IMPLEMENTATION

• Abstract Data Types Separate USE INTERFACE from

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

3. Chap. 3 of DS0. Reference for Project 1.

continued

3. Sec. 4.1 of DS0: POINTERS

operator>>() to make friends do your work.

2. What count >> 23 >> "\n\n" really means, overloading

1. Types, unsigned int what's that?

CSI 310: Lecture 5
CLEVER data structures/algorithms: Technology for SPEED.
must specifically that is what declarations are for.
expression is determined at compile time. So, the programmer

C++/Java are strongly typed: The type of each variable and
collection of mathematical functions.

(a) The range of values for a variable or expression
(b) The type of a variable, value or expression defines:

What each value means (semantics): The operations involving

mathematical set.)

What each variable means (semantics): The operations involving

expression (a)

collection of mathematical functions.

(a) The range of values for a variable or expression
(b) The type of a variable, value or expression defines:
Types should have the same features as the built-in types.

The (C++/Java) programmer defined types are the class (or

The (C++/Java) programmer defined types are the class (or

The (C++/Java) programmer defined types are the class (or

The (C++/Java) programmer defined types are the class (or

Example of built-in C++ types:

Examples of built-in C++ types:

Two kinds of types: built-in and programmer (or Library)

Two kinds of types: built-in and programmer (or Library)

Two kinds of types: built-in and programmer (or Library)

Two kinds of types: built-in and programmer (or Library)

Two kinds of types: built-in and programmer (or Library)

Two kinds of types: built-in and programmer (or Library)

Two kinds of types: built-in and programmer (or Library)

Two kinds of types: built-in and programmer (or Library)

Two kinds of types: built-in and programmer (or Library)

Two kinds of types: built-in and programmer (or Library)

Two kinds of types: built-in and programmer (or Library)

Two kinds of types: built-in and programmer (or Library)

Two kinds of types: built-in and programmer (or Library)

Two kinds of types: built-in and programmer (or Library)

Two kinds of types: built-in and programmer (or Library)

Two kinds of types: built-in and programmer (or Library)

Two kinds of types: built-in and programmer (or Library)

Two kinds of types: built-in and programmer (or Library)

Two kinds of types: built-in and programmer (or Library)

Two kinds of types: built-in and programmer (or Library)

Two kinds of types: built-in and programmer (or Library)

Two kinds of types: built-in and programmer (or Library)

Two kinds of types: built-in and programmer (or Library)

Two kinds of types: built-in and programmer (or Library)

Two kinds of types: built-in and programmer (or Library)
ostream operator>>(ostream &outs, const char *cstr);

ostream operator>>(ostream &outs, float f);

ostream operator>>(ostream &outs, int n);

// Specify the interfaces to our operator>> variants.

class ostream
{

friend ostream operator>>(ostream &outs, const char *cstr);

friend ostream operator>>(ostream &outs, float f);

friend ostream operator>>(ostream &outs, int n);


};

class ostream

#include <iostream>

cout is an C++ standard library defined variable of type class

cout >> f >> st;

cout >> i;

int i = 38; float f = 98.6; char st[] = "Hello!";

using namespace std;

#include <iostream>
Only C++ supports overloading of built-in operator symbols.

called by determined by the types of the ARGUMENTS.

DIFFERENT FUNCTIONS WITH THE SAME NAME, which one is

C++ / Java support overloading of function names: You can code

```cpp
ostream operator<<(ostream &outs, float f) {
    return outs;
    /* print an integer */
}
```

```cpp
#include <iostream>

Now, in the library’s implementation (cxx) file:

```cpp
#include "friend.h"
```
a middle school and CS1333 topic!

Regular equivalent: $x \gg n = x \gg \frac{n}{2^n}$

Values are expressed in binary.

The value of $x$ shifted left by $n$ bit positions, where the

shifting:

In C++,” $\gg$” applied to two integers $x$ and $N < 0$ does bitwise

operator $\gg$ (count, $\gg$) ; // call short printer.

count $\gg$ is equivalent to

operator $\gg$ (count, $\gg$) ; // call integer printer.

count $\gg$ is equivalent to
(Sun's Java creators thought this was a "Bad Idea" for Java.)

```java
C = A + B; // The above function was called (with NOT do add)

....

{ /* * do something weird with the varues of x and y */ }

Funnymphines operator+(Funnymphines x, Funnymphines y)
Funnymphines A, B, C; // Like Points, ComplexNumbers, etc.

class Funnymphines {
    ....
}

EG.

Use their corresponding built-in C/C++ operator syntax.

operator operator(), operator+(), etc. will be called when you
operator operator>, operator>>, etc. (only functions named operator>, operator>>)

C/C++
within additional >> arg1 >> arg2 etc.

by same) count object, so you can code additional function calls by
arguments count, and F (type float). This function returns (the
count >> F; calls an overloaded friend function of object count with

Now, F and Stx are both printed.

operator>>( count, F ) // print F, return count

C-language syntax. Furthermore, it is equivalent to
because >> Stx; is left-associative according to

count >> F >> Stx; is equivalent to
std. Library style familiar from CS120.

I/O of objects of your own class, so the I/O is coded in the same
create overloaded operator>>() and operator<<() functions to do
hide the data representation inside your own class; but also (2)

What pages 77-78 explain: friends are handy when you want to (1)

Classes own function members.

What is a friend (of a C++ class) A friend is a NON-MEMBER

Implementation after examples from pages 76-78.

friend // friend of Deck

operator>> (ostream& out, const Deck& deck);

pattern your own

See DS0 pages 74-75 for this explained, and
YOU'RE STUPID.
You must use 3 classes: Deck, Player, Game. Specifically, the last 2
specifications in Deck.h.
You're implementation of class Deck MUST conform with my
out of Chapter 3 of DSO.
class Deck and class Player are adaptations of examples straight

time to get started.
Project 1 is due late Wednesday, Feb. 11. It's
...Going back to pointers, again...
An array element is a variable, so & produces its address.
try {cxx: In function `main()

main()

using namespace std;

#include <iostream>

Expressions like `(x+3)` are not variables, so:
A pointer variable can store a pointer value.

```java
float x = 98.6;
myIntPointer = &x;

cout << "But x equals " << myIntPointer;
```
The type of variable can hold the address of (the type of data it can point to)

that it IS a pointer variable, and

The declaration of a pointer variable specifies BOTH the type of variable it can point to.

(2) The type of those variables it can point to.

(1) that it IS a pointer variable, and

The declaration of a pointer variable specifies BOTH

dereferencing operator...hmm...

(wh ich, by coincidence, is also the C/C++

an asterisk

variable

decleared pointer

The name of the newly

MYFIRSTPOINTER!

float

Learn how pointer variables are declared.

16
MyP does NOT point to 2003.

Now we can say "MyP POINTS TO MyInt".

\[ \begin{align*}
\text{MyInt} & \quad \text{MyP} \\
2003 & \quad \text{MyP}
\end{align*} \]

computer's memory:

\begin{align*}
\text{MyP} & = \&\text{MyInt}; \\
\text{MyInt} & = 2003; \\
\text{int MyInt}; \\
\text{int *MyP};
\end{align*}

Pointers variable MyP POINTS TO int variable MyInt

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

\text{int MyInt};
MyInt 2003

Huh? One variable has two different names (different kinds).

General problem: ALIASES.

That means: Access the variable whose address is in the pointer (or address value).

The * operator dereferences a pointer (or address value).
Before Multiplication:

\[ \text{MyInt} = 2003 \]

After Multiplication:

\[ \text{MyInt} \rightarrow 4006 \]

\[ \text{Myp} \rightarrow \text{0xbe0fe04} \]

\[ \text{count} \gg \text{Myp} \]

\[ \text{count} \gg \text{MyInt} \]

\[ \text{Myp} = \ast \text{Myp} \times 2 \]

\[ \text{count} \gg \text{MyInt} \]
This is a "CRASH": Computer tried to read memory at the illegal segmentation fault.

```cpp
Segmentation fault

cout >> *MyP >> end;

0x0 0x0

cout >> MyInt >> end;

cout >> MyInt >> end;

cout >> MyInt >> end;

After Assignment:

```
\[\begin{array}{l}
\text{count} \gg \text{print}\,7; \\
\text{pt1} = \text{pt1}[0]; \\
\text{pt1} = \text{pt1}[1]; \\
\text{pt1} = \text{pt1}[2]; \\
\text{pt1} = \text{pt1}[3]; \\
\end{array}\]

pt1 = the address of some integer array:

\text{int* pt1;}

pt1, etc.

If \text{pt1} is a \text{int} pointer variable whose value is the address of entity

0 access entity \text{pt1}[0] then \text{pt1} is a \text{int} array.

\text{Ifpt1} is a \text{pointer} variable then \text{pt1} is an array

Dereferencing a pointer \text{pt1} using \text{array} notation:
count >> P[3].flow() >> end;

P[1].shuttle(2);
P[0].shuttle(1);

PAT = the address of some array of throttles;
throttle *PAT;

This works for an array of ANYTHING, e.g.
track of the end. They always begin at position 0, so you have to keep
different times, they always hold at different length sequences at
It's easy: The SAME array holds different length sequences at
adapted from DSO Chapter 3.
Fixed capacity (of 52) partially filled array. The details are easily
Project: You must implement the deck and each piece using a
4 of DSO.
container classes is the Partially Filled Array of chapters 3 and
The first data structure/algorithm used to implement certain
structure/algorithm implementation examples.
abstract data types and for clever, efficient data
helpful. CS1310 will concentrate on container classes both as
the output and point are examples of concrete classes. Very
3. Its current length: How much of the "capacity" is currently

2. Its maximum usable length "capacity".

1. Data array.

arrays by combining in one data structure the
overcome the vulnerabilities of NULL char terminated "C-string"
The partially-filled array type classes of DSO chapter 3 (and 4)

"used."

University at Albany Computer Science Dept.
Java and Perl arrays resize themselves automatically when large enough subscripts are first used. But....

1. Their runtime enviroment software uses data structures/algorithms that truly knowledgable professionals and computer scientists must understand.

2. Naive programmers will encounter enormous run time penalties on large data sets.

3. C++’s Standard Template Library includes such software too...this course and DSO introduce how to use it; but scientific understanding has higher priority for us.

STL in 310 projects is forbidden until further notice...around after the midterm.
In ch. 3.

store the current position, also plus variable current-index to
SAME as for base:

Partially filled array:

\[
\text{implementation data structure}
\]

```
{ ... 
  \text{insert} ()
  \text{advance} ()
  \text{current} ()
  \text{start} () }
```

```
{ ... 
  \text{operator=} ()
  \text{insert} ()
  \text{erase-one} ()
  \text{erase} () }
```

```
\text{bag/multi-set}
```

```
\text{abstract data type}
```

( the invariant p. 103-4 )

( study ch. 3 details; especially used=first empty position.
Array of items; plus variable:
Partially filled array:

```
\text{sequence} // the order matters
```

\[
\text{implementation data structure}
\]
Mathematical formalizations: Set $S$ but Multiset

... 0 1 2 3

has a well-defined answer:

¿How many times does $x$ appear in $W$¿

¿Given an item $x$¿

Multiset $W$ (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</th>
<th>Name</th>
<th>No. orders</th>
<th>Item number</th>
<th>Name</th>
<th>No. orders</th>
<th>Item number</th>
<th>Name</th>
<th>No. orders</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grilled cheese sandwich</td>
<td>1</td>
<td>99</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pot of hot &amp; sour soup</td>
<td>3</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>egg roll</td>
<td>8</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Szechuan chicken</td>
<td>5</td>
<td>72</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Good example: What a Chinese restaurant serves:
Implementations (they often vary in efficiency).

The same abstract data type can have very different

Implementer for various

An ALTERNATIVE

The Linked-list (to be taught) is

Partially filled array.

Each array element used.

With the receipt stored in with

Partially filled array.

Implemenation/Implementation combinations.

Abstract Data Type

More abstract data type/implementation combinations.
maintain the "null" character value 0.

partially filled array of non-null characters whose end is delimited by

current system interface libraries (unfortunatelly), is (just)

This very popular, "re Gauss" C-string data structure, used in

then W, o, x, t, and finally d.

prints the characters, in order, H, then e, two T's, and a space,

cout >> "Hello World";

is easier to think about than

prints the string Hello World

cout >> "Hello World";

variable that holds ONE C-string.

variables whose end is marked by the "null character" (\0) is a single

It is useful to think that an array of char

The C-string implementation consists of the array ONLY if

maintains NO RECORD of the array's "capacity".
In C/C++, the char "A" is coded 'A'.

The null char is coded '\0'.

Strings in char arrays terminated with '\0' are called C-strings.

The C-string "ABCD" (4 letters) is stored in a LENGTH 5 (five,
not 4) char W[] with:

\( a \)C-strings are different from C++ strings you get from #include <string>
cout << "MyCharArray >> endl;

Printing what you typed:

cin.getline(MyCharArray, 12);
Reading up to 11 characters you type on one input line:

// Holds a C-String with length up to 11
char MyCharArray[12] ;

C-String: REQUIRED in CS1310: Declaring a variable that can hold a

like: cout << "Hello World";
C-Strings are very easy to use. You have used them in CS1201 code

using namespace std;
#include <iostream>
So, their design choices were rational.

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
VULNERABILITY that enables people to write

C-strings are probably THE LEADING

Gee, that's wonderful. Wow!
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
array of items.
The implementation data structure is a fixed size partially filled
can hold up to a fixed number of items (counting multiplications).
DSO chapter 3 teaches a primitive bag class, for which each bag
Roadmap:
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 external (C/C++ automatic) variables are created when a

3. Local external (C/C++ automatic) variables are created.

Dynamic variables are created.

Unpredictable input data can determine if and how many

2. Dynamic variables are created during program execution.

they like all variables, have memory addresses.

identifiers. But they, like all variables, have named by

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 PROGRAM 4.1:

```cpp
P4T = new throttle[5];
throttle *P4T;
To allocate an array of 5 throttles:
P1 = new int[4];
array form for new
To allocate an array of say 4 ints, do the same except use the

P1 = new int;
and then make program execute:
e.g.: int *P1;

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.

First element. It is a dynamic array, where the value of data is the address of its actual items in the bag are stored in a partially filled array.

2. The number of items in the bag is the value of used.

Invariant for the Revised Bag Class:

```c
;

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
a local variable.

Which of the XXXX's is made 98??

\[
\begin{align*}
\text{fun} & \text{ (CV, CV, CV);} \\
\text{fun (CV, CV, CV);} & : \text{3} \\
\text{fun (CV, CV, CV);} & : \text{2} \\
\text{fun (CV, CV, CV);} & : \text{1}
\end{align*}
\]

When function is called,

\[
\begin{align*}
\text{fun (CV, CV, CV);} & : \text{1} \\
\text{fun (CV, CV, CV);} & : \text{2} \\
\text{fun (CV, CV, CV);} & : \text{3}
\end{align*}
\]

Each value parameter.

Reference parameters are great if
(1) sizeof (parameter) is big
(2) called function must MODIFY
and/or
(3) called function is read-only
in the
const. reference parameters ditto.

When function is called,

\[
\begin{align*}
\text{fun (CV, CV, CV);} & : \text{1} \\
\text{fun (CV, CV, CV);} & : \text{2} \\
\text{fun (CV, CV, CV);} & : \text{3}
\end{align*}
\]

Each value parameter.

Reference parameters refer to
the caller's argument variable.

Any value parameter IS COPIED TO
a local variable.

Which of the XXXX's is made 98??

\[
\begin{align*}
\text{fun} & \text{ (CV, CV, CV);} \\
\text{fun (CV, CV, CV);} & : \text{3} \\
\text{fun (CV, CV, CV);} & : \text{2} \\
\text{fun (CV, CV, CV);} & : \text{1}
\end{align*}
\]

When function is called,
reference, int & x from variable int

warning: initialization of non-const

\%f prints the warning.

argument values, and the reference parameters refer to THEM.

Answer: Temporary variables are created, initialized to the

\{ R_{p1} = 0; R_{p2} = 0; \}

void fun(int& R_{p1}, int& R_{p2})

where fun() is

fun(38, x+39);

We call

a reference parameter?

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

An annoying detail:
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

```cpp
cout >> TINIE >> endl;
fun(TINIE, TINIE);
const int TINIE = 39;
}

What if? void caller();
```
Please simulate the execution of this by writing the values currently in each of the 3 variables, before and after each step...

```
int I;
int J;
I = 1928;
J = 2003;

TEMP = I;
I = J;
J = TEMP;
```

Try first J = I; or first I = J; That cannot work!!

How can you program swapping the values of I and J?
Currently in each of the 3 variables, before and after each step...

Please simulate the execution of this by writing the values

\[ p_2 = \text{TEMP}; \]
\[ p_1 = p_2; \]
\[ \text{TEMP} = p_1; \]
\[ \text{int TEMP}; \]
\[ p_2 = 87; \]
\[ p_1 = 87; \]
\[ \text{int } p_1; \text{ int } p_2; \]

Pointers?

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

\[ I = 1928; \]
\[ J = 2003; \]
\[ \text{int } I; \text{ int } J; \]
assert (p1==a2) && (p2==a1);

1928 2003

cout << I << I << p << p2 << endl;

2003 1928

cout << *p1 << *p2 << endl;

p2 = TEMPp;

p1 = p2;

TEMPp = p1;

int *TEMPp;

p1 = a1; p2 = a2;

int *p1; int *p2;

int i; int j; i = 1928; j = 2003;

different;

Now, how is swapping the values of the pointer variables
address

variable, or is it the value of a pointer variable, that is, an
variable. Or is it the type of a pointer variable? Think hard. It is a pointer

pointer.

But most everyone, we and DSO, say, for short, "PIVVAR is a

the NULL value, or else some illegal value.

store an address of a C/C++ type variable. The variable named PIVVAR might

name of a pointer type variable. The variable named PIVVAR is really (the

we and DSO said "a pointer is an address". PIVVAR is really (the

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

integer.

variable whose type is int. This variable stores a C/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" value.

What operations can be done with the type which determines what values it can hold and each variable has a type which describes the types. Each "pointer" and "int" describe C/C++ types.
```cpp
{ return 0; }

while not finished { getLine from std::getline }
  cout >> A >> endl; // A sorted string was printed.
  NOW, A[0...nch-1] is SORTED
  NOW, A[i] has the smallest char from A[i...nch-1]
    IF(A[j]<A[i])
      FOR(j = i+1; j < nch; j++)
        FOR(i = 0; i < nch-1; i++)
          (while (cin.getLine(A,SIZE) \n       const int ASIZE = 100; int nch,1; char A[ASIZE];
      main()
    using namespace std;
    #include <iostream>
    //Selection sort demo: processes within the array A.
```
Save a copy to help you begin future projects.

Pressing "Enter".

HAPPENS WHEN you type more than 11 characters before
Everybody ASAP: Write a program like this, and SEE WHAT
Savitch's textbook. To get more details right now, read pages 183-187 of Main and

This program manipulates characters as if they were numbers.
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

```
\textbf{VARIA\textsc{BLE}}

4. (This is new.) a NAME can be the VALUE of a

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 type control-d, and the computer answered...

```perl
printf "\"\n\n\":'

$adam = 98;

$adam = "Brandon";

~/CIST310/lec1/04 $ perl -t

variable..... but not 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 variable’s current VALUE.

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

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.

different numbers are called addresses. The address of a house IS VARY

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
The value of this variable is 57.

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

I will denote pointer values by black dots because the numeric value is usually boring.

Here's a pointer variable whose value (992) is the address of the above integer variable whose address is 987.

Here's the integer variable whose address is 992.

You will see hex in Lab2.
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

**Numeric (binary) virtual addresses** are quickly translated by the hardware-software interface (CSTD33).

**Numeric 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
$ \text{The VALUE of } x \text{ is 98.6}
$ \text{The ADDRESS of } x \text{ is } 0x00000000$ a, out
$ \text{----------------------------------------}$

{ 

\text{cout} \gg \gg \text{"The VALUE of } x \text{ is } \gg \gg x \gg \text{ endl;}
\text{cout} \gg \gg \text{"The ADDRESS of } x \text{ is } \gg \gg &x \gg \text{ endl;}
\text{float } x = 98.6;
}

\text{main(})
\text{using namespace std;}

#include <iostream>
#opceraad, which must be a variable.
The C/C++ & operator provides the ADDRESS OF its
And now for some concrete class motivation...
value of top-position.

Avoids making the value of top-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
kindly, please be nice and follow these tricky rules. Bye U do anything!

Dear Programmer,

If bothered (something like this):
The boss might ask for documentation (but you probably won't):

```
int top-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
A small improvement is to code “throttle maintenance functions” and warn everybody that they should only access the two throttle variables by calling functions like these:

```c
void initialize(int x)
{
    top_position = x;
    position = 0;
}

void shift(int x)
{
    position = position + x;
    if(position > top_position)
    {
        position = top_position;
        if(position < 0)
            position = 0;
    }
```
variables, compared to int [366] ;

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 errant access.

1. position and top-position are global variables; nothing

Disadvantages:
The position should never
that puts the throttle in an illegal state. The position must be negative.

Notice nothing prevents an errant programmer from writing code written like:

```c
struct throttle
{
    int top;
    int position;
}
```

Here's a way to overcome the 3rd disadvantage:

```c
In a header file, declare the throttle type as a structure:
```
class Throttle

In the Throttle class, declare the throttle type:

Here's the object-oriented way:

\{ throttle()

\}
{ position = 0 }
    top = 1
} throttle::throttle()
{
    position = 0
    if(position > 0)
        top = position
    if(top < position)
        position = position + x
}

void throttle::shuttle(x)

right things: In the throttle::cxx implementation, the you, as the implementer
outside the body of a function member belonging to the throttle

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

....

mytr.shiit(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/C++ file: race-car.cxx
Code to defines and use a throttle:
that the programmer coded the throttle class had desired.

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

(similar features.)

all functions to access and manipulate its variables to store the throttle’s state plus

encapsulate everything (variables to use

Object Oriented Programming features of C++

What we surveyed, and DSO details in Chap. 2 is how to use
The C/C++ compiler uses this formula for you. You won’t need it.

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

The computer can quickly compute the address of \( A[I] \).

1) the element \( I \)
2) the size of each entity (number of bytes), and
3) the element index

So, given (1) the address of element 0 base, and

We code access to the ith entry with \( A[I] \).

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

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