept.
The same abstract data type can have very different implementations. For various variants of each array element used, the implementation for various variants of the linked-list (to be taught) is:

\[
\text{Partially Linked array: }
\]

\[
\text{Partially Linked array: }
\]

\[
\text{Partially Filled array: }
\]

Abstract Data Type

More abstract data type/implementation combinations:
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
DSO chapter 4 improves this bag class, so the number of items is

dynamic.

The implementation data structure is a fixed size partially filled
array of items.

The implementation data structure is a fixed size partially filled array of items (counting multiplicitys).

DSO chapter 3 teaches a primitive bag class, for which each bag

Roadmap:
Currently in each of the 3 variables, before and after each step

Please simulate the execution of this by writing the values

\[
p2 = \text{TEMP} ;
\]

\[
p1 = *p2 ;
\]

\[
\text{TEMP} = *p1 ;
\]

\[
\text{int TEMP} ;
\]

\[
p2 = \text{82} ;
\]

\[
p1 = \text{81} ;
\]

\[
\text{int } p1 ; \text{ int } p2 ;
\]

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

\[
I = 1928 ; J = 2003 ;
\]

\[
\text{int } I ; \text{ int } J ;
\]
assert (p1 == p2) && (p2 == p1);
1928 2003
count >> I >> J end;
count >> *p1 >> *p2 end;
p2 = TEMP;
p1 = p2;
TEMP = p1;
int *TEMP;
p1 = 81;
p2 = 81;
int *p1;
int *p2;
int I; int J; I = 1928; J = 2003;

different?
Now, how is swapping the values of the pointer variables
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/C++ automatic) variables are created when a
3. Local extant (C/C++ automatic) variables are created.
Dynamic variables are created.
Unpredictable input data can determine it and how many
2. Dynamic variables are created during process execution.
identifiers. But they, like all variables, have memory addresses.
1. Dynamic variables are not declared. They are not named by
(you need pointers to access them!)
Dynamically Allocated Variables
new is a keyword in C++. See DS0 FIGURE 4.1

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

    PI = new int[4];
    array form for new
    To allocate an array of say 4 ints, do the same except use the

    PI = new int;
    and then make program execute:

    e.g. int *PI;

    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.

The 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 Reversed 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
```
Reference parameters refer to the caller’s argument variable. Any value parameter is copied to the caller’s argument variable. Reference parameters are great if

1. sizeof(parameter) is big
2. called function must modify and/or
3. const reference parameters ditto.

Which of the XXXX’s is made 98??

RP = 98; Changes CV

Changes CV

Each value param.

When function is called,

fun(CV, CV, CV);
fun();

Refers to

void caller()
{
    int CV;
    int& RP; // reference parm.
    int VP; // value parameter.
    void fun(int VP, // value
              const int& CRP // const ref. parm.)
    {
        VP = VP + 1; // inside called fun's body...}

        const int CRP
        int RP; // reference param.
        int VP; // value parameter.
    void fun( int VP, // value
              const int& CRP // const ref. parm.)
    {
        VP = VP + 1; // inside called fun's body...}

        const int CRP
        int RP; // reference param.
        int VP; // value parameter.
    void fun( int VP, // value
              const int& CRP // const ref. parm.)
    {
        VP = VP + 1; // inside called fun's body...}

        const int CRP
        int RP; // reference param.
        int VP; // value parameter.
An annoying detail:

What if a constant or a computed value is used as an argument for a reference parameter?

Temporary variables are created, initialized to the argument values, and the reference parameters refer to THM.

Answer: Temporary variables are created, initialized to the argument values, and the reference parameters refer to THM.

```c
{ Rp1 = 0; Rp2 = 0; }

void fun(inta Rp1, inta Rp2)
{
    where fun() is

    fun(38', X+39');
}
```

6++ prints the Warning:

warnInt: initialization of non-const

reference 'int &' from variable 'int'

Reference 'int &' from variable 'int'.
must NOT PRINT ANY WARNINGS.

Rule for CS1310: All compilations must be done with -Wall and

0 (not 39) (bad)

Program compiles and might print

You get a warning about a conversion that discards const;

But the

cout >> TINTRE >> endt;

fun(TINTRE, TINTRE);

const int TINTRE = 39;

} What if? void caller()
curiously in each of the 3 variables, before and after each step.

Please simulate the execution of this by writing the values

\[
\begin{align*}
\text{J} & = \text{TEMP''} \\
\text{I} & = \text{J''} \\
\text{TEMP''} & = \text{I''} \\
\text{J''} & = \text{TEMP''} \\
\text{J''} & = \text{J''} \\
\text{TEMP''} & = \text{I''}
\end{align*}
\]

Try first J = I'' or first I = J''.

That cannot work!!

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

\[
\begin{align*}
\text{J} & = 2003'' \\
\text{I} & = 1928'' \\
\text{J''} & = \text{J''} \\
\text{I''} & = \text{I''}
\end{align*}
\]
address.

variable, or is it the value of a pointer variable, that is, an
variable, or else some illegal value.

store an address of a C++ int variable, or else it might have
name of a pointer type variable. The variable named PIVAR might
name is an address. PIVAR is really the
int *PIVAR; What is PIVAR? Is it a "pointer"?

int Integer.

variable whose type is int. This variable stores a C++
Most say "It's an integer but, really, IVAR is (the name of) an
int IVAR; What is IVAR?"

A linguistic pitfall—try not to fall into it!
Perhaps we should always use the word "address" for "pointer" what operations can be done. Each variable has a type which determines what values it can hold and Technically, "pointer" and "int" describe C++ types.
{ return 0; }

while (finish) { //
    cout >> A[0]; // A sorted string was printed.
    Now, A[0] is CHANGED TO

    [i] Now, A[i] has the smallest char from A[i]. Now! //

        for (j = i + 1; j < num; j++) { ++j; }
        for (i = 0; i < num - 1; i++) {
            while (cin >> get_string(A, 0); A[i] = '0', 0, \n            cout int A_size = 100; int num, i, j; char A[A_size];
        }
    main()

    using namespace std;
    #include <iostream>

    Selection sort demo: Processes chars within the array A.
Save a copy to help you begin future projects.

Pressing "enter" when you type more than 11 characters before.

HEADER ASAF: Write a program like this, and SEE WHAT

Everybody's textbook.

To get more details right now, read pages 183-187 of Main and

This program manipulates chars as if they were numbers.