

Unit One: Binary & Simple Variables

Introduction to Data Structures

Kinds of Data 2



NUMERICAL DATA

Integers: -12 0 100

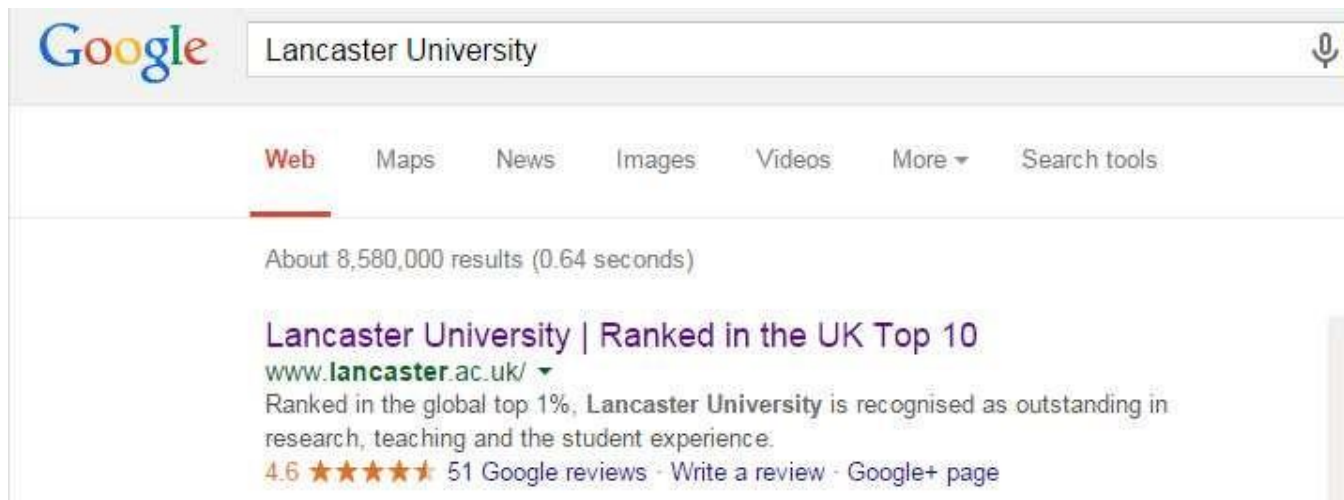
reals or floating numbers (fractional numbers): 3.14, 88.2, 0.157, 7.39×10



TEXTUAL DATA

Characters: 'w' 'G' '6' '(' '%' '-'

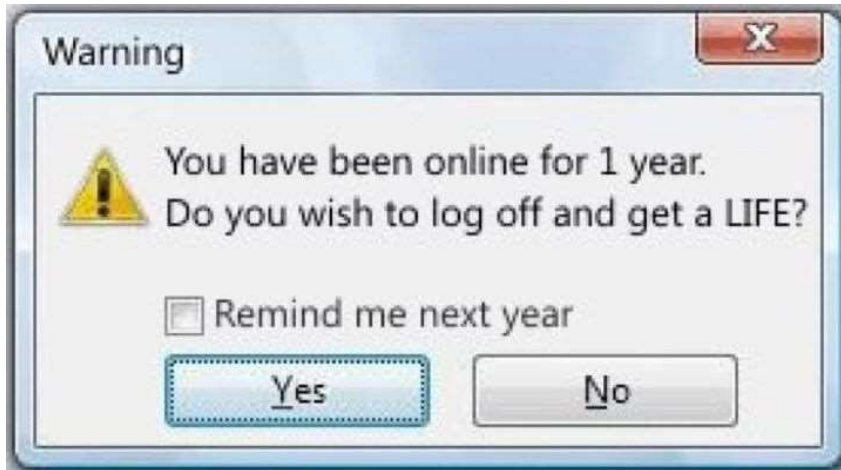
Strings (sequences of characters): "I detest him!" "-88.6"



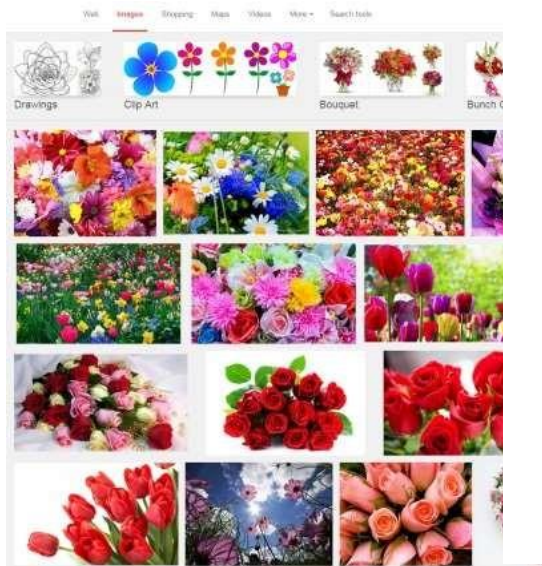
Kinds of Data

LOGICAL DATA

Booleans, truth values: true/false, yes/no, on/off



More Complex Data: **Image, Audio, Video etc.**



beyond scope of this course.

Numbering Systems

- A numbering system assigns meaning to the position of the numeric symbols.
- For example, consider this set of symbols:

532

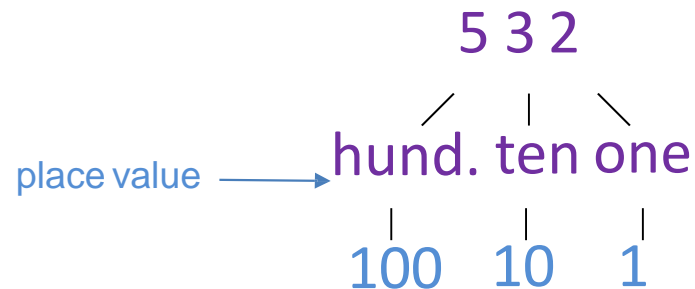
- What number is it? Why?

Numbering Systems – Positional Notation

6

It depends on the numbering system.

532 in base 10 (decimal) is

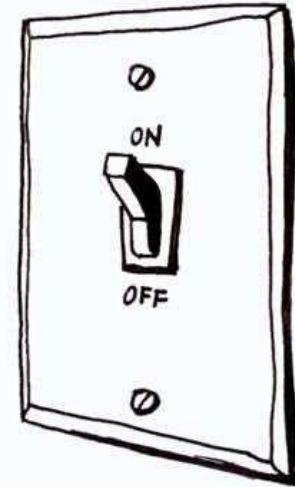


$$5 \times 100 + 3 \times 10 + 2 \times 1 = 532$$

Binary Systems

A computer is an electronic machine.

- works by controlling the flow of electrons
- presence of a voltage – digital state “1”
- absence of a voltage – digital state “0”



- Computers use a numbering system which has exactly 2 symbols, representing on and off.

Binary

Decimal is base 10 and has 10 digits:

0,1,2,3,4,5,6,7,8,9

Binary is base 2 and has 2 digitals, so we use only 2 symbols:

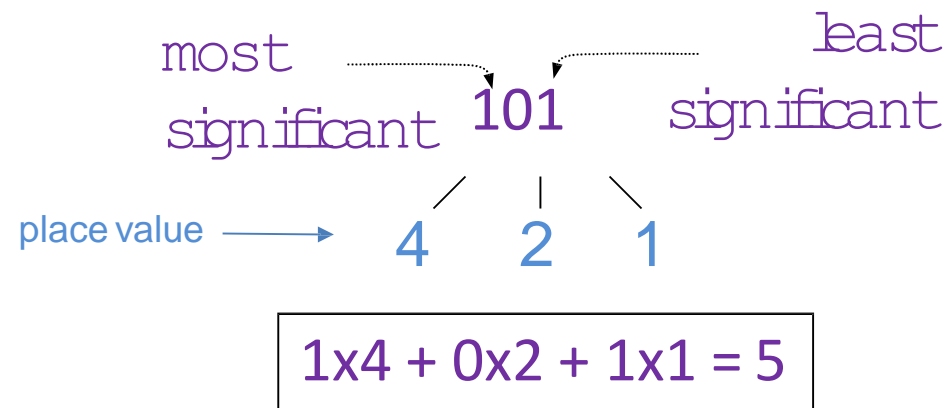
0,1

A binary digit or bit can take on only these two values.

Binary numbers are built by concatenating a string of bits together. Example: 10101010

Binary Positional Notation

101 in base 2 positional notation is



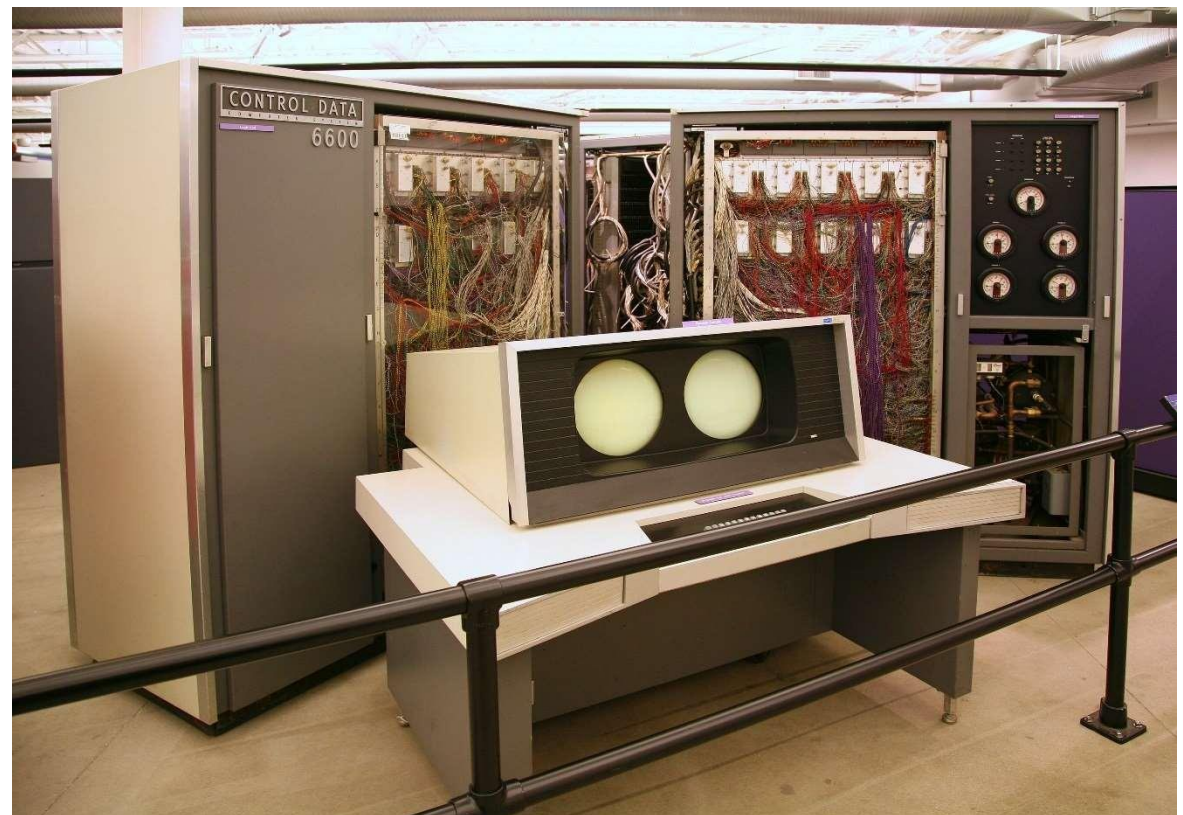
Question: 1010 = ?

Binary States¹⁰

- A bit has two possible states: 0 or 1
- Values with more than two states require multiple bits.
 - A collection of two bits has four possible states:
00, 01, 10, 11
- A collection of n bits has 2^n possible states.

NUMERICAL DATA

Integers & floating point numbers



Unsigned Integers (cont.)

- An n -bit unsigned integer represents 2^n values:
from 0 to $2^n - 1$.

4	2	1	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Unsigned Binary Arithmetic

- Base-2 addition – just like Base 10!
 - add from right to left, propagating carry

$$\begin{array}{r}
 10010 \\
 + 1001 \\
 \hline
 11011
 \end{array}
 \qquad
 \begin{array}{r}
 \begin{array}{c} +1 \text{ carry} \\ \curvearrowright \end{array} \\
 10010 \\
 + 1011 \\
 \hline
 11101
 \end{array}
 \qquad
 \begin{array}{r}
 \begin{array}{c} \curvearrowright \quad \curvearrowright \quad \curvearrowright \quad \curvearrowright \\ 1111 \end{array} \\
 + 1 \\
 \hline
 10000
 \end{array}$$

$$\begin{array}{r}
 10111 \\
 + 111 \\
 \hline
 \end{array}$$

Subtraction, multiplication, division,...

Signed Integers

- How do we write negative binary numbers?
- Historically: 3 approaches
 - Sign-and-magnitude
 - 1's-complement
 - 2's-complement

Signed Integers : sign-magnitude

- Positive numbers
 - are represented by having a leading 0 (sign-bit), the rest bits represent the magnitude (value)

0101 = 5

- Negative numbers
 - Leading bit is 1, other bits are the magnitude (value):

1101 = -5

N = 4	Number Represented	
Binary	Unsigned	Signed Mag
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-0
1001	9	-1
1010	10	-2
1011	11	-3
1100	12	-4
1101	13	-5
1110	14	-6
1111	15	-7

Problems of Sign-magnitude

- Two zeros, +zero (0000) and -zero (1000)
- Arithmetic circuits are complex
– e.g. $2 + (-3)$

$$\begin{array}{r}
 0010 \text{ (2)} \\
 + 1011 \text{ (-3)} \\
 \hline
 1101 \text{ ???} \\
 1001 \text{ (-1)}
 \end{array}$$



N = 4	Number Represented	
Binary	Unsigned	Signed Mag
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-0
1001	9	-1
1010	10	-2
1011	11	-3
1100	12	-4
1101	13	-5
1110	14	-6
1111	15	-7

Signed Integers: 1's Complement

- Flip every bit to represent negative

• 0101 = 5

• 1010 = -5

- Have the same problems as Sign-magnitude

- Two zeros

- Arithmetic circuits are complex

- e.g. try $4 + (-3)$

$$\begin{array}{r}
 0100 \text{ (4)} \\
 + 1100 \text{ (-3)} \\
 \hline
 1 \ 0000 \text{ ??} \\
 0001 \text{ (1)}
 \end{array}$$



N = 4	Number Represented		
Binary	Unsigned	Signed Mag	1's Comp
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	8	-0	-7
1001	9	-1	-6
1010	10	-2	-5
1011	11	-3	-4
1100	12	-4	-3
1101	13	-5	-2
1110	14	-6	-1
1111	15	-7	-0

Signed Integers: 2's Complement

- To get the negative representation of positive number x
 1. find the 1's complement of x
 - By flipping bits
 2. and then add one to the 1's complement

$$\begin{array}{r}
 10011011 \\
 01000100 \leftarrow \text{1's complement} \\
 + 1 \\
 \hline
 01000101 \leftarrow \text{2's complement}
 \end{array}$$

Signed Integers: 2's Complement 19

Representation

- Positive numbers or zero
 - normal binary representation
 - Leading bit is always 0
- Negative numbers
 - start with positive number
 - take its 1's complement
 - then add one

2's complement is used by all modern computers

N = 4	Number Represented			
Binary	Unsigned	Signed Mag	1's Comp	2's Comp
0000	0	0	0	0
0001	1	1	1	1
0010	2	2	2	2
0011	3	3	3	3
0100	4	4	4	4
0101	5	5	5	5
0110	6	6	6	6
0111	7	7	7	7
1000	8	-0	-7	-8
1001	9	-1	-6	-7
1010	10	-2	-5	-6
1011	11	-3	-4	-5
1100	12	-4	-3	-4
1101	13	-5	-2	-3
1110	14	-6	-1	-2
1111	15	-7	-0	-1

Normal Addition Circuits with 2's²⁰ Complement

- 2's comp. addition is just binary addition

- Assume all integers have the same number of bits
- Ignore carry out
- e.g. $4 + (-3)$

$$\begin{array}{r} 0100 \text{ (4)} \\ + 1101 \text{ (-3)} \\ \hline 1\ 0001 \text{ (1)} \end{array}$$

Assuming 4-bit 2's complement numbers.

N = 4	Number Represented			
Binary	Unsigned	Signed Mag	1's Comp	2's Comp
0000	0	0	0	0
0001	1	1	1	1
0010	2	2	2	2
0011	3	3	3	3
0100	4	4	4	4
0101	5	5	5	5
0110	6	6	6	6
0111	7	7	7	7
1000	8	-0	-7	-8
1001	9	-1	-6	-7
1010	10	-2	-5	-6
1011	11	-3	-4	-5
1100	12	-4	-3	-4
1101	13	-5	-2	-3
1110	14	-6	-1	-2
1111	15	-7	-0	-1

Converting Binary (2's C) to Decimal

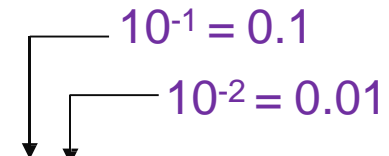
1. If the leading bit is 0, treat it as an unsigned binary representation
2. If leading bit is 1, take 2's complement to get a binary representation for the positive number.
 1. Convert the binary to decimal
 2. Then add a minus sign.

$$\begin{aligned} X &= 01101000_b \\ &= 64+32+8 \\ &= 104_d \end{aligned}$$

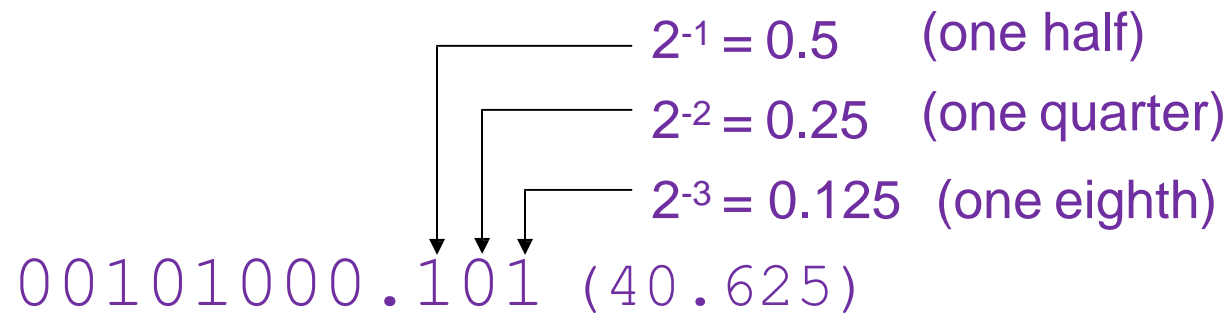
$$\begin{aligned} X &= 11100110_b \\ -X &= 00011010 \\ &= 16+8+2 \\ &= 26_d \\ X &= -26_d \end{aligned}$$

Binary Fractional Numbers

- Decimal real (fractional) numbers: 200.52



- Same rules apply in binary:



11111110.110 (-2.25) ← 2's complement rep.

Binary 011.110 = ? 3.75

Decimal Floating Point Numbers

- A real decimal value (base 10) can be defined by the following scientific notation:

$$\text{sign} * \text{fraction} * 10^{\text{exp}}$$

The fraction is a decimal number between 1 and 10.

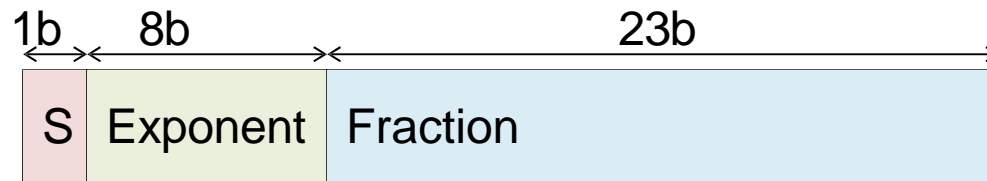
$$12345.67 = 1.234567 \times 10^4$$

The representation is called **floating point** because the decimal point “**floats**”

Binary Floating Point Numbers

IEEE 754 Floating-Point Standard (32-bits):

$$\text{sign} * \text{fraction} * 2^{\text{exp}}$$

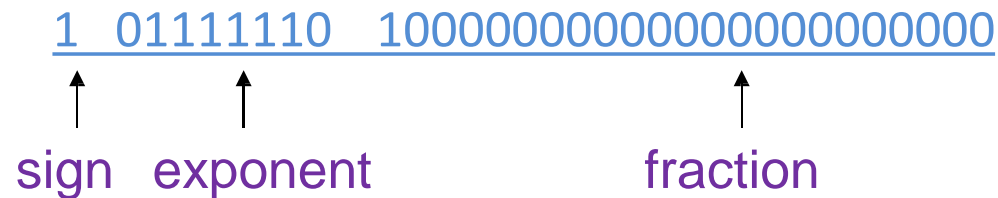


$$N = (-1)^S \times 1.\text{fraction} \times 2^{\text{exponent}-127}, \quad 1 \leq \text{exponent} \leq 254$$

$$N = (-1)^S \times 0.\text{fraction} \times 2^{-126}, \quad \text{exponent} = 0$$

Floating Point Example

- 32-bit IEEE floating point number:



$$N = (-1)^S \times 1.\text{fraction} \times 2^{\text{exponent} - 127}$$

- Sign is 1 -> negative number
 - Exponent is 01111110 -> 126 (decimal).
 - Fraction is 0.100000000000... -> 0.5 (decimal).
- Value = $-1.5 \times 2^{(126-127)} = -1.5 \times 2^{-1} = -0.75$.

OTHER 'SIMPLE' VARIABLES

byte

- A byte contains 8 bits.
- Some programming languages have a “byte” data type.

	type name	bytes
C	char	1
C#	byte	1
Java	byte	1

values	min	max
signed	-128	+127
unsigned	0	+255

Booleans : possible representations

- One bit representation
 - 0 is false
 - 1 is true
- Multi-bit representation
 - Inconvenient to manipulate only one bit
 - In C:

0000...0000	is false
all other	are true

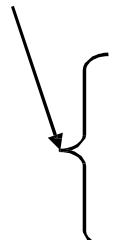
Character

- character = { '0', '1', '2', '3', ... 'a', 'b', 'c', ... 'Y', 'Z', ... '\$', '@', '?', '+', '-', '(', '~', ']' }
 - (hundreds of possible values)
- characters are mapped onto integers behind the scenes

ASCII

Special
Control
characters

ASCII: Maps 128 characters to 7-bit code.



0	NUL	1	SOH	2	STX	3	ETX	4	EOT	5	ENQ	6	ACK	7	BEL
8	BS	9	HT	10	NL	11	VT	12	NP	13	CR	14	SO	15	SI
16	DLE	17	DC1	18	DC2	19	DC3	20	DC4	21	NAK	22	SYN	23	ETB
24	CAN	25	EM	26	SUB	27	ESC	28	FS	29	GS	30	RS	31	US
32	SP	33	!	34	"	35	#	36	\$	37	%	38	&	39	'
40	(<p>Are 128 characters enough? http://www.unicode.org/ </p>													
48	0														
56	8														
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[92	\	93]	94	^	95	_
96	`	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124		125	}	126	~	127	DEL

char

- Java uses UCS-2 (Universal Character Set) to encode character values.
 - two bytes (16 bits) to represent a single character.
- C uses ASCII, an 8-bit code to represent characters.

	type name	bytes
C	char	1
Java	char	2

Enumerated Types

- Many languages ('C', Java) allow the programmer to define special discrete types by listing all the possible values, e.g.:—

```
gender  is  { male, female, unknown }  
status is  {working, unemployed, retired}
```

- Support for enumerated types differs among languages.

Enumerated Types (2)

- Variables of such types can be declared, initialised, tested, etc., much as with integers, floats, reals etc.

```
status employment;
```

```
employment = working;
```

```
if (employment == retired) {...}
```


STORAGE, MACHINE SIZE,
DISCRETE VS CONTINUOUS
VALUES

Memory

- A common approach is to have memory as a series of memory cells, each capable of storing a number of **bytes**
- Each memory cell has a unique address that allows us to access data in that cell

address value

0

23

1

65

2

255

...

...

n

199

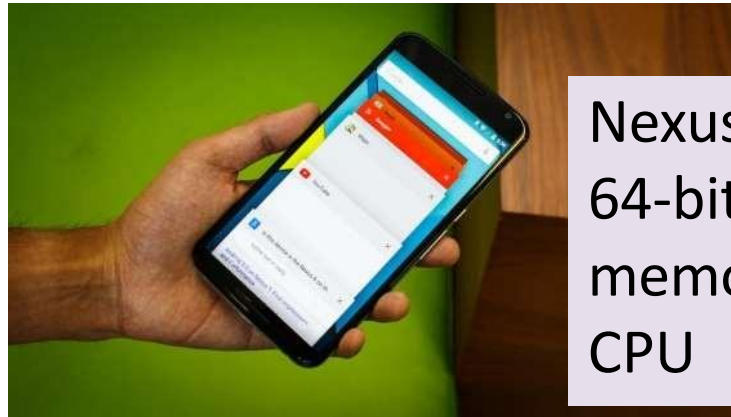


Word

- A computer **word** is a group of bits which are passed around together during computation.
- The **word-size** of the computer's processor is how many bits can be moved in/out at once.



Apollo Guidance Computer
16-bit, 64Kbyte memory,
0.043MHz CPU



Nexus 6 Phone
64-bit, 32GB
memory, 2GHz
CPU

Discrete Values

- Discrete values include:
 integers, characters, Booleans
- 'Discrete' means there are no 'in-between' values. There is
 - no integer between 12 and 13
 - no character between 'c' and 'd'
 - no Boolean value between true and false

Continuous Values

- Continuous values include floating point numbers
 - How many floating point numbers between 11.50 and 12.00?
- Continuous Values are represented approximately using a fixed number of bits
 - How will you represent floating numbers between 0.0 and 1.0 with two bits?

Floating Point Rounding Errors

Decimal example: $\frac{1}{3} = 0.3333333333$ 10-digital decimal rep.

When a calculator adds up three $\frac{1}{3}$, we get: 0.9999999...

The same problem for binary floating point arithmetic

$0.10_{10} \approx 0.0001100$ (binary)

$$0.10_{10} \cong \frac{0}{2} + \frac{0}{4} + \frac{0}{8} + \frac{1}{16} + \frac{1}{32} + \frac{0}{64} + \frac{0}{128} + \dots$$

$$0.10 + 0.20 = 0.2999999999999999$$

Floating Point Number⁴¹ Comparision

- So, the test

```
e = a + b;  
if (e == 0.3) {...}
```

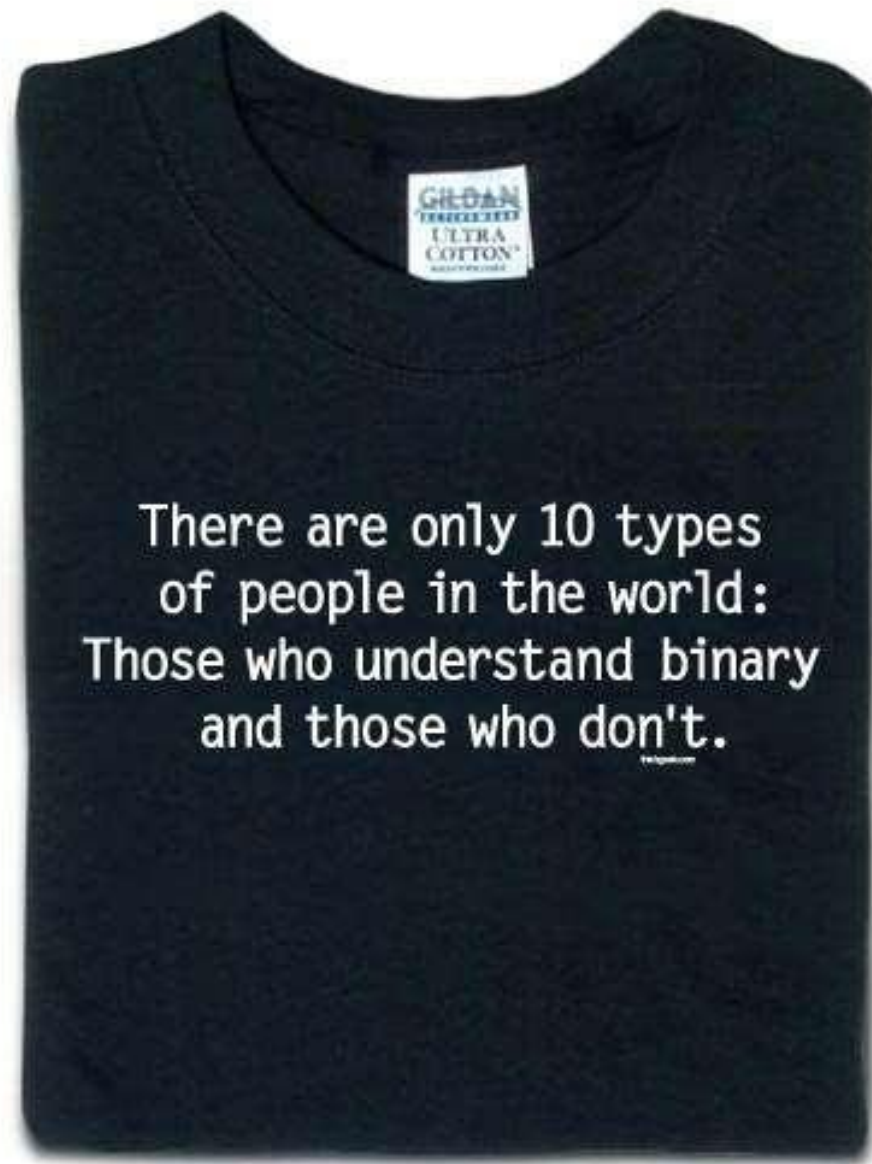
is **unsafe**

- Instead, you should do something like this:

```
e = a + b;  
if (abs(e-0.3) < 0.00000001) {...}
```

where function `abs()` in C returns the absolute value of the difference between `e` and `0.3`

THE END



Unit Two

Simple Arrays

SCC 120 Introduction to Data

Just imagine ...

- You are asked to write a program that analyses a piece of text ...
- Let's say you only have to count up the occurrence of lower-case vowels ...
 — a, e, i, o, u

SCC Student Handbook

07/15/15 11:02:02

the start of your final year you have the necessary work experience, other extra-curricular activities and knowledge of the job market to put together a successful application for your first graduate job.

2. Extracurricular Activities and the Lancaster Award

At Lancaster we not only value your academic accomplishments, but also recognise the importance of those activities you engage with outside your programme of study. The student experience is enhanced by including extracurricular activities and, with more graduates than ever before and increasing competition for jobs upon leaving University, these are vital to your future prospects. We want to encourage you to make the very most of your University experience and to leave Lancaster as a well-rounded graduate. We have a wealth of opportunities to get involved in with initiatives such as work placements, volunteering, extracurricular courses, societies and sports. The Lancaster Award aims to encourage you to complete such activities, help you to pull them together in one place and then be recognised for your accomplishments. We want you to stand out from the crowd – the Lancaster Award will help you to do this. For more information see <http://www.lancs.ac.uk/careers/award/>.

3. Student Support

Lancaster has adopted a student-centred approach in which access to high quality support across a range of areas is provided by different agencies in a way which best meets each student's individual circumstances and needs. This is summarised in the [Student Support Policy](#). Academic support is provided in this School by lecturers and course convenors teaching specific modules for everything that is related directly with this module. Since SCC operates an open door policy academics should be available when required. Further, general academic support is also provided by the SCC Teaching Office (located in C12), and the Part I and Part II Tutors (listed above).

4. Academic Tutor and College Adviser

Further, each student has an assigned academic tutor in the School, who the student should meet at least once a term on a one-to-one basis at least once a term. You will also be able to contact your academic tutor to make an appointment at any mutually convenient time. This tutor will be interested in and knowledgeable about your progress and will be in a position to provide academic advice and support. A list of allocated academic tutors and how to contact him/her is available on the intranet. Where possible, your academic tutor will remain the same throughout your studies at Lancaster University.

In addition, during the first year of study, you will be assigned to a named College Advisor. That person can also provide advice and support to you on accessing these services, or upon any other issues you may need help with.

A Simple Solution

```
int aCount = 0; int eCount = 0;  
int iCount = 0; int oCount = 0;  
int uCount = 0;
```

for each letter:

```
    if (letter == 'a')  
        aCount = aCount + 1;  
    if (letter == 'e')  
        eCount = eCount + 1;  
    ...
```

What if ..?

- Now you find that you need to count up all the letters in the alphabet ...
 - So you need 26 integer variables ...
- And let's say you need to cope with upper and lower case ...
 - Now you need 52 integer variables ...

What if ..?

- And let's say you need to count numerals ..
 - Now you need 62 variables ...
- And let's add punctuation marks and spaces ...

```
int aCount = 0;
int bCount = 0;
int cCount = 0;
int dCount = 0;

int eCount = 0;
int fCount = 0;
int gCount = 0;
int hCount = 0;

int iCount = 0;
int jCount = 0;
int kCount = 0;
int lCount = 0;

int mCount = 0;
int nCount = 0;
int oCount = 0;
int pCount = 0;

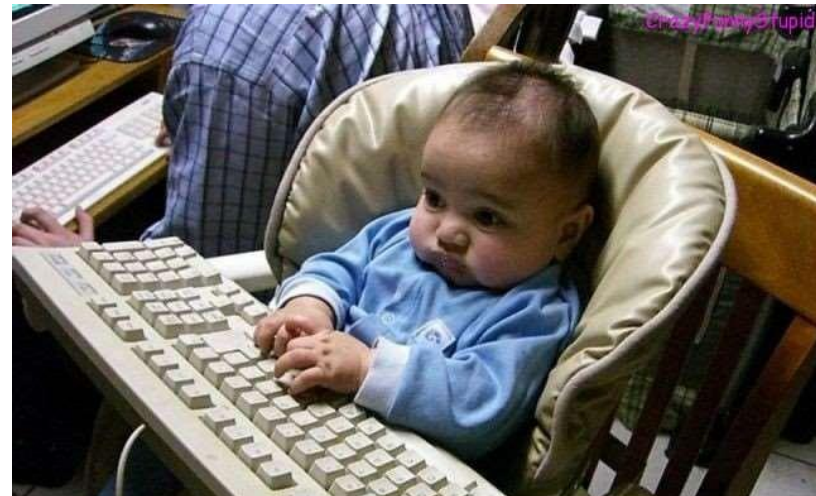
int qCount = 0;
int rCount = 0;
int sCount = 0;
int tCount = 0;

int uCount = 0;
int vCount = 0;
int wCount = 0;
int xCount = 0;

int yCount = 0;
int zCount = 0;

int n0Count = 0;
int n1Count = 0;

if (letter == 'a')
aCount = aCount + 1;
if (letter == 'b')
bCount = bCount + 1;
```



There must be a better way!

Let's consider counting 26 alphabetic letters for now

Arrays

- What we need is a data structure, i.e. **array**, which has **26** elements, each of which we need to be able to access (read/write) individually.
- Declare an array in C:

`int counts[26];`

data type variable name size

`Counts' Array

```
int counts[26];
```

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
indices	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
counts	0	0	0	1	1	0	0	1	0	0	0	3	0	0	2	0	0	1	0	0	0	0	1	0	0	0

Here is the array after we have counted up the characters in “hello world”

- To increment the count for ‘l’, we can write

```
counts[11] = counts[11] + 1;
```
- To print the value of the count for ‘h’ we can write

```
printf("%d\n", counts[7]);
```

Upper and Lower Bounds

counts	0	0	0	1	0	..	0	0	1	0	0	0	elements
	0	1	2	3	4	..	20	21	22	23	24	25	indices

- Upper bound (upb): index of the last element
- Lower bound (lwb): index of the first element
 - upb: 25; lwb: 0 for 'counts'

Linear (1-dimensional) Arrays

books	2	5	7	17
	0	1	2	3

- This linear array:
 - has the name “books”
 - has 4 elements
 - all its elements contain integers
 - each element has an index
 - here, the indices range from 0 to 3

Linear Arrays (2)

books	2	5	7	17
	0	1	2	3

- To define the shape of such an array, we must specify:
 - the type of values it can contain (`int`)
 - its name (`books`)
 - size of the array (`4`)

Declaring an array

- We can declare an array while also providing an initial set of values, or we can declare an “empty” array.

- In Java

```
int [] books = new int [4];  
int [] books = { 2, 5, 7, 17};
```

- In C

```
int books[4];  
int books[4] = {2, 5, 7, 17};  
int books[] = {2, 5, 7, 17};
```

Accessing individual elements

books	2	5	7	17
	0	1	2	3

- We use an index value to specify which element of the array we are dealing with.
- Once a single element of an array has been selected, we can use it in the same way as a simple variable.

```
int x; int y = 0;  
x = y + books[2];  
  
draw[2] = 9;  
z = 2;  
x = y + draw[z];
```

“Empty” Arrays

books	2	5	7	17
	0	1	2	3

- `int books[4];` creates an empty array for holding integers, but the integers are not yet initialised.
 - The array might contain random values without initialisation
 - Using uninitialised variables could be dangerous!

“Empty” Arrays

books	2	5	7	17
	0	1	2	3

- It may well be safest to initialise each element to what we consider to be a “safe” value

```
books[0] = 0; books[1] = 0;  
books[2] = 0; books[3] = 0;
```

- Problem: What if we had a hundred elements?



Arrays and For loops are meant for each other

FOR LOOPS

Example Loop Control Structures¹⁶

```
for (int i = 0; i < 4; i++) {  
    <loop_body>;  
}
```

‘i’ is the **loop control variable** (l.c.v)

l.c.v's range of values

- In Java and C, we specify what condition allows the loop to continue.

- `for (int i = 0; i < 4; i++)
 <loop_body>;`

- As long as 'i' is less than 4, keep going.
- We must also specify how the value of 'i' changes.

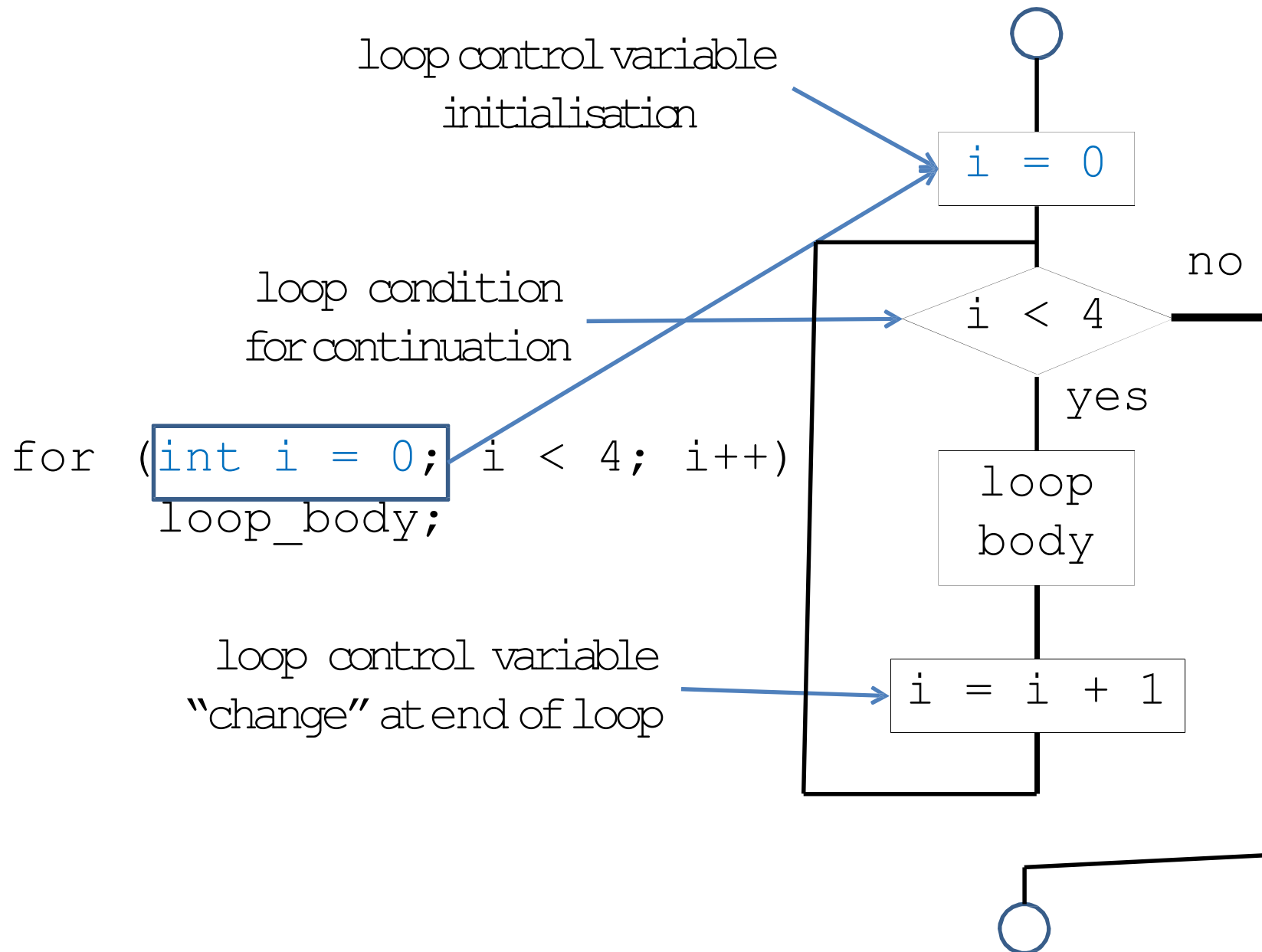
- `for (int i = 0; i < 4; i++)
 <loop_body>;`

- So we are incrementing the value of 'i' in steps of one.
 - `i++` is the short-hand of `i = i+1`

```
for (int i=0; i<4; i++)  
    <loop_body>;
```

Flow charts

- Here we are saying:
 - Initialise the value of 'i' to zero.
 - If $i < 4$, do the loop body.
 - do the loop body.
 - increment the value of 'i' by one.
 - If $i < 4$, go back and do the loop body again
 - otherwise, stop the loop and go onto the next instruction.
- We can visualise this in a flow chart.





ARRAY SCANNING : INITIALISING
ALL THE ELEMENTS

Playing at being the Computer

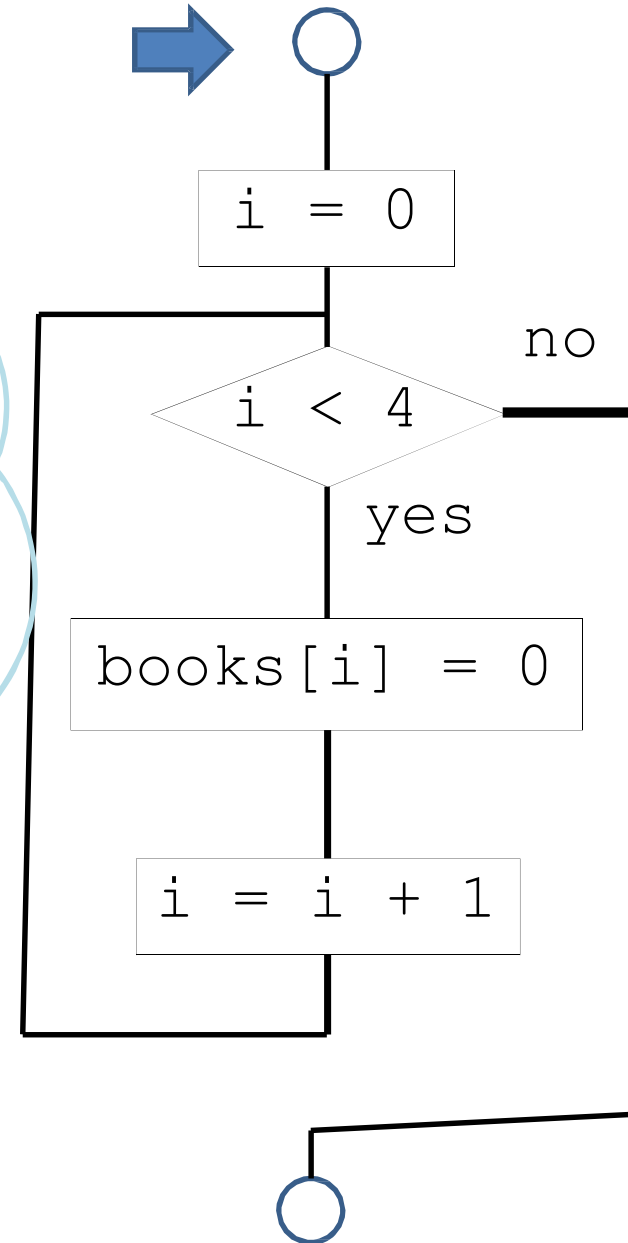
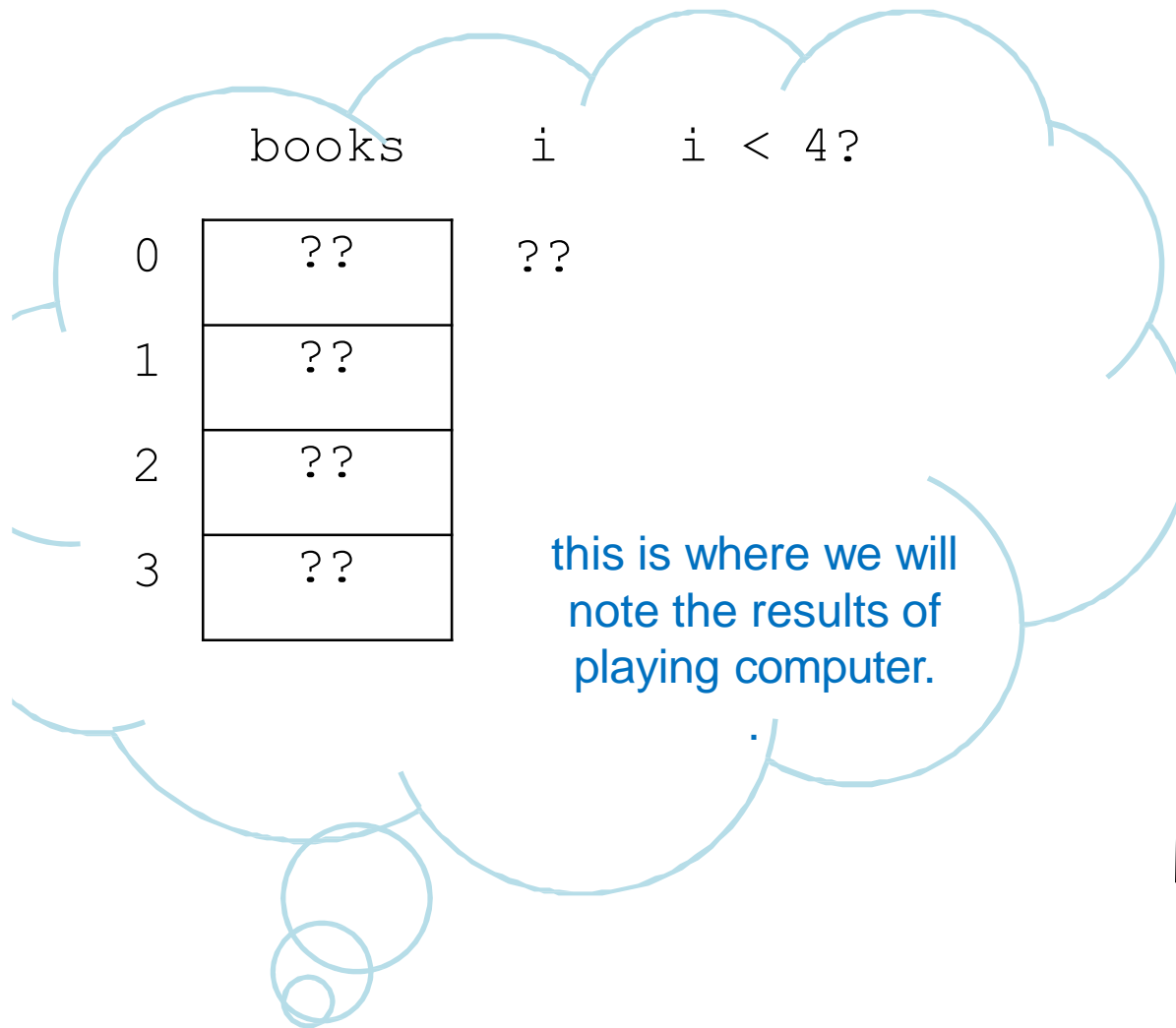
- or simply **Playing Computer**.
- This is a powerful debugging technique if there is something going wrong with your code.
- You must NOT “**skip over**” lines of code, as this defeats the purpose.
- We are playing computer here to illustrate how/why the example works.



Worked Example One (1)

25

```
for (int i = 0; i < 4; i++)  
    books[ i ] = 0;
```

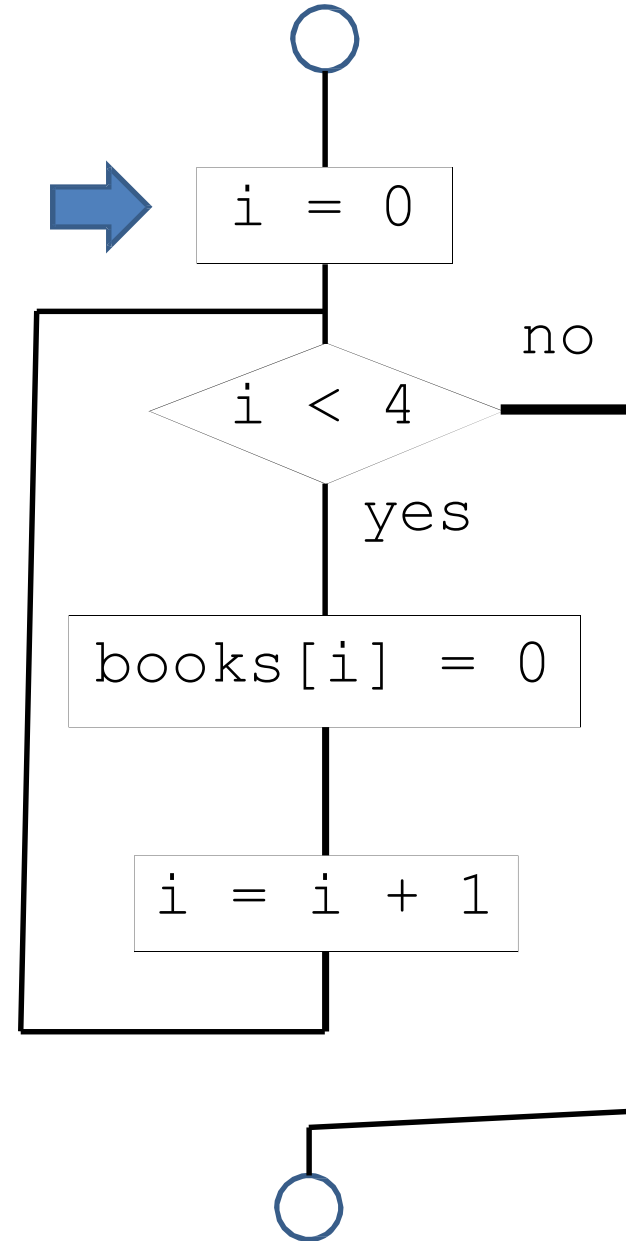
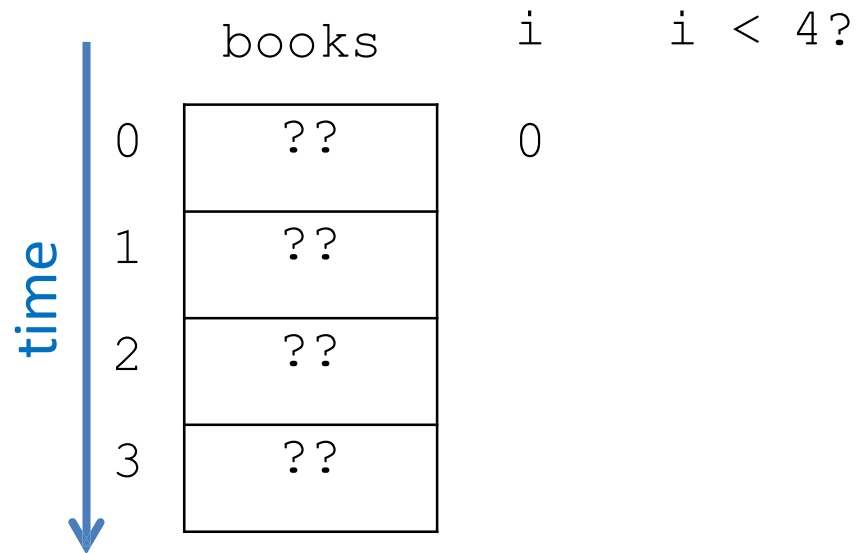




Worked Example One (2)

26

```
for (int i = 0; i < 4; i++)  
    books[ i ] = 0;
```

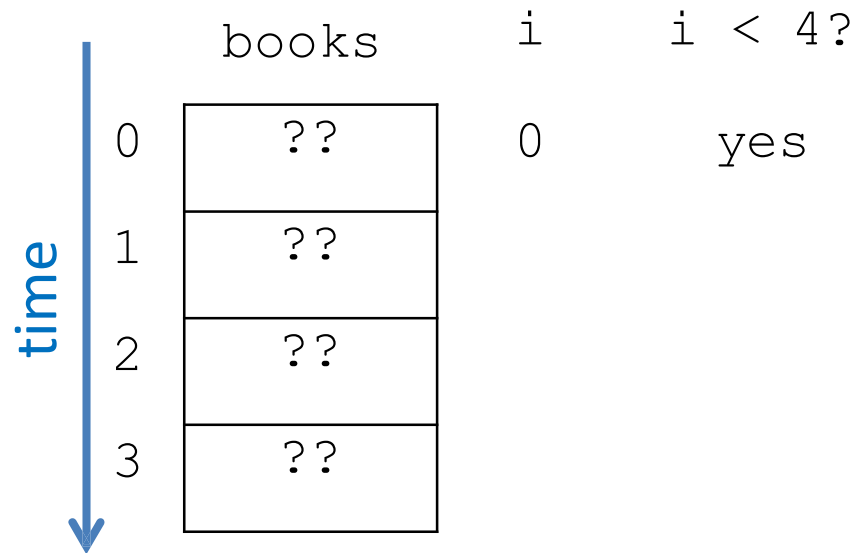




Worked Example One (3)

27

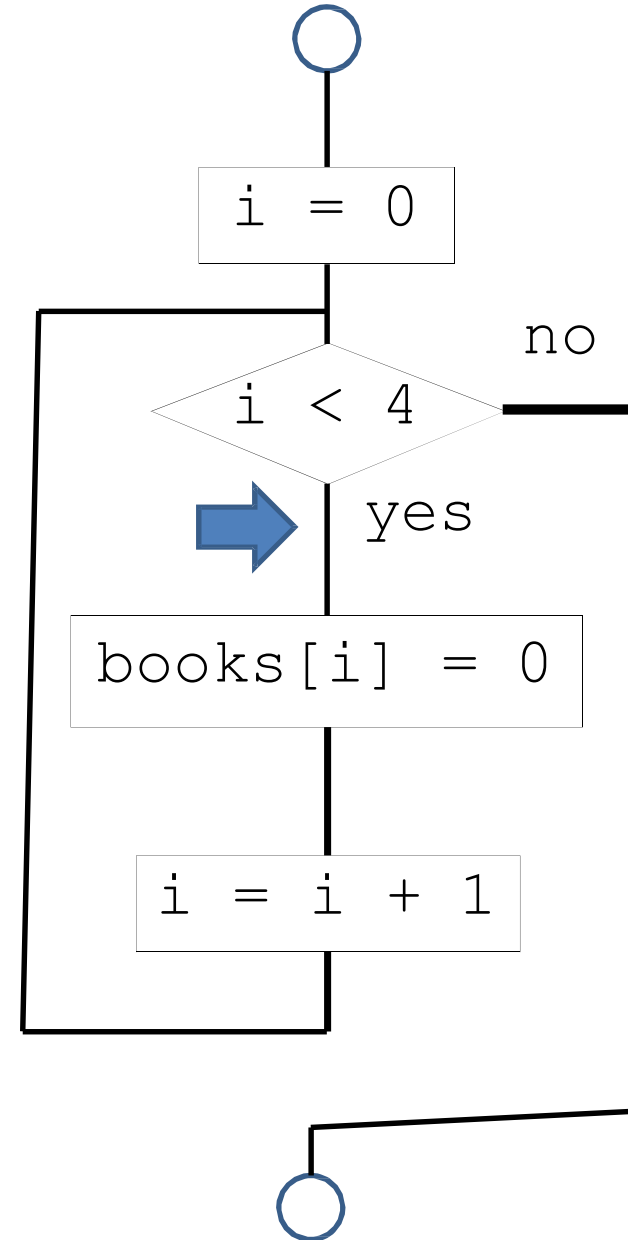
```
for (int i = 0; i < 4; i++)  
    books[ i ] = 0;
```



is $i < 4$?

i currently equal to 0

$0 < 4$? yes !

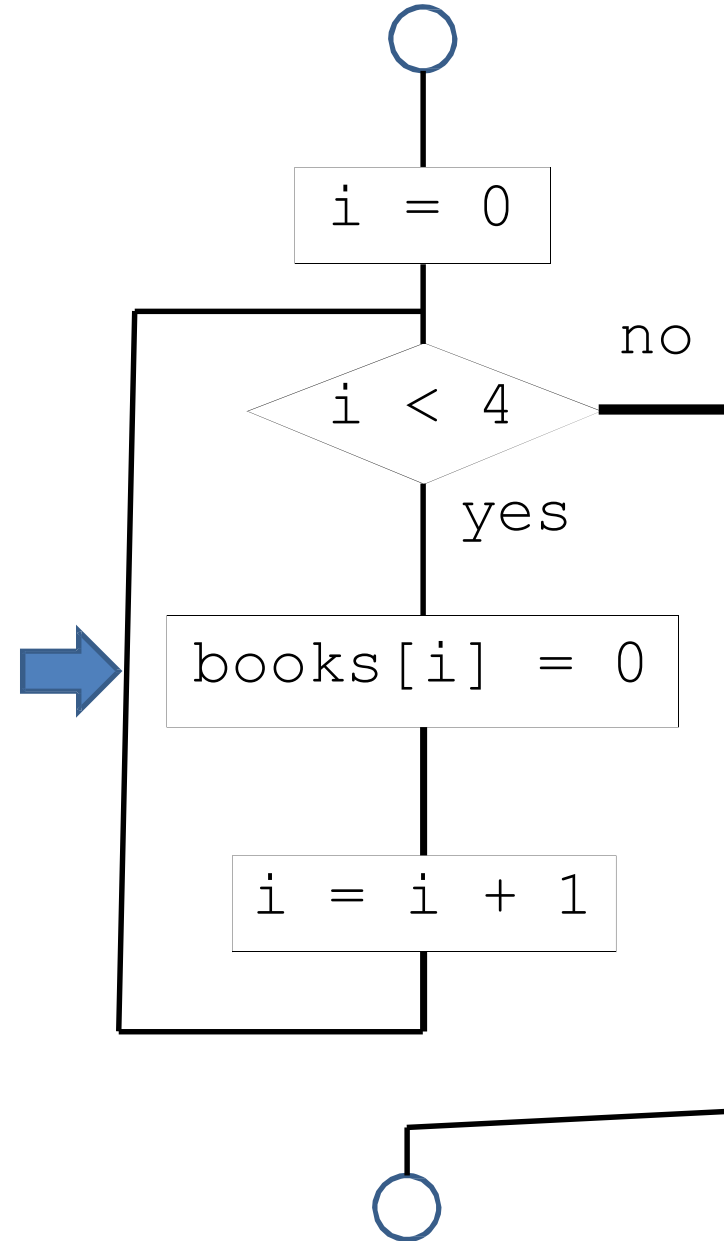
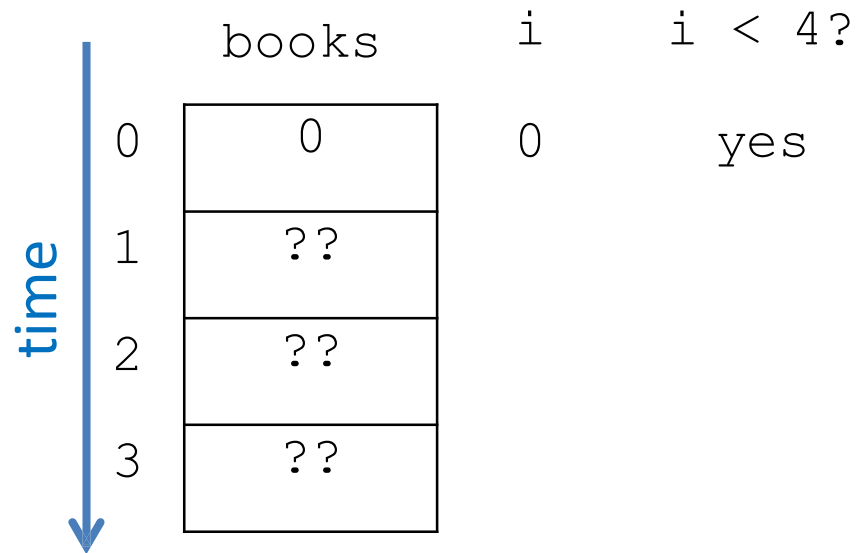




Worked Example One (4)

28

```
for (int i = 0; i < 4; i++)  
    books[ i ] = 0;
```





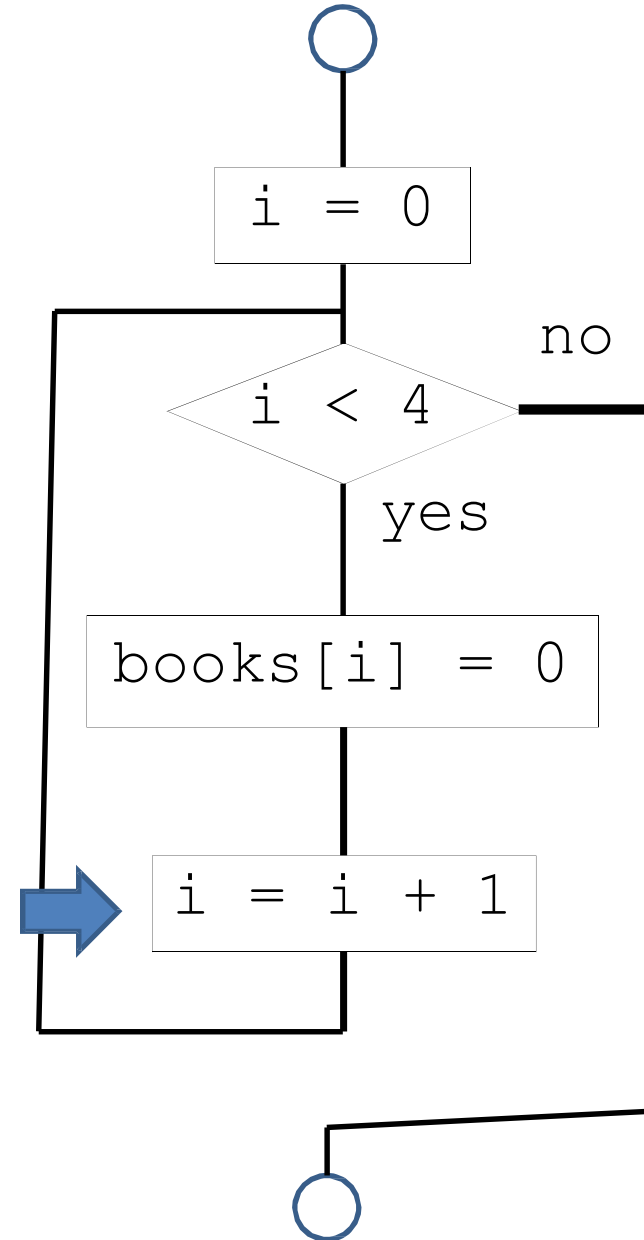
Worked Example One (5)

29

```
for (int i = 0; i < 4; i++)  
    books[ i ] = 0;
```

	books	i	i < 4?
0	0	0	yes
1	??	1	
2	??		
3	??		

time ↓





Worked Example One (6)

30

```
for (int i = 0; i < 4; i++)  
    books[ i ] = 0;
```

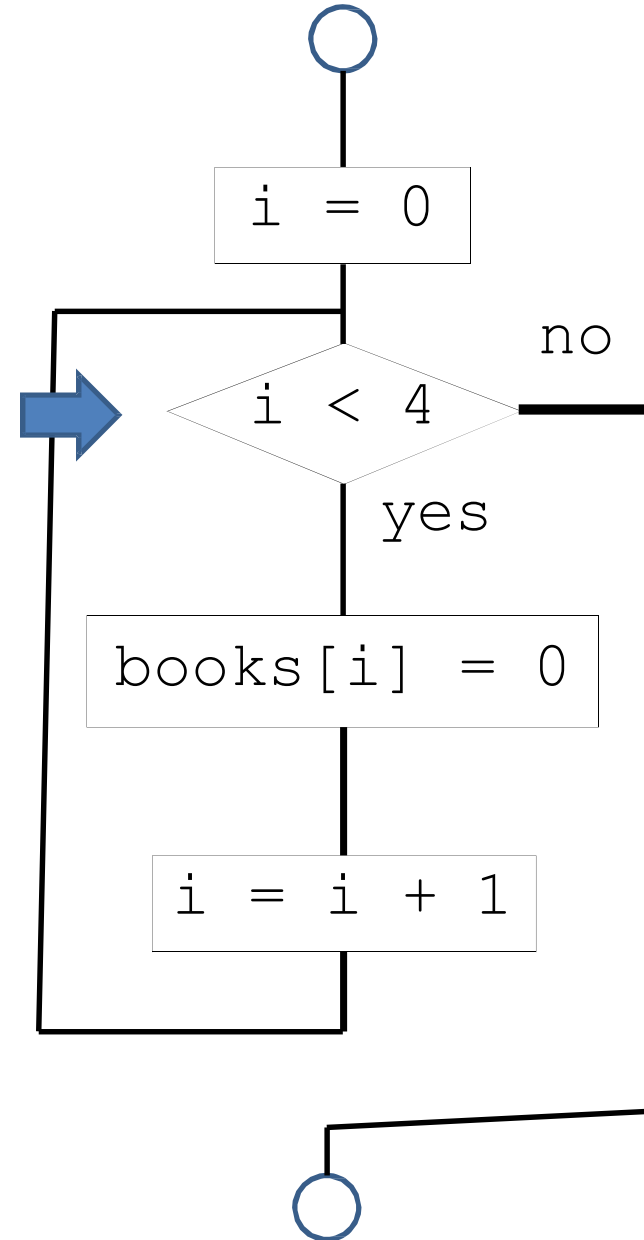
time ↓

	books	i	i < 4?
0	0	0	yes
1	??	1	yes
2	??		
3	??		

is $i < 4$?

i currently equals 1

$1 < 4$? yes !





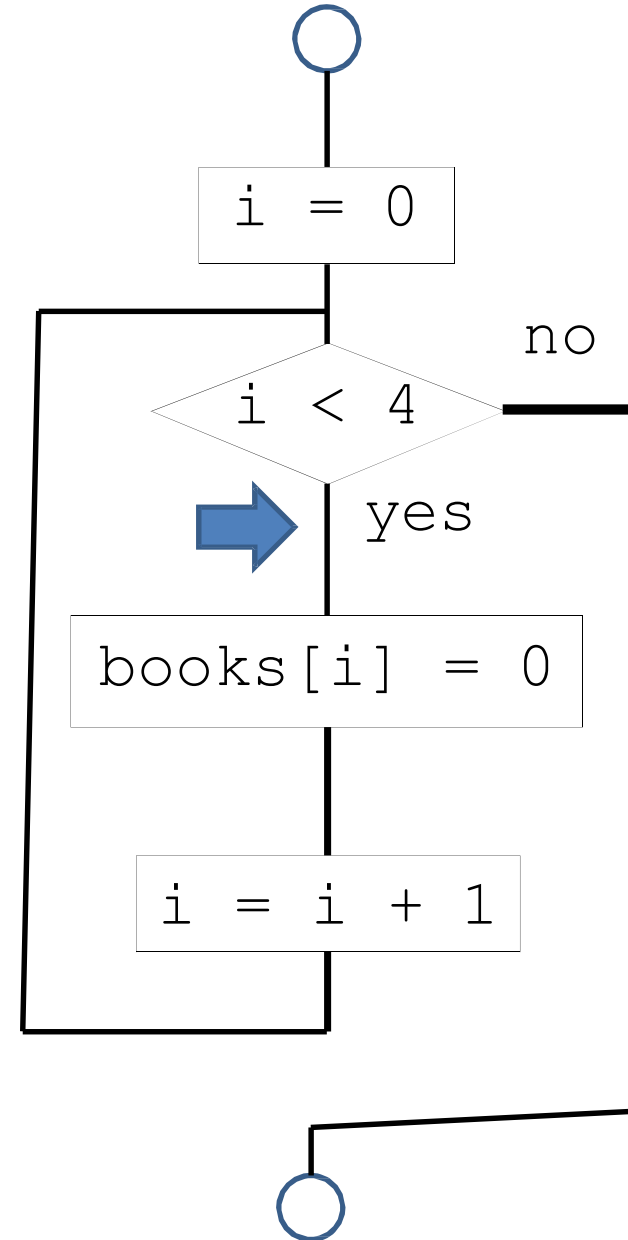
Worked Example One (7)

31

```
for (int i = 0; i < 4; i++)  
    books[ i ] = 0;
```

time ↓

	books	i	i < 4?
0	0	0	yes
1	??	1	yes
2	??		
3	??		





Worked Example One (8)

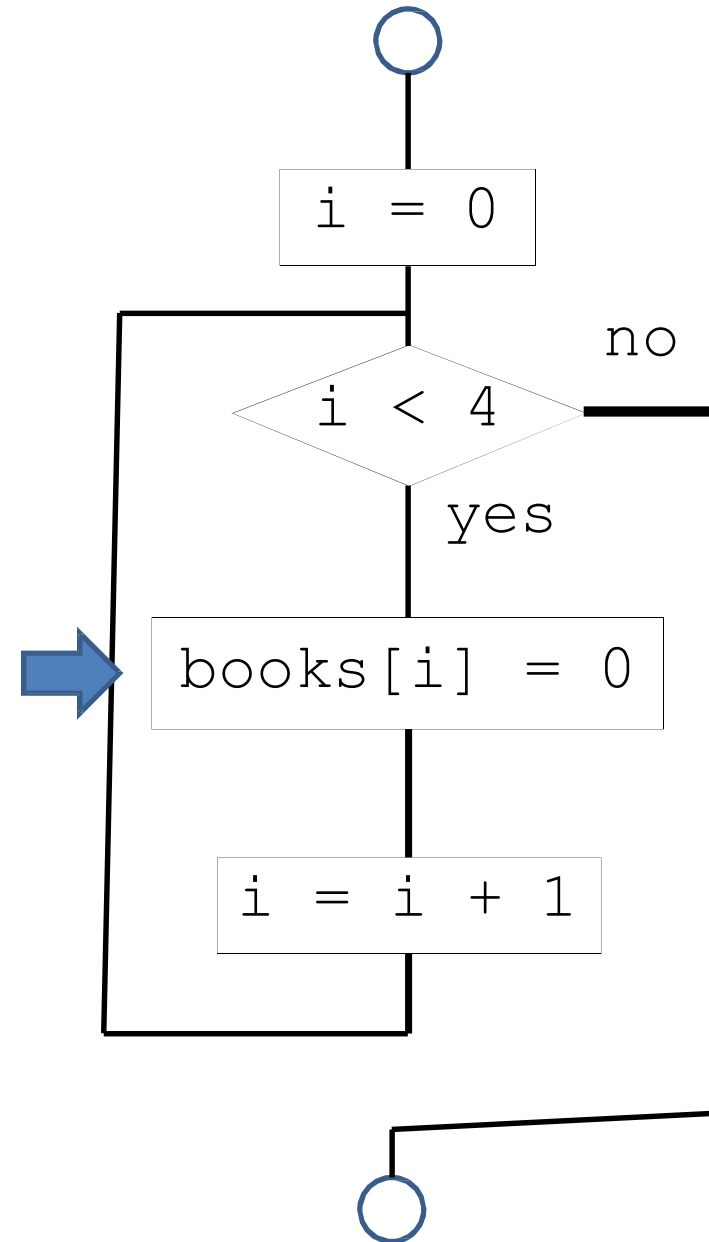
32

```
for (int i = 0; i < 4; i++)  
    books[ i ] = 0;
```

time ↓

	books	i	i < 4?
0	0	0	yes
1	0	1	yes
2	??		
3	??		

set the ith element of draw to 0
`draw [i] = 0`
i currently equals 1, so
`draw [1] = 0`





Worked Example One (9)

33

```
for (int i = 0; i < 4; i++)  
    books[ i ] = 0;
```

time ↓

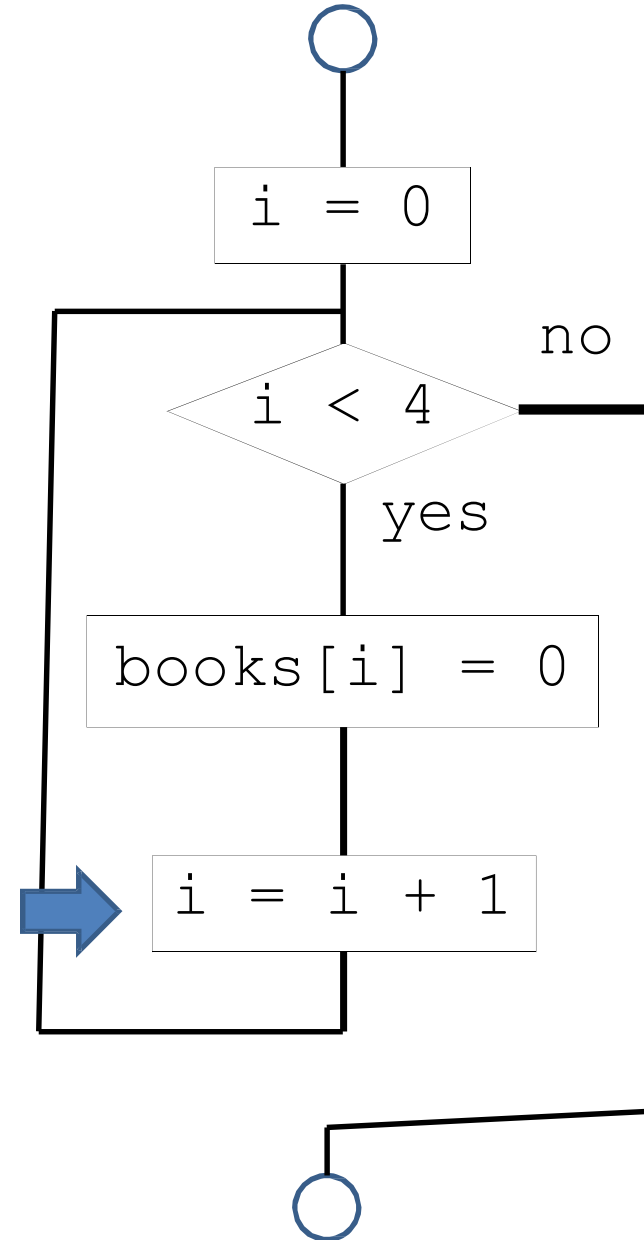
	books	i	i < 4?
0	0	0	yes
1	0	1	yes
2	??	2	
3	??		

increment i by 1 : $i = i + 1$

i currently equals 1, so

$i = 1 + 1$

$i = 2$





Worked Example One (10)

34

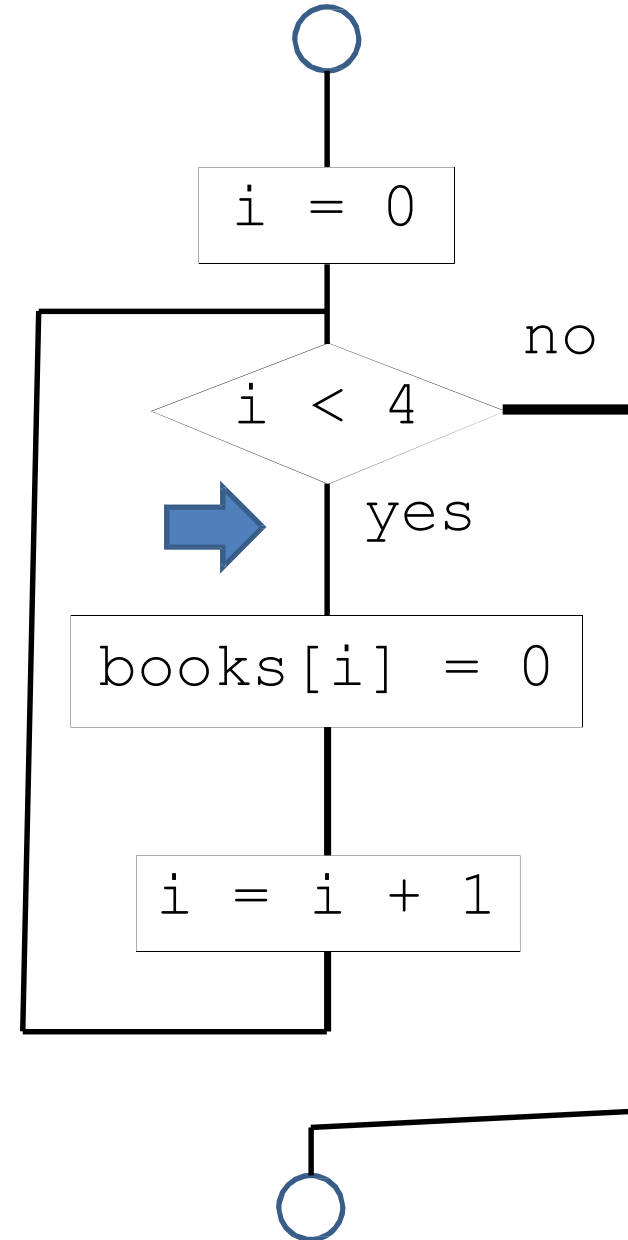
```
for (int i = 0; i < 4; i++)  
    books[ i ] = 0;
```

	books	i	i < 4?
0	0	0	yes
1	0	1	yes
2	??	2	yes
3	??		

is $i < 4$?

i currently equals 2

$2 < 4$? yes !





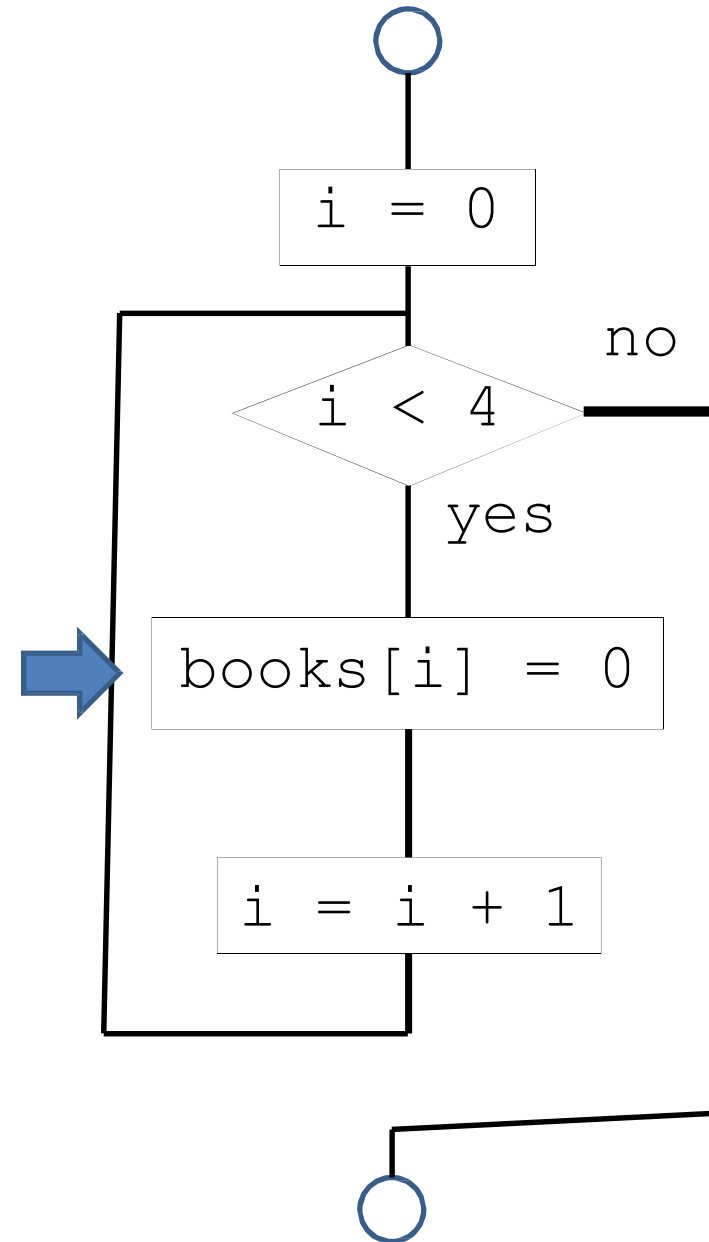
Worked Example One (11)

35

```
for (int i = 0; i < 4; i++)  
    books[ i ] = 0;
```

time ↓

	books	i	i < 4?
0	0	0	yes
1	0	1	yes
2	0	2	yes
3	??		



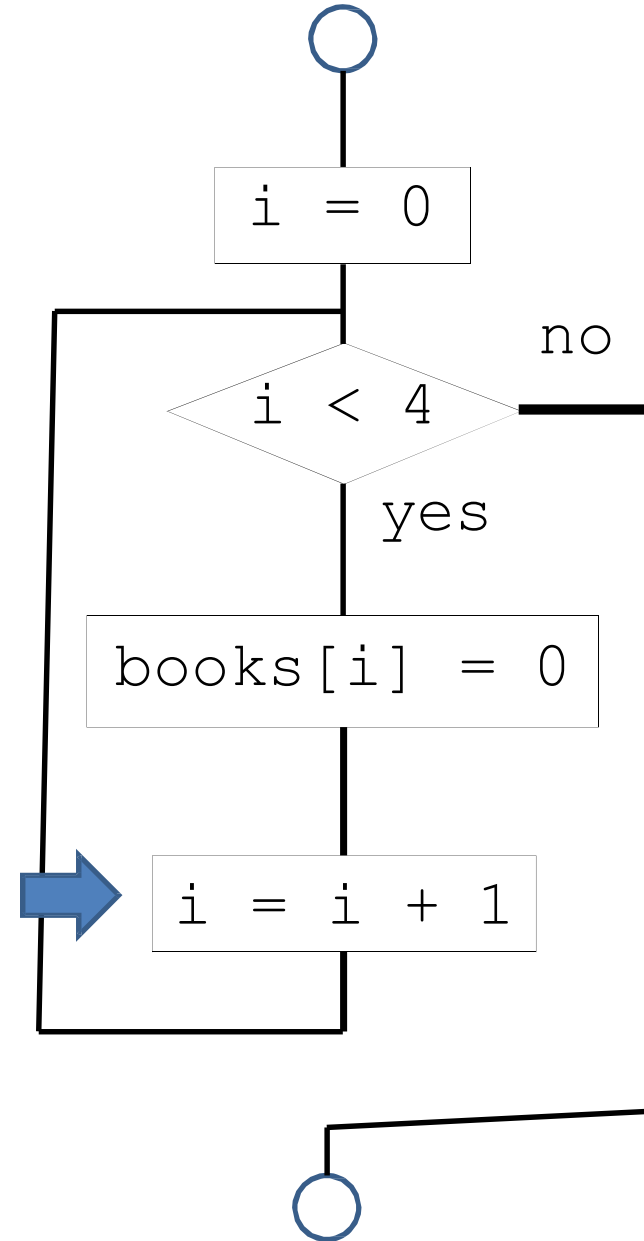


Worked Example One (12)

36

```
for (int i = 0; i < 4; i++)  
    books[ i ] = 0;
```

	books	i	i < 4?
0	0	0	yes
1	0	1	yes
2	0	2	yes
3	??	3	





Worked Example One (13)

37

```
for (int i = 0; i < 4; i++)  
    books[ i ] = 0;
```

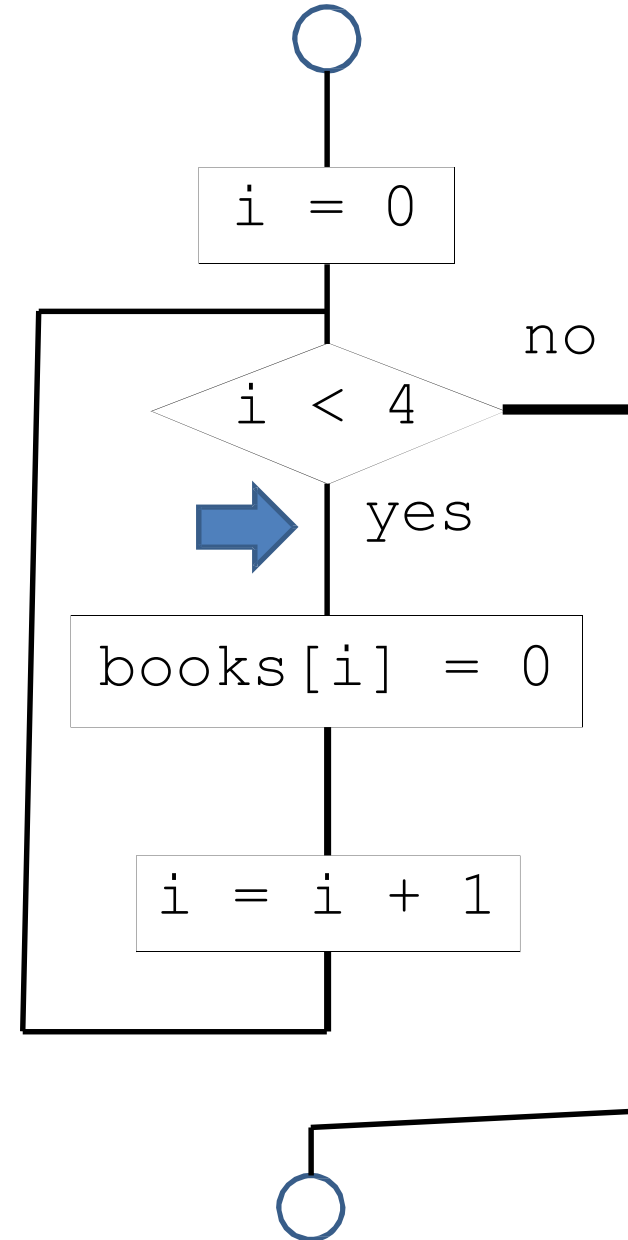
	books	i	i < 4?
0	0	0	yes
1	0	1	yes
2	0	2	yes
3	??	3	yes

time ↓

is $i < 4$?

i currently equal to 3, so

3 < 4 ? yes !





Worked Example One (14)

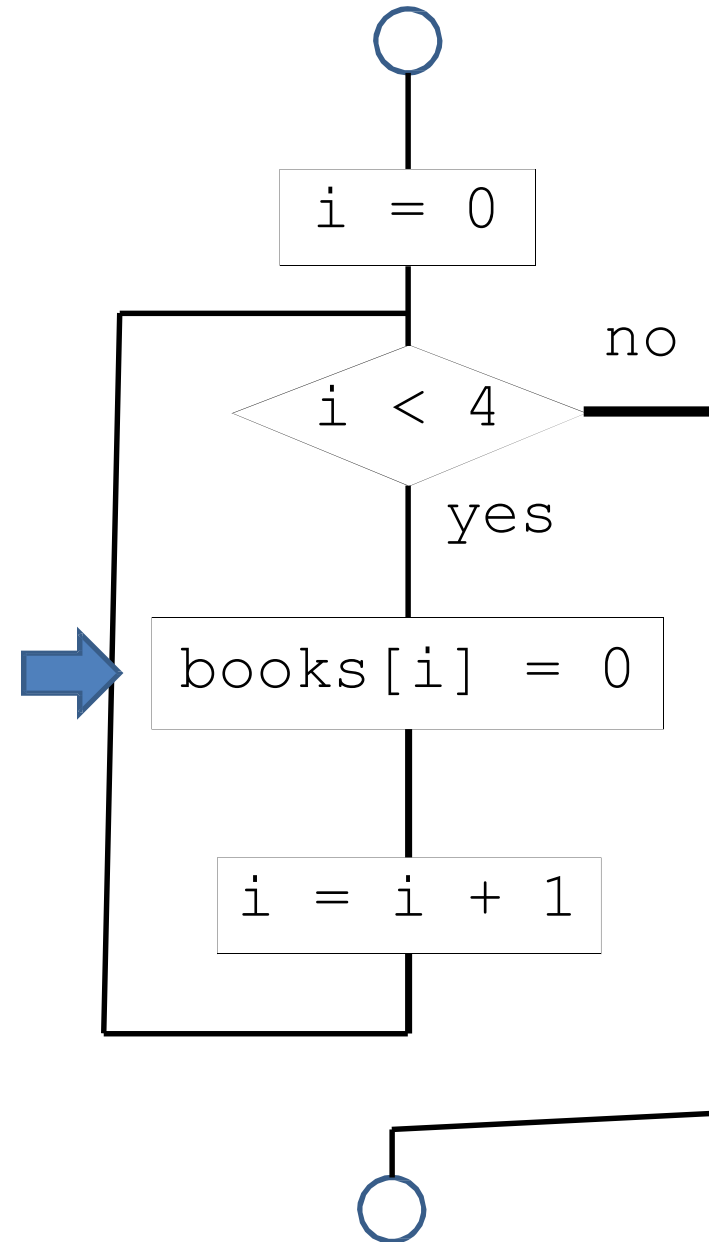
38

```
for (int i = 0; i < 4; i++)  
    books[ i ] = 0;
```

time ↓

	books	i	i < 4?
0	0	0	yes
1	0	1	yes
2	0	2	yes
3	0	3	yes

set the ith element of draw to 0
draw [i] = 0
i currently equals 3, so
draw [3] = 0





Worked Example One (15)

39

```
for (int i = 0; i < 4; i++)  
    books[ i ] = 0;
```

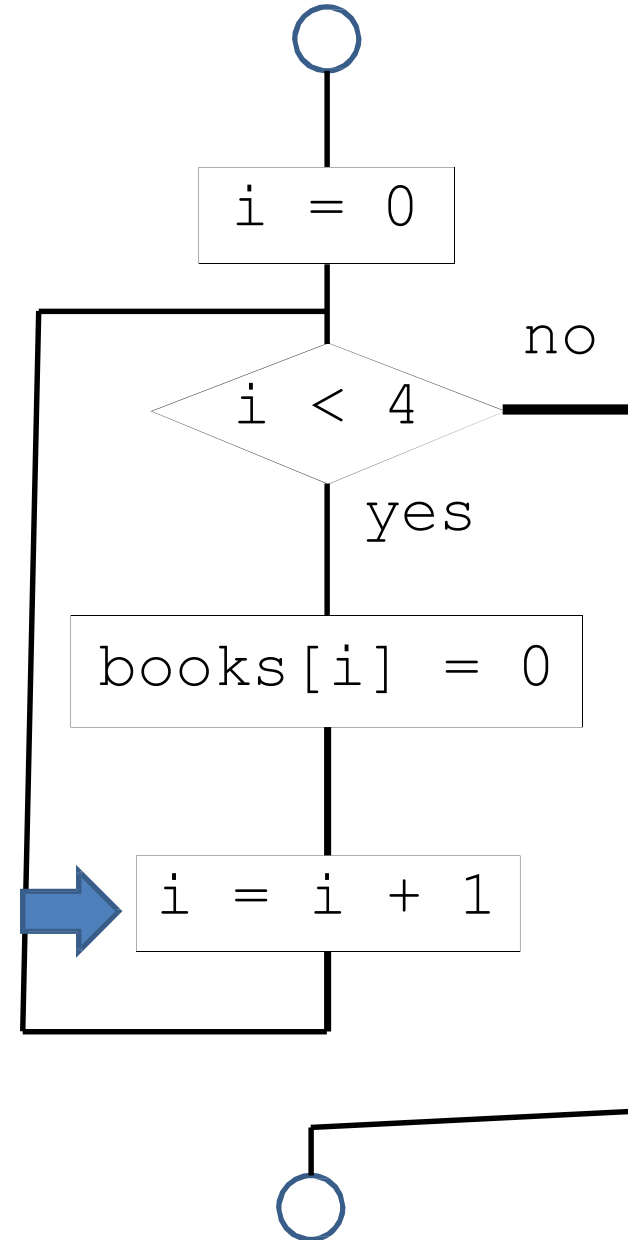
	books	i	i < 4?
0	0	0	yes
1	0	1	yes
2	0	2	yes
3	0	3	yes
		4	

increment i by 1 : $i = i + 1$

i currently equals 3, so

$i = 3 + 1$

$i = 4$





Worked Example One (16)

40

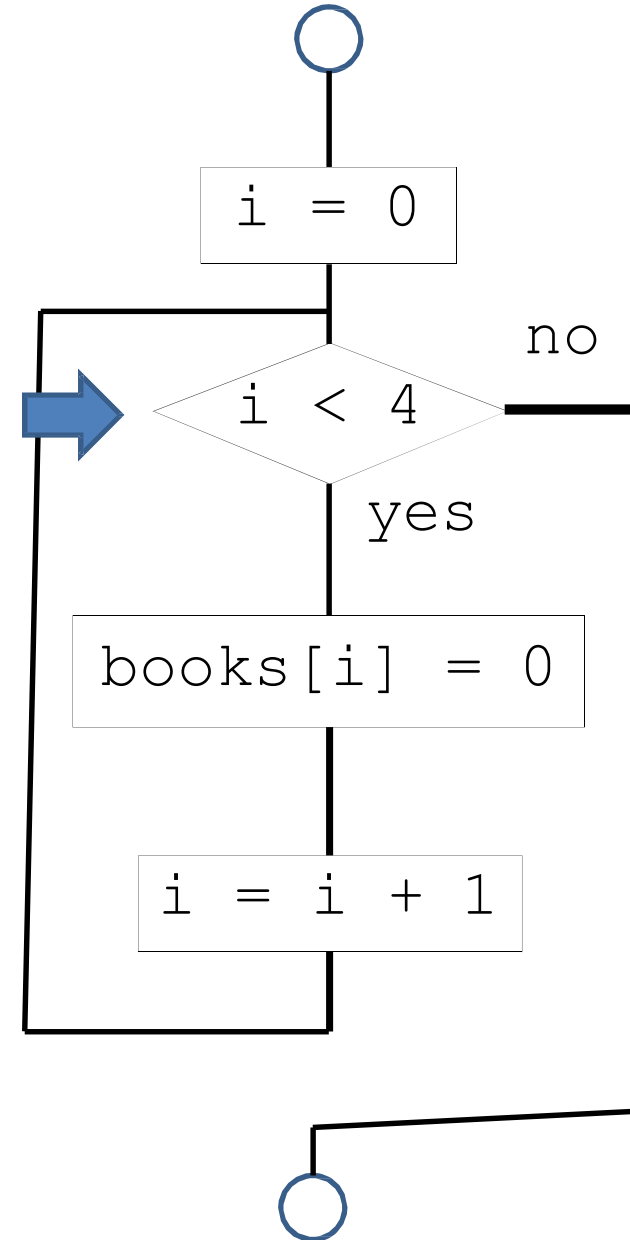
```
for (int i = 0; i < 4; i++)  
    books[ i ] = 0;
```

	books	i	i < 4?
0	0	0	yes
1	0	1	yes
2	0	2	yes
3	0	3	yes
		4	no

is $i < 4$?

i currently equals 4, so

4 < 4 ? no!





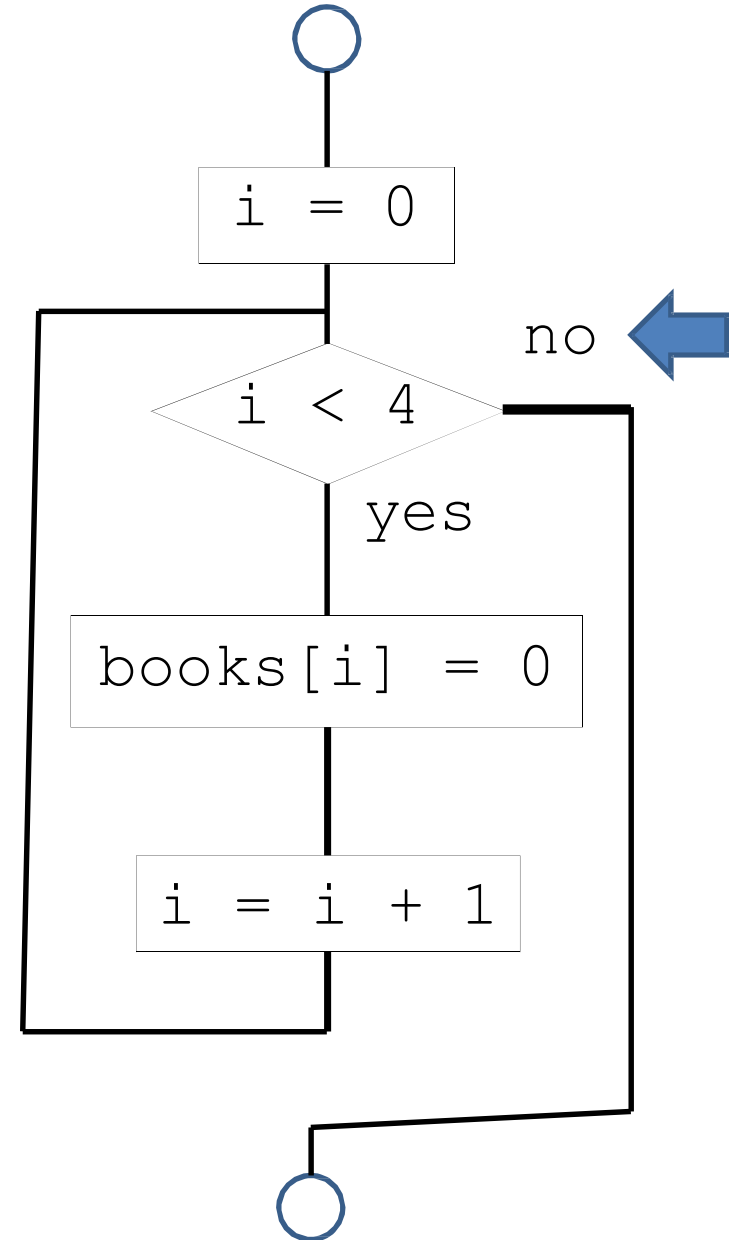
Worked Example One (17)

41

```
for (int i = 0; i < 4; i++)  
    books[ i ] = 0;
```

	books	i	i < 6?
0	0	0	yes
1	0	1	yes
2	0	2	yes
3	0	3	yes
		4	no

time
↓





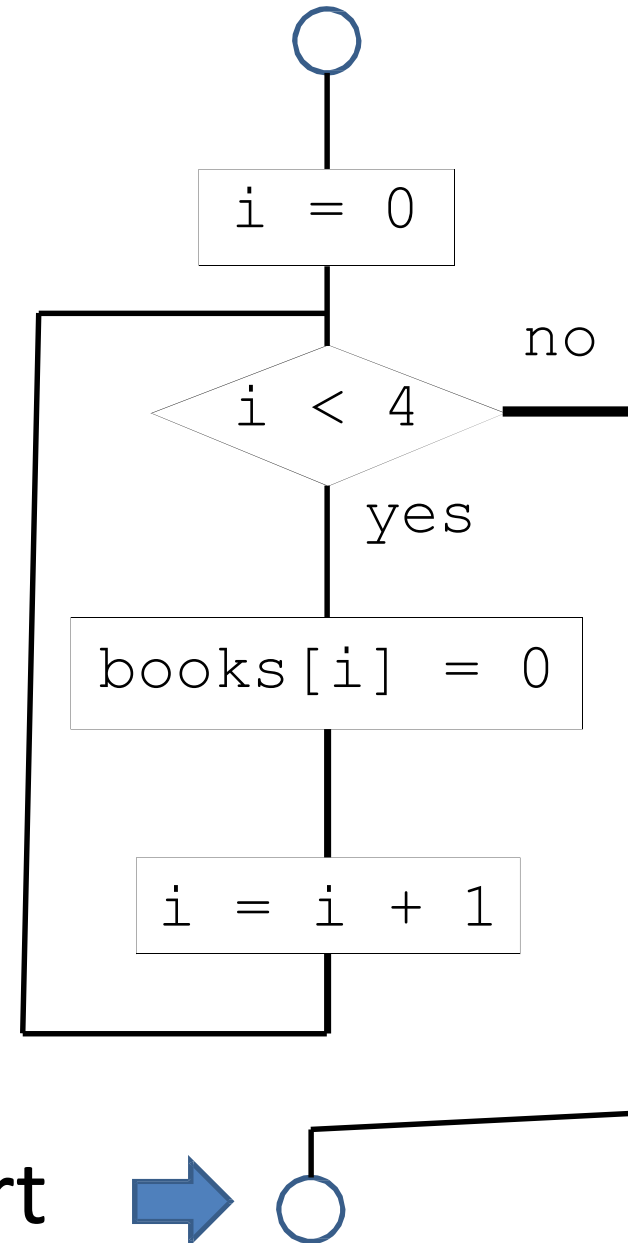
Worked Example One (18)

42

```
for (int i = 0; i < 4; i++)  
    books[ i ] = 0;
```

	books	i	i < 4?
0	0	0	yes
1	0	1	yes
2	0	2	yes
3	0	3	yes
		4	no

time
↓



End of flow chart



Exercise Point

	draw	i
0	??	
1	??	
2	??	
3	??	

- Draw the flowchart for the `for` loop.
- Follow the execution and fill in the table on the left.
- What does the `draw` array contain at the end of the loop?

```
for (int i = 0; i < 4; i++)  
    books[ i ] = i * 2;
```


Indexing (1)

- Access to array elements is via an *index*
 - Arrays are random-access data structures
- e.g. `x = books[3];`
- Random access makes it efficient to access an element of the array if the index is known.
- WHY?

Indexing (2)

- Indexing is efficient because:
 - All the cells in an array are stored contiguously in memory
 - the memory address allocated to the first cell is known as the array's **base address**
 - The array is homogeneous
 - everything is of the same type
 - The amount of memory needed for each cell is known
 - defined by the type

Indexing (3)

- Assuming each integer occupies 4 bytes
- The base address is 2000
- The address of the `books[i]` is

$2000 + i * 4$ (array mapping function)

	books
(base address) 2000	2
2004	5
2008	7
2012	17



ARRAY SCANNING & LATE
TERMINATION

Comparing two arrays

- Consider a lottery game where we pick four numbers.
- The array “mine” contains the four numbers on my ticket, while “draw” contains the actual four numbers drawn.

– mine : { 2, 5, 9, 17 }
– draw : { 2, 5, 7, 17 }

- This Java code decides if I have won:

```
boolean win = true;  
for (int n = 0; n < 4; n++)  
{  
    if ( mine[n] != draw[n] )  
    {  
        win = false;  
    }  
}
```



Array Scanning

- Scan through “mine” and “draw” 'in parallel'
 - If a pair is not equal (!=) set win to false
 - If this never happens, win remains true after looping.... Congratulations!

```
boolean win = true;
for (int n = 0; n < 4; n++)
{
    if (mine[n] != draw[n] )
    {
        win = false;
    }
}
```

Late Termination

```
for (int n = 0; n < 4; n++)  
{  
    if (mine[n] != draw[n] )  
{  
        win = false;  
    }  
}
```

```
mine : { 2, 5, 9, 17}  
draw : { 2, 5, 7, 17}
```

The code iterates to the end of the loop even though a difference was found on the 3rd items

What if each array has 1 million items and the first items are different?

Better Code

```
for (int n = 0; n < 4; n++)  
{  
    if (mine[n] != draw[n] )  
    {  
        win = false;  
        break;  
    }  
}
```


Late Termination is NOT Always⁵² Avoidable

- e.g. our program to search for the **largest** element in an **unordered array** will need to loop over the entire array ...
- ... because we have no way of knowing that the current item is the largest element, we have to look at all the others to check.

Code Efficiency

- Even as CPUs get ever-faster, efficiency matters.
 - Nothing comes for free
 - Unnecessarily inefficient code is sloppy
- Late loop termination is only one source of inefficiency

Real programmers count from 0.

THE END

Unit 3: Multi-Dimensional Arrays

SCC120 Introduction to Data
Structures

Jidong Yuan

yuanjd@bjtu.edu.cn

Scenario



- In the Premier Football League, we want to keep statistics that show how many times team A has beaten team B. (Draws don't count).
- How could we do this on paper?

Name (20 teams)
Arsenal
Aston Villa
Blackburn Rovers
Bolton Wanderers
Chelsea
Everton
Leicester City
Liverpool
Manchester City
Manchester United
Newcastle United
Norwich City
Queens Park Rangers
Stoke City
Sunderland
Swansea City
Tottenham Hotspur
West Bromwich Albion
Wigan Athletic
Wolverhampton Wanderers

- We could make a list of {teamA, teamB, number of times teamA has beaten teamB} triples.
 - {Arsenal, Norwich City, 4}
 - {Arsenal, Stoke City, 6}
 - {Norwich City, Arsenal, 1}
- We could store this in a table.

Team A	Team B	Victories for A
Arsenal	Norwich City	4
Arsenal	Stoke City	6
Norwich City	Arsenal	1

- Quite difficult to use the table.
- Would be very big (20 teams * 19 rows, 380 rows), difficult to find the information
- Or to do calculations with i.e. how many victories and defeats has Norwich City had?

Team A	Team B	Victories for A
Arsenal	Norwich City	4
Arsenal	Liverpool	6
Norwich City	Arsenal	1
Newcastle United	Norwich City	5

Name	Abbr.	Number
Arsenal	AR	0
Aston Villa	AV	1
Blackburn Rovers	BR	2
Bolton Wanderers	BW	3
Chelsea	CH	4
Everton	EV	5
Leicester City	FU	6
Liverpool	LI	7
Manchester City	MC	8
Manchester United	MU	9
Newcastle United	NU	10
Norwich City	NC	11
Queens Park Rangers	QP	12
Stoke City	SC	13
Sunderland	SU	14
Swansea City	SW	15
Tottenham Hotspur	TH	16
West Bromwich Albion	WB	17
Wigan Athletic	WA	18
Wolverhampton Wanderers	WW	19

Encoding

- So that we don't have to use the full names of the teams on the slides, here are some encodings.

- Showing 13 of the teams.

A Matrix

	AR	AV	BR	BW	CH	EV	LC	LI	MC	MU	NU	NC	QP	...
AR	--							6				4		
AV		--												
BR			--											
BW				--										
CH					--									
EV						--								
LC							--							
LI								--						
MC									--					
MU										--				
NU											--			
NC	1											--		
QP													--	
...														--

Now to find out a team's victories, you look for the appropriate row.

To find a team's defeats, you look for the appropriate column.

To calculate total victories, add up the numbers in the row.

To calculate total defeats, add up the numbers in the column.

Representing our solutions in software⁷

- All the contents of the table are integers
 - We label each team with an integer
 - What we require now is a two-dimensional array where each element is a simple integer.

2-D Arrays

- Elements of Two-dimensional (2D) arrays are accessed by two indexes (subscripts), one for the row and one for the column.

row **col**

rating[0][2] = 2
rating[1][3] = 8

rating

movie (second index)

	0	1	2	3
reviewer (first index) 0	4	6	2	5
1	7	9	4	8
2	6	9	3	7



2-D Arrays

rating

- To create this array in C:

```
int rating[3][4];
```

- To create this empty array in Java:

```
int [][] rating = new int[3][4];
```

Another Example

```
char c[2][3];
```

- ‘c’ has 2 rows, indexed 0 to 1, and 3 columns, indexed 0 to 2.
- Each element is of type ‘char’.
- Just as with linear arrays, we need to be able to access each individual element.

c	0	1	2
0			
1			

Array Example (cont)

```
char c[2][3];
```

- Indices of 2-D array 'c'.

c	0	1	2
0	[0][0]	[0][1]	[0][2]
1	[1][0]	[1][1]	[1][2]

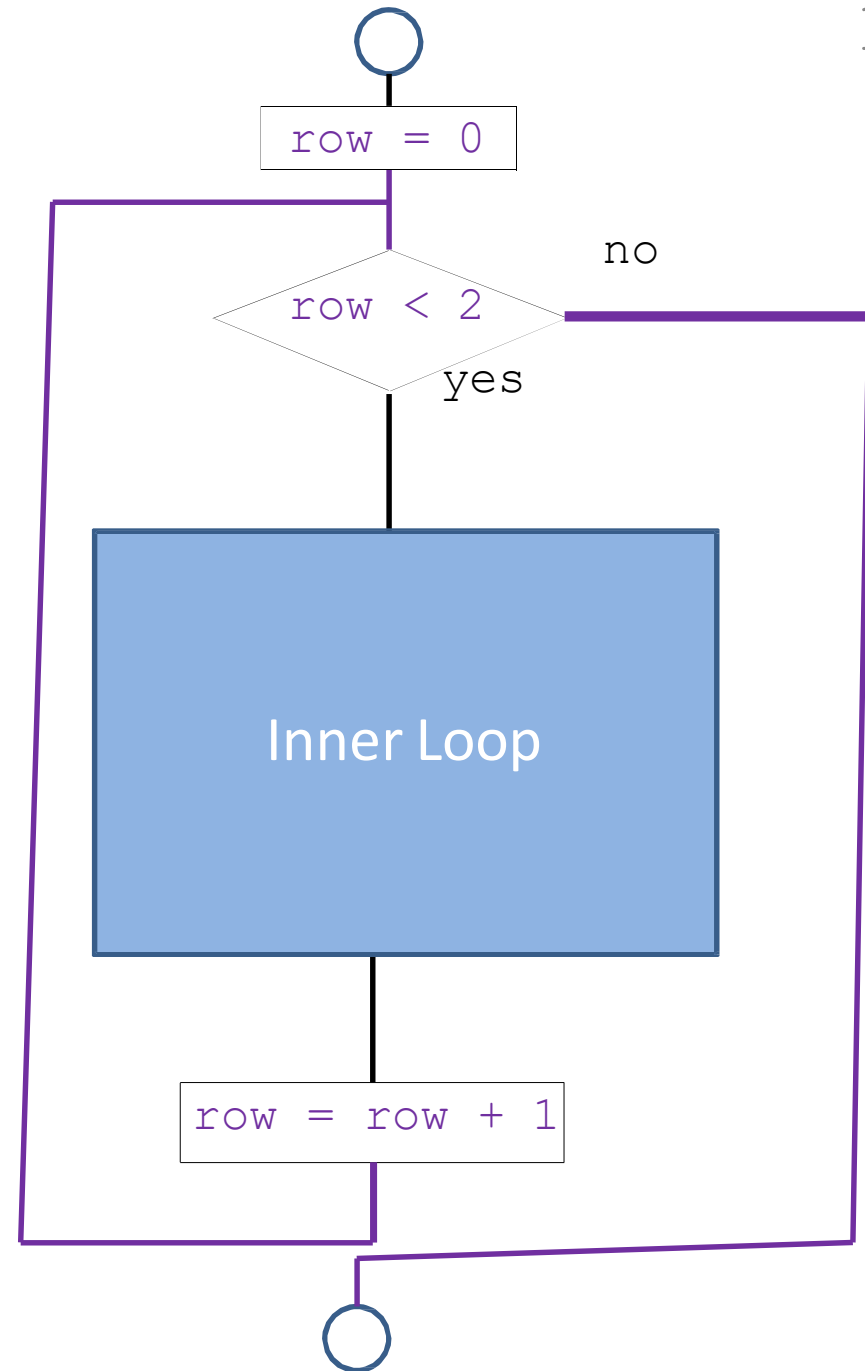
- How can we generate each pair of indices?
 - Use a “double loop”, or a loop within a loop.

```
for (int row = 0; row < 2; row++)  
    for (int col = 0; col < 3; col++)  
        print row, col
```

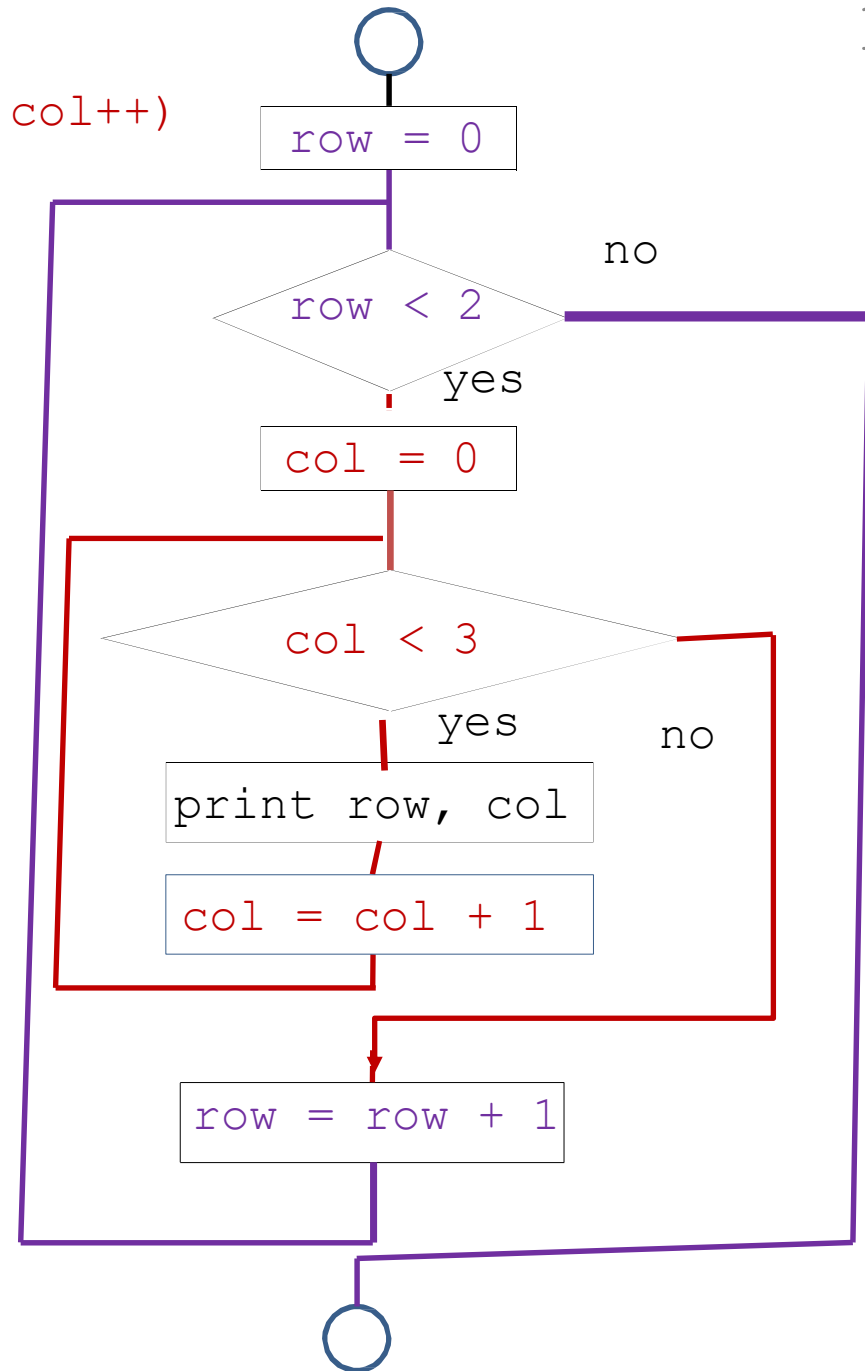
12

```
char c[2][3];
```

Outer Loop



```
for (int row = 0; row < 2; row++)  
    for (int col = 0; col < 3; col++)  
        print row, col
```



Inner Loop

Quiz: Average Rating

		movie			
		0	1	2	3
reviewer	0	4	6	2	5
	1	7	9	4	8
	2	6	9	3	7



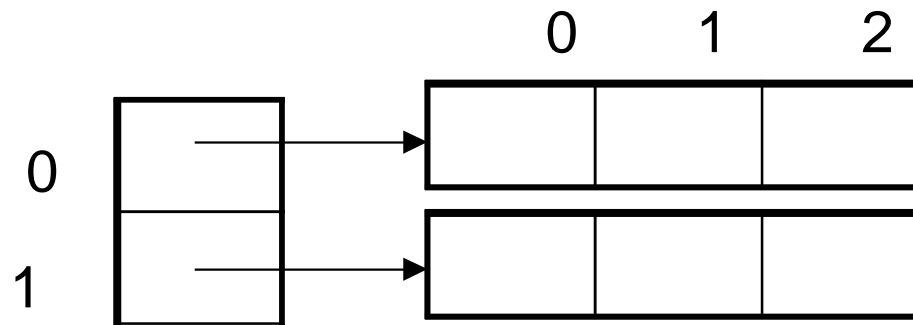
```
float avg_rating[4];
float sum;

for (int col=0; col<4; col++) {
    sum = 0.0;
    for (int row=0; row<3; row++) {
        sum = sum + rating[row][col];
    }
    avg_rating[col] = sum / 3;
}
```

2D Array Implementation in Java/C¹⁵

- A 2-D array is a 1-D array of (references to) 1-D arrays.

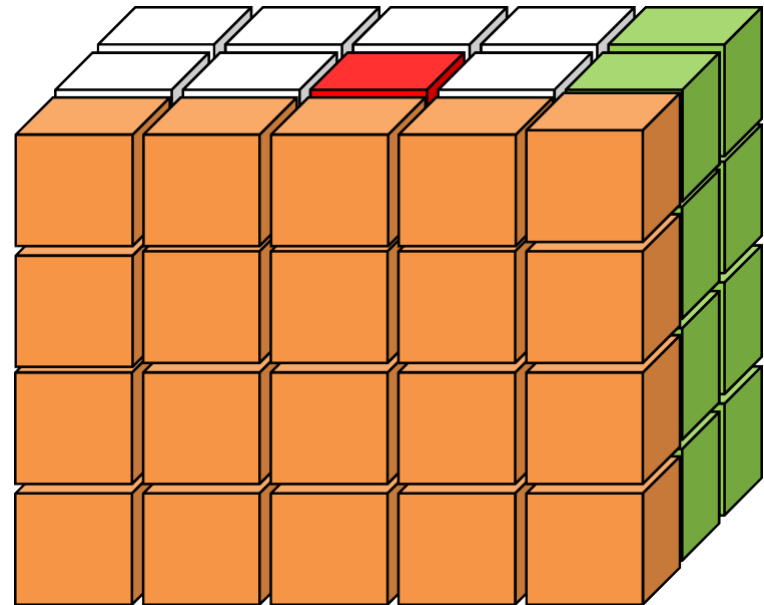
```
char c[2][3];
```



3-D and N-D arrays

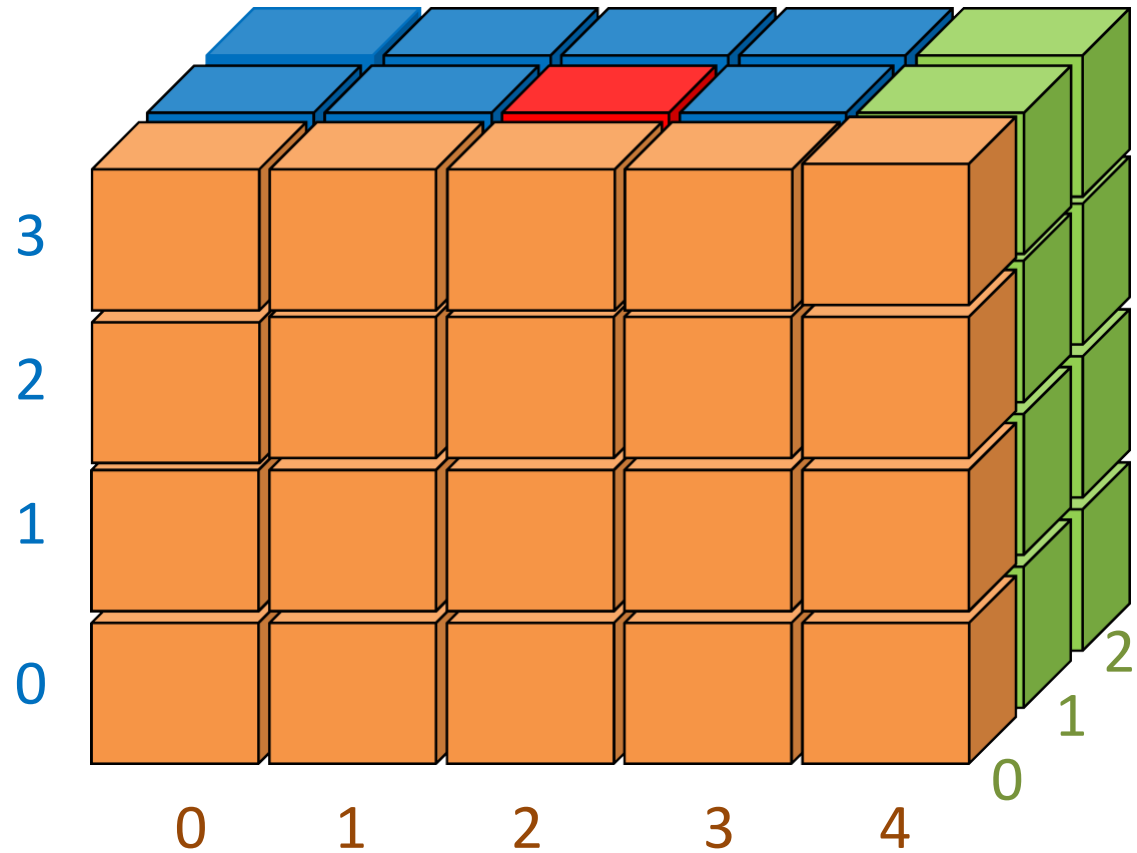
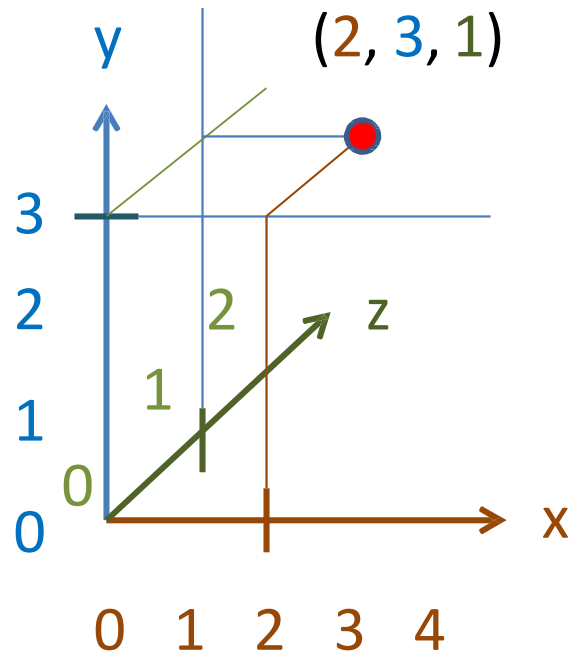
- Questions:
 - How many elements in the box array?
 - If it takes a “double” loop to access every element of a 2-D array, how many loops for a 3-D array?
- Theoretically there is no limit to the number of dimensions for an array.

```
enum colour box[5][4][3];
```



```
box [2][3][1] = red;
```

X, Y, Z axis



```
enum colour box[5][4][3];
```

Revisit: Our Football Problem

[illegible]

Our Football Problem

```
#define NUMBER_OF_TEAMS    20
int stats [NUMBER_OF_TEAMS][NUMBER_OF_TEAMS];

int getStats(int team_a, int team_b)
{
    return stats[team_a][team_b];
}
```

- Here we set up the 2-D array to be used as our data structure.
- We also provide an “access routine” – `getStats()`

```
void initStats()  
{  
    // initialise stats array  
    for (int i = 0; i < NUMBER_OF_TEAMS; i++)  
        for (int j = 0; j < NUMBER_OF_TEAMS; j++){  
            stats[i][j] = 0;  
        }  
  
    stats[18][2] = 4;  
    stats[18][9] = 6;  
    stats[2][4] = 1;  
}
```

- This code uses a “double loop” to ensure that every element of the “stats” array is set to zero.

```
void printStats()
{
    for (int i = 0; i < NUMBER_OF_TEAMS; i++)
        for (int j = 0; j < NUMBER_OF_TEAMS; j++)
        {
            if (stats[i][j] != 0)
                printf("%d, %d = %d\n", i, j, stats[i][j]);
        }
}
```

- This code also uses a double loop to examine each element of the “stats” array.
- If the value of the element is not zero, it prints out the values of the two indices (i and j) and the value of the element.
- Remember, **i** and **j** map onto two football teams.


```
void main (int argc, char *argv[])
{
    printf("Football Stats package testbed application
running\n");
    initStats();
    printStats();
    printf("application terminated\n");
}
```

- In a separate test program we use two procedures that operate on the data structure.

```
Football Stats package testbed application running
2, 4 = 1
18, 2 = 4
18, 9 = 6
application terminated
```

```
stats[18][2] = 4;
stats[18][9] = 6;
stats[2][4] = 1;
```

Name	Number
Arsenal	0
Aston Villa	1
Blackburn Rovers	2
Bolton Wanderers	3
Chelsea	4
Everton	5
Leicester City	6
Liverpool	7
Manchester City	8
Manchester United	9
Newcastle United	10
Norwich City	11
Queens Park Rangers	12
Stoke City	13
Sunderland	14
Swansea City	15
Tottenham Hotspur	16
West Bromwich Albion	17
Wigan Athletic	18
Wolverhampton Wanderers	19

Encoding

- To find out what the team names are, we can use the encoding from earlier, and decode our stats about teams 2, 4, 9 and 18.
- Not very nice for us to have to do this.
- Next Lecture show you how to fix this problem!

Q: Why did the programmer quit
his job?

A: Because he didn't get arrays.
(a raise)

THE END