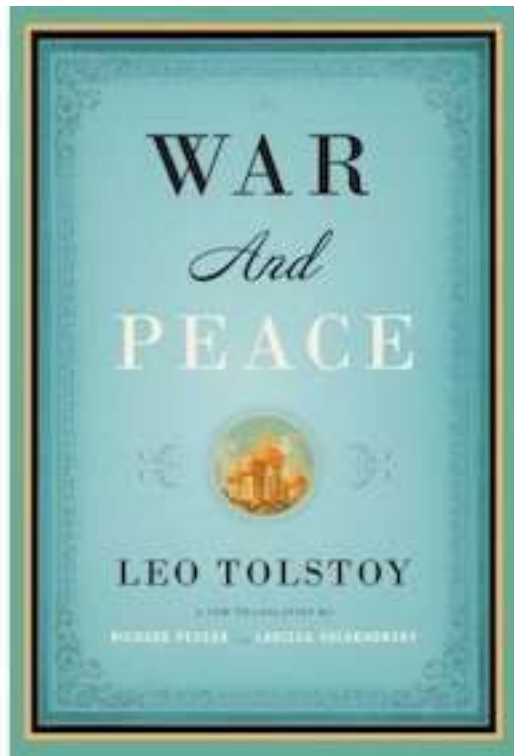


# Unit 4: Strings

# Examples of Strings



**Dara Ó Briain** @daraobriain

16 Nov

About to go for dinner, before sleep, before Zero-g flight tomorrow. Must choose a meal that returns well. Excited though.

Expand

Reply Retweet Favorite \*\*\* More



# Strings in Ada

Ada supports Strings at the **language level**.

- it is a basic type like integer, real, character, boolean etc.

Declaration –

- `name : STRING (1..5) ;`

- Assignment –

- `name := "Tommy";`

name	'T'	'o'	'm'	'm'	'y'
	1	2	3	4	5

# Indexing in Ada

- We can access individual elements (characters) of a string using an **index** or **subscript**.
  - `name (2) := 'a' ;`
- The result of indexing a String is a single character.

name	'T'	'a'	'm'	'm'	'y'
	1	2	3	4	5

# Slices – parts of a String

- As well as accessing single elements at a time, Ada can choose a number of contiguous elements simultaneously, by creating a slice.

```
name(1..2) := "Ji";
put(name(2..4)); //like printf in C
```

would output "imm".

name	J	i	m	m	y	name (1..2)
	1	2	3	4	5	

name	J	i	m	m	y	name (2..4)
	1	2	3	4	5	

# (con)catenation : the & operator

name := "Jimmy"

operation	Value of name after
name := "Tom" & "my";	"Tommy"
name := "Ji" & name(3..5);	"Jimmy"
name := name & "Johnson";	"JimmyJohnson"
name := name(1..4) & 'o';	"Jimmo"

# String Comparison

- Imagine an exhaustive dictionary with every possible combination of printable ASCII letters.
- The dictionary is sorted according to Alphabetical order
- String comparison is

A	0	"Clare" == "Clare " -> true
AA	1	if("AAAA" < "BB") -> true
AAA	2	Because "BB" is further down in the list
AB	...	than "AAAA"
...		
Clare		if ("Diana" < "Clare") -> false
...		Because "Clare" comes before "Diana" in
Diana	..	the list

# Comparing Strings

```
name := "Clare";
```

- Normal comparing operations (=, <, >, etc.)
- Alphabetical order is used
- Case-sensitive (upper-case letters smaller than lower-case)

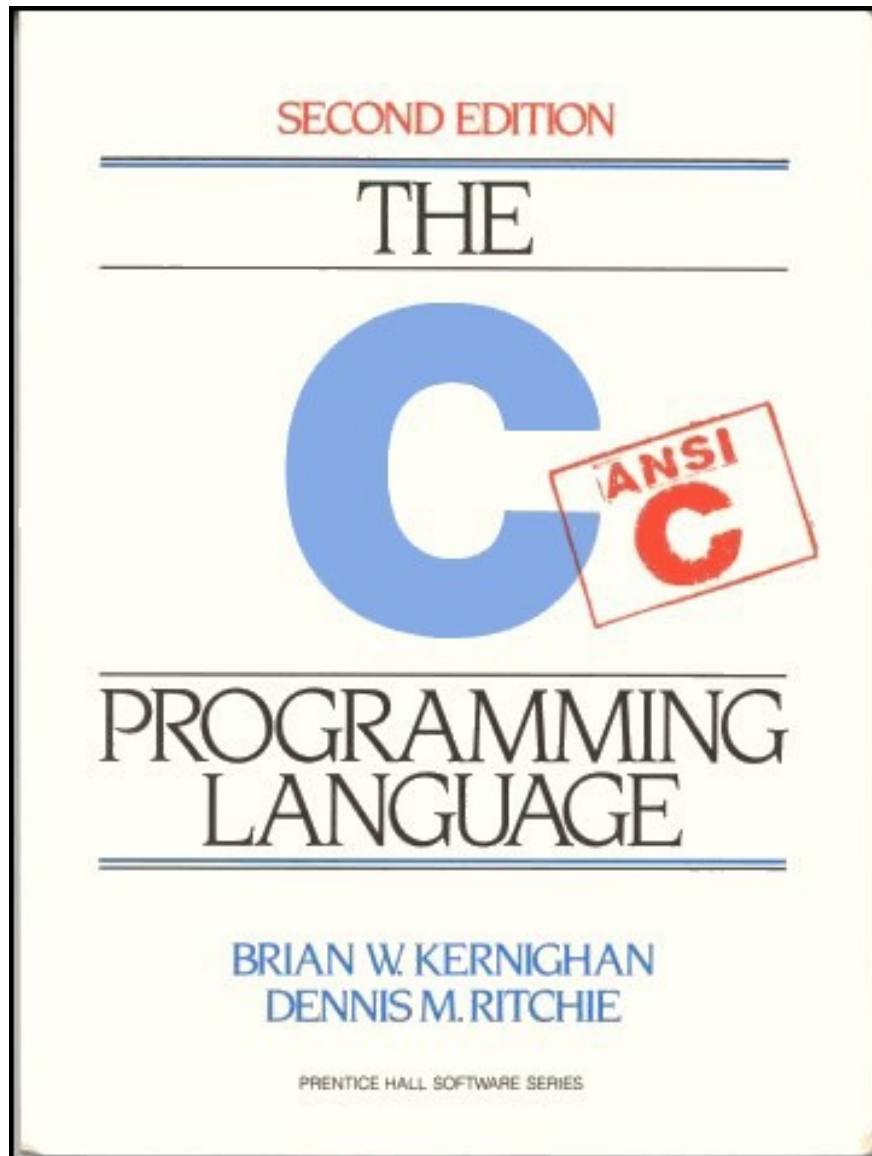
name = "Clare"	true
name < "Diana"	true
"Betty" < "Peter"	true
"Jill" > "Jack"	true
"Liz" > "Elizabeth"	true
"Adam" < "Eve"	true
"Victor" /= "Victoria"	true
"Victor" < "Victoria"	true
"Rose" < "rose"	true



# Basic String Operations

denotation	"squid-pie"
concatenation	"them" & "atic" = "thematic"
substring (slice)	"exotic"(4..6) = "tic"
comparisons	if (string1 > string2) ...

- If they are not available in the language, they are usually supplied in a software library.
- In Java, they are methods in the String class.
- In 'C', they are in a software library.



## STRINGS IN 'C'

# Missing Operators in C

- There are no
  - string assignment operators (':=').
  - string comparison operators ('<', '=', '>').
  - string combination operators ('&').
- However, there are built-in functions to do this common tasks.

## Some 'C' 13

declaration	<code>char text [50];</code> <code>char text [] = "this is a string"</code>
copy	<code>strcpy(text, "hello Mum");</code>
concatenation	<code>strcat(name, "Johnson");</code> (result placed in first parameter)
substring (slice)	not directly supported
length	<code>unsigned size = strlen(text);</code>

# Some more 'C'<sup>14</sup>

comparisons	<pre>int res = strcmp(s1, s2);</pre> <ol style="list-style-type: none"><li>1. returns a negative value if s1 precedes s2 in alphabetical order;</li><li>2. returns zero if s1 and s2 are equal;</li><li>3. returns a value greater than zero if s1 follows s2.</li></ol>
	<pre>int res = strncmp(s1, s2, n);</pre> <p>As above but compares only the first n characters of s1 and s2</p>

# Strings in 'C'

- You must include “string.h” at the start.
  - `#include <string.h>`
- 'C' strings are an array of characters which are always assumed to be terminated by a NULL byte (`'\0'`).
- 'C' string literals always have a NULL byte added at the end.
  - “this is a string” contains the 16 characters that appear between the quotes plus a NULL byte therefore it has 17 characters.

# Strings in 'C'

- Consider this :

```
char text[10];
```

```
strcpy(text, "can do");
```

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

0	1	2	3	4	5	6	7	8	9
'c'	'a'	'n'	' '	'd'	'o'	'\0'	?	?	?

– `strlen(text)` = 6, not 10, nor 7.

# String Comparisons in C <sup>17</sup>

- `!=` and `==` are NOT string operators in C

```
int foo() {  
    char str1[]="apple";  
    char str2[]="apple";  
    if (str1 == str2)  
        return 1;  
    return 0; }
```

This actually compares the base address of two arrays but not strings! Should be `if strcmp(str1, str2)==0`

Other common mistakes:

- seems reasonable to assume that `strcmp` returns “true” (nonzero) if `s1` and `s2` are equal; “false” (zero) otherwise
- In fact, exactly the opposite is the case!





# Revisit

- Consider our example from [unit 2](#), counting the occurrence of lower-case letters in a String (text).
- To illustrate this, we discuss how Strings are stored in low-level languages such as C.

# String Scanning : how?

- Recall how do we scan a 1-D array :  
`for (int i = 0; i < N; i ++)` ....
- In general, we know how many elements are in the array (here, N) and so we can use a `for` loop.
  - The number of elements are the length of the string.

# Scanning by length<sup>21</sup>

- In C, we can use `strlen()` to get the length of the string..

```
char cString[100] = "hello";  
int count = strlen(cString);  
for (int i = 0; i < count; i++)  
    printf( "%c", cString[i]);
```

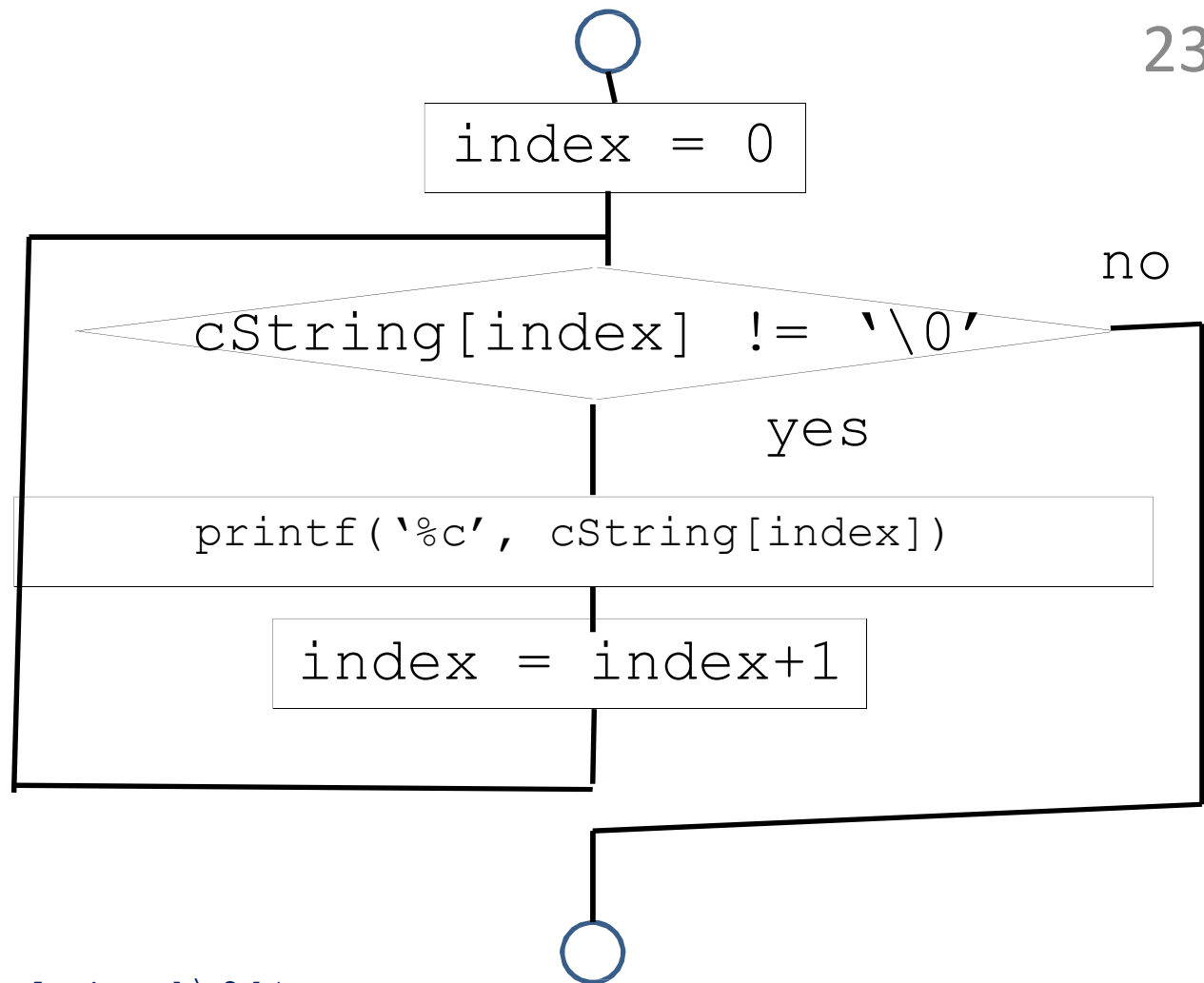
0	1	2	3	4	5	6	7	....	99
'h'	'e'	'l'	'l'	'o'	'\0'	?	?	....	?

# Scanning with a terminator

- Recall that to terminate the string with a NULL byte.
- So instead of a for loop, we can also use a while loop, and we continue printing characters until we reach an element containing a NULL byte (ASCII 0).

```
int index = 0;
while (cString[index] != '\0')
{
    printf("%c", cString [index]);
    index++; //remember to increment index!
}
```

0	1	2	3	4	5	6	7	...	99
'h'	'e'	'l'	'l'	'o'	'\0'	?	?	...	?



```
int index = 0;
while (cString[index] != '\\0')
{
    printf('%c', cString [index]);
    index++;
}
```

# Revisit: Unit 3: Football <sup>24</sup>

## Example

	0	1	2	3	4	5	6	7	8	9	10	11	12	...
0								6				4		
1														
2														
3														
4														
5														
6														
7														
8														
9														
10														
11	1													
12														
...														



Name	Abbr.	Number
Arsenal	AR	0
Aston Villa	AV	1
Blackburn Rovers	BR	2
Bolton Wanderers	BW	3
Chelsea	CH	4
Everton	EV	5
Fulham	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

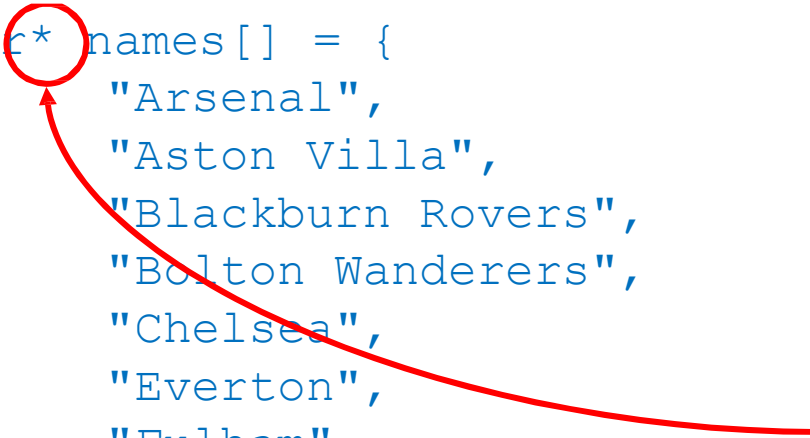
- So that we don't have to use the full names of the teams on the slides, here are some encodings.



# Our Football Example

- In our football application, we were printing out the encoded values of team names.
  - Numbers not team names – confusing!
- What we would like is some code that:
  - transforms a number into the appropriate team name.
  - transforms a team name into the appropriate number

```
char* names[] = {  
    "Arsenal",  
    "Aston Villa",  
    "Blackburn Rovers",  
    "Bolton Wanderers",  
    "Chelsea",  
    "Everton",  
    "Fulham",  
    "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 will be looking at the “\*” pointer notation later in the course.

For now, think of this as a 1D array, where each element is a ‘C’ string, which in turn is of course a 1D array of characters.

```
#define NUMBER_OF_TEAMS 20
```

```
int encode (char* team_name) {  
    int i = 0;  
    while (i < NUMBER_OF_TEAMS) {  
        if (strcmp(team_name, names[i]) == 0){  
            return i;  
        }  
        i++;  
    }  
  
    return -1;  
}
```

```
void initStatsWithNames()
{
    int team_a, team_b;
    // 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;

    team_a=encode("Arsenal"); team_b= encode("Norwich City");
    stats[team_a][team_b] = 4;

    team_a=encode("Arsenal"); team_b=encode("Stoke City");
    stats[team_a][team_b] = 6;
}
```

Still using numbers; the team names don't matter here.

Using the encode function; we use team names and they are converted to the “code number” of the team, and then used as indices.

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

- No need for a function here; just a straight “table lookup”

```
Arsenal, Norwich City = 4
Arsenal, Stoke City = 6
```

# Home Exercise

Implement the Football example using C.

# String processing examples (1)

**Holidaymakers have been warned to watch their words after two friends were refused entry to the US on security grounds after a tweet.**

Before his trip, Leigh Van Bryan wrote that he was going to "destroy America".

He insisted he was referring to simply having a good time - but was sent home.

Trade association Abta told the BBC that the case highlighted that holidaymakers should never do anything to raise "concern or suspicion in any way".

The US Department for Homeland Security picked up Mr Bryan's messages ahead of his holiday in Los Angeles.



Post-9/11 USA is highly cautious of any perceived threat, Abta said

---

## Related Stories

---

**Twitter airport hoax appeal fails**

**Man guilty of Twitter**

<http://www.bbc.co.uk/news/technology-16810312>

# String processing examples (2)

demostration examples - DUE 21-Jun-2009 What's New Paper 17 of 17

Originality GradeMark PeerMark

anorexia essay BY C K

turnitin 90% SIMILAR -- OUT OF 0

## 10 What is anorexia nervosa?

8 Anorexia nervosa is a distorted body image that overestimates personal body fatness and an eating disorder affecting mainly girls or women, although boys or men can also suffer from it. It usually starts in the teenage years. It is estimated that about one out of every 100 adolescent girls has the disorder. Caucasians are more often affected than people of other racial backgrounds, and anorexia is more common in middle and upper socioeconomic groups. 2 The overwhelming desire to become thin drives people with anorexia nervosa to refuse to eat even when they are hungry. Although adults often describe people with anorexia as "model students" their personal lives are usually marred by low self-esteem, social isolation and unhappiness. Anorexia nervosa cannot be self-diagnosed.

2 We can characterise the people with this disease by their body because their weight 4 is maintained at least 15 per cent below that expected for a person's height. It is self-induced weight loss caused by avoiding fattening foods and may involve taking

### Match Overview

1	www.canadiancnc.com	Internet source	28%
2	Submitted to Universit...	Student paper	16%
3	blogs.myspace.com	Internet source	15%
4	Submitted to Universit...	Student paper	10%
5	www.drugfare.com	Internet source	8%
6	www.slideshare.net	Internet source	7%
7	www.medicinenet.com	Internet source	3%

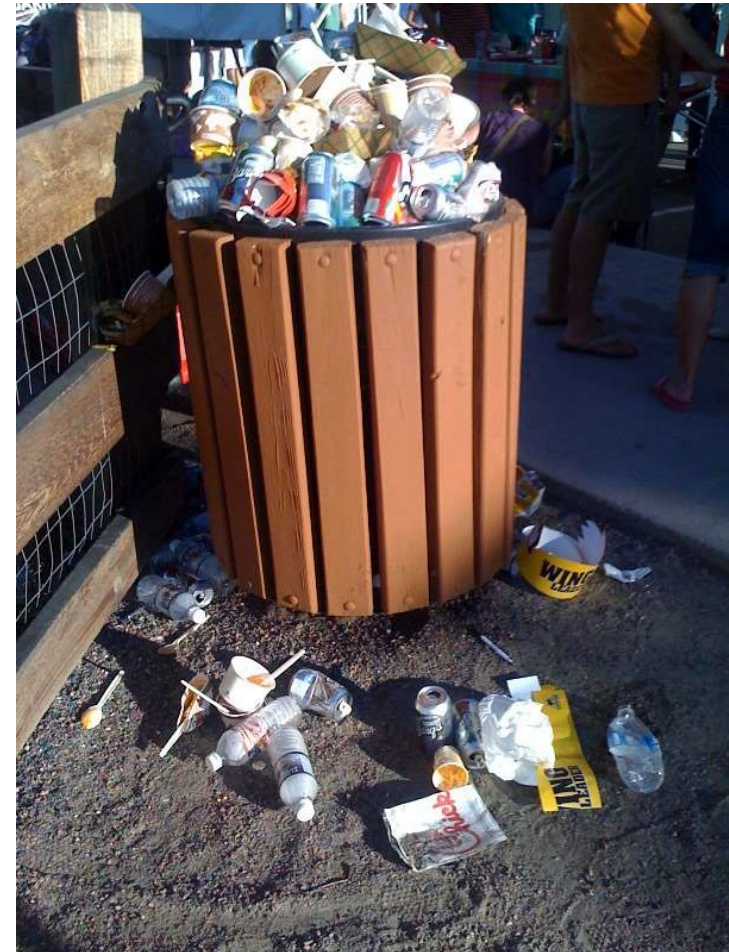
PAGE: 1 OF 4

Text-Only Report



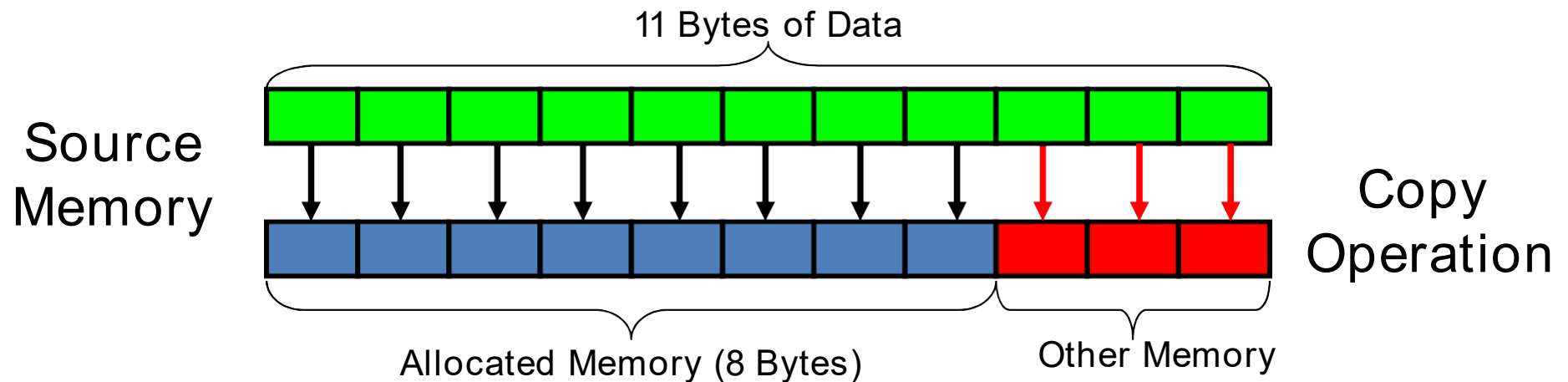
# Buffer Overflow

- A buffer overflow occurs when data is written outside of the boundaries of the memory allocated to a particular data structure.
- Buffer overflow is very danger!

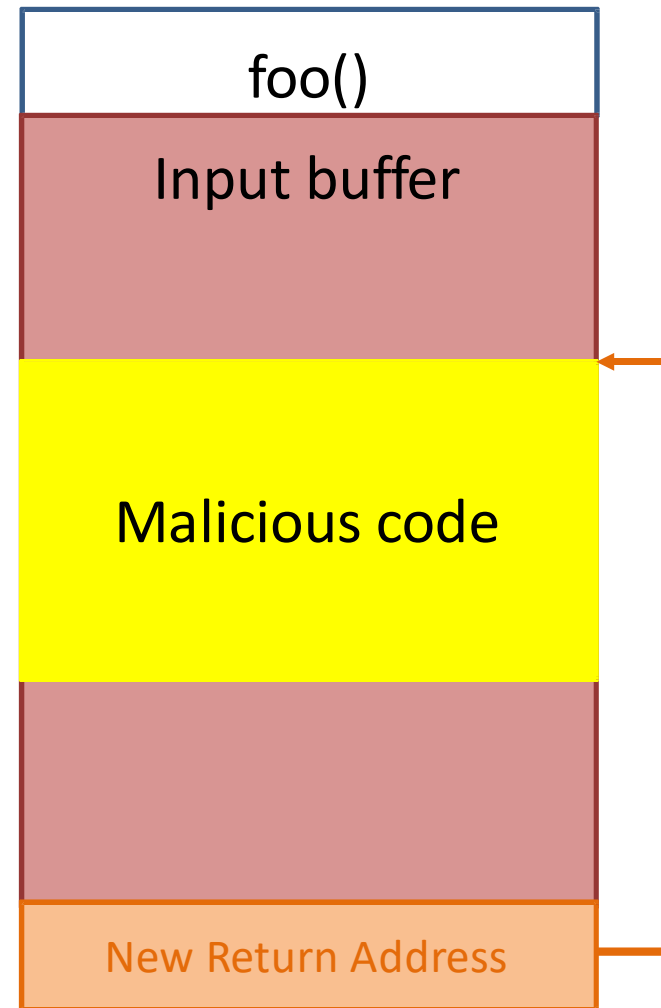
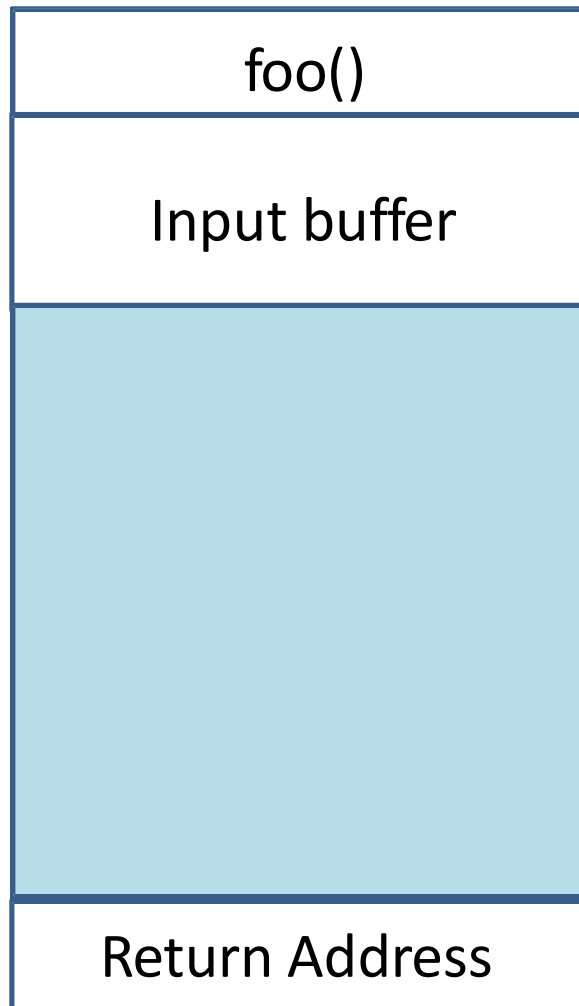


# Buffer Overflow: String Copy

```
char password[8];  
strcpy(password, "12345678910");
```

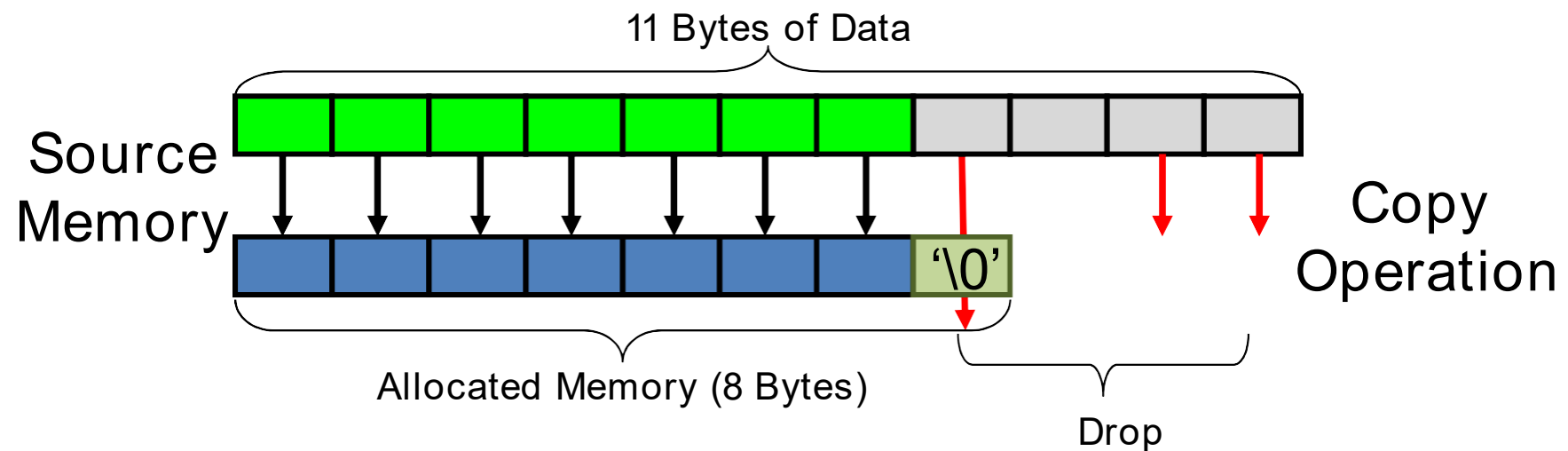


# Exploitation of Buffer Overflow



# Buffer Overflow Fixed

```
char password[8];  
strcpy(password, "12345678910");  
strncpy(password, "12345678910", 8);  
/*overflow safe - copy at most 8  
characters*/
```





# THE END

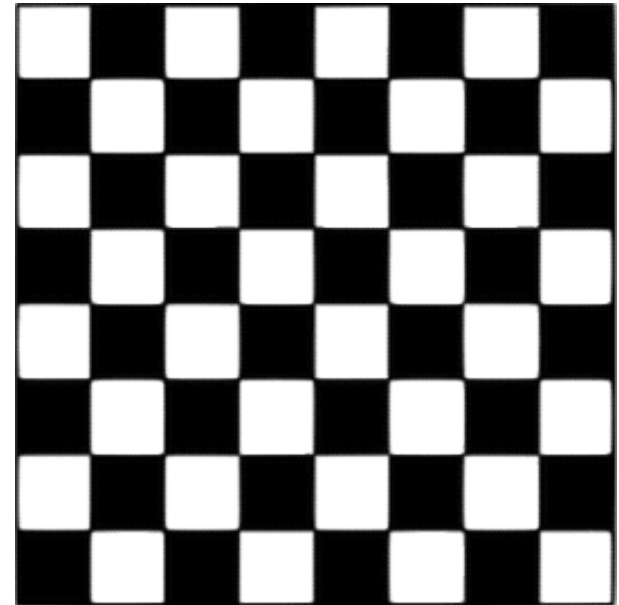
# Unit 5: Structures and Objects

# Consider a chessboard

- We can represent it as a 2-D array of integers

```
# define WHITE 0
# define BLACK 1
# define SIDE_SIZE 8

int chessboard
[SIDE_SIZE][SIDE_SIZE];
```

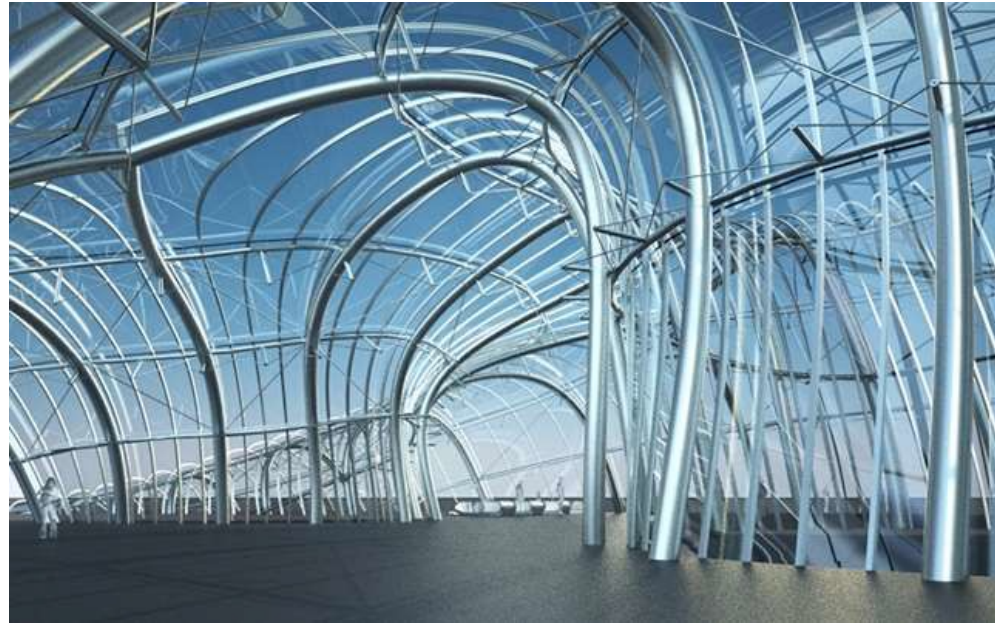


# But ....

- Colour isn't the only thing we want to know about a square.
  - Whether it is occupied or free.
  - If occupied ...
    - By what value of piece?
    - By what colour of piece?







Part One

# STRUCTURES

# Structure

- A structure (record) is a compound item
- Unlike an array, it is heterogeneous – i.e., components of various types
  - Its components are called fields
  - Each field is identified using a name (not a subscript/index)
  - A structure holds various different properties of a single entity

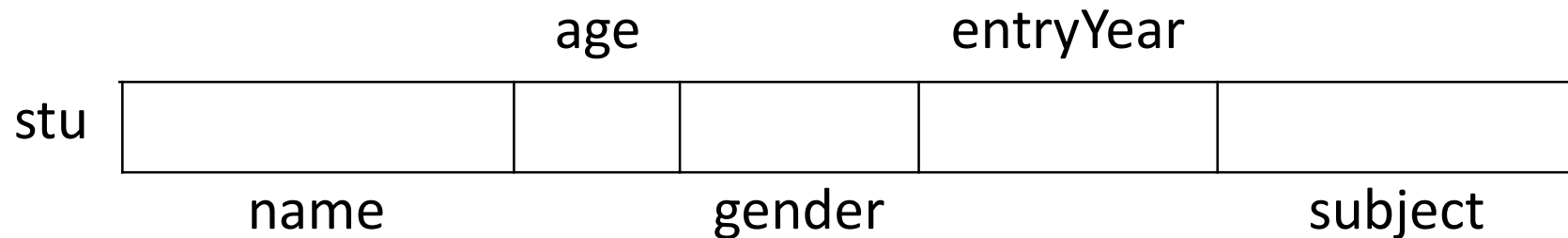


## Example

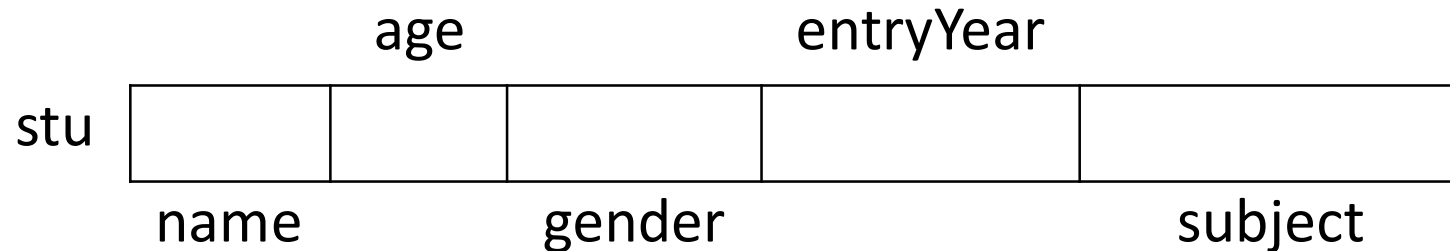
- Here is a 'student' structure of LUSI

The fields types are:

String, int, char, int, String



# Defining a structure type 7

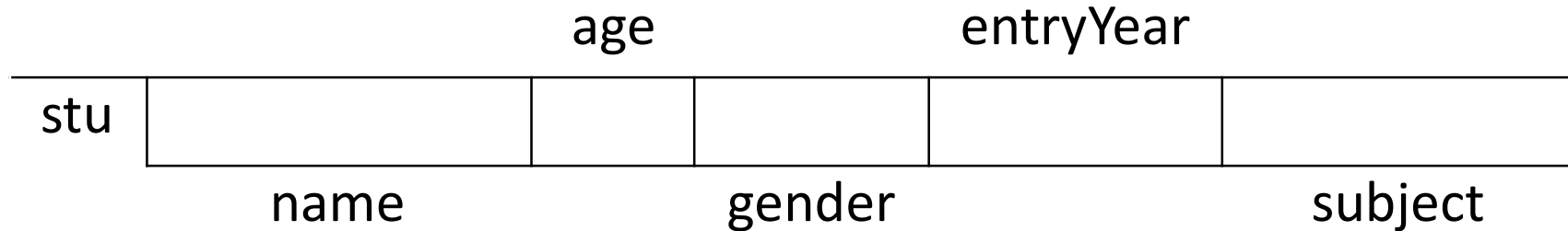


- We define a structure type by
  - giving the type a name
  - listing the field names and field types

```
struct studentRecord {  
  
    String name;  
    int age;  
    char gender;  
    int entryYear;  
    String subject;  
  
};  
struct studentRecord    stu;
```

General form of a  
structure type  
declaration ...

# More on Structures 8



- Note that –
  - Field names are user-defined symbols
  - We cannot scan a structure – only refer to each field by its name
  - Access typically uses 'dot notation':
    - `stu.age` refers to the age field of the structure
- Examples:

```
if (stu.age > 24)
    printf("%s is a mature student", stu.name);
```

# Structures in 'C'<sup>9</sup>

- To model a student in 'C' we can do one of the following.
  - Declare a structure with an appropriate name.
  - To create a variable of this structure

- `struct Student stu;`

```
struct Student {  
    char name[50];  
    int age;  
    char gender;  
    int entryYear;  
    char major[50];  
};
```

```
stu.gender = 'm';  
strcpy(stu.name, "Joe Bloggs");
```

# Structures in 'C'

- It is probably preferable to make this into a new type.
- To create a variable of this structure

```
Student stu;
```

```
typedef struct{  
    char name[50];  
    int age;  
    char gender;  
    int entryYear;  
    char major[50];  
} Student;
```

# Chessboard revisited

```
#define SIDE_SIZE 8

typedef enum{black, white} Colour;
typedef enum {pawn, knight, bishop, rook, queen, king} PieceValue;

typedef struct{
    Colour pcoulour;
    PieceValue value;
} ChessPiece;

typedef struct{
    Colour sqcolour;
    bool occupied;
    ChessPiece occupier;
} Square;

Square chessboard [SIDE_SIZE][SIDE_SIZE];
```





# Abstract data types (ADTs)

- From Latin: to ‘pull out’—the essentials
  - To defer or hide the details
- I don’t need a mechanic’s understanding of what’s under a car’s hood in order to drive it
  - What’s the car’s interface?
  - What’s the implementation?



# Floating point numbers revisited

- You don't need to know how much about floating point arithmetic works to use `float`
  - All you need to know is the syntax and meaning of operators, `+`, `-`, `*`, `/`, etc.
- Hiding the details of implementation is called encapsulation (data hiding)

# Abstract Data Types (ADTs)

- A simplified definition of what an ADT is ...
  - A set of values, and a set of operators on those values.
    - ADT = properties + operations
  - Think of a chesspiece ...

## Taken together ...

- Type safety and abstraction make programming
  - Less error-prone
  - Easier



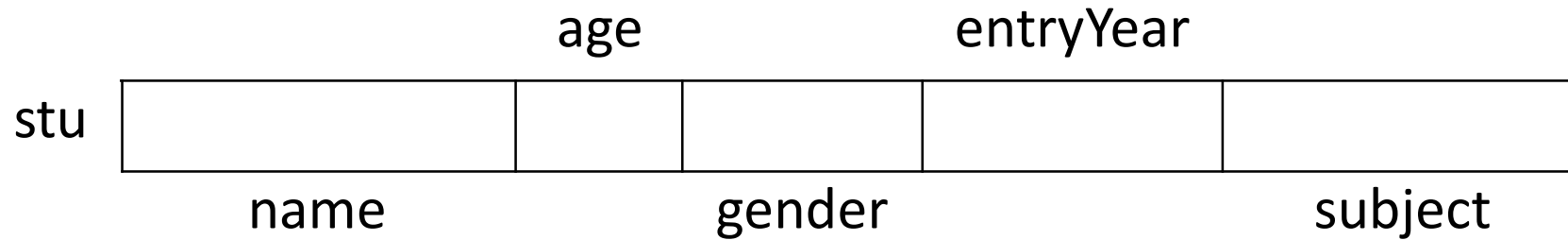
# There are two types of programmer ...

- The programmer of the ADT
- The user of the ADT
- The former should be suspicious of the latter!
- They should prevent them from doing daft things
- But they should take care to make their ADT do useful things

Part Two

# OBJECTS





```
class Student
{
    public String name;
    public int age;
    public char gender;
    public int entryYear;
    public String subject;
};
```

- Note that:
  - the attributes (name, age, etc.) are not private
- In effect, this defines a structure type, like the one we were just looking at.

- Using the definition of the Student class, we can now create an actual record:

```
Student stu = new Student();
```

- and update its fields using dot-notation:

```
stu.name = "Arthur Daley";  
stu.age = 62;    // a mature student!  
stu.gender = 'M';  
stu.entryYear = 2015;  
stu.course = "Computer Science";
```



```
class Student
{
    private String name;
    private int age;
    private char gender;
    private int entryYear;
    private String subject;
    public int get_age() {return age;}
    public void set_age(int ag) {age=ag;}
}
```

- We have now made the attributes private, which means users of this class cannot directly access them.
- We have to provide “get” and “set” methods for the attributes we wish to grant restricted access to.

## Quiz

- Let's say we have a char attribute "maritalStatus", which can only sensibly have these values : 's', 'm', 'd', 'w'.

Question:

- How can we make sure no other value is stored?

```
import java.util.*;
import java.io.*;
class Student
{
    private String name;
    private int age;
    private char gender;
    private int entryYear;
    private String subject;
    private char maritalStatus;
    public boolean setMaritalStatus(char x)
    {
        boolean ok = false;
        switch (x)
        {
            case 'm' : case 'w' : case 's' : case 'd' : ok = true; break;
            default : ok = false;
        };
        if (ok) maritalStatus = x;
        return ok;
    }
};
```

allowed values

case 'm' : case 'w' : case 's' : case 'd' : ok = true; break;

only do the assignment  
if it is valid to do so

# SOME PRACTICAL DIFFERENCES BETWEEN STRUCTURES AND OBJECTS

# Structures and Objects

- A structure is just a collection of data items.
- An object is a higher-order structure, because it consists of both data items and **methods bundled up together**.
  - The methods are used for performing operations on the data items
  - It is the bundling together of methods and data which is important
    - e.g. we cannot apply a method to an object of the wrong class.

# Data Abstraction

- Use of 'get' and 'set' methods means that the programmer's view of the data may be different from how it is actually held.
- Example
  - Suppose our Student objects also have a date attribute (e.g., date-of-birth).
  - There might be a `setDoB` method, allowing the programmer to specify the day, month and year:
    - `stu.setDoB( 27, 5, 1981);`

# Data Abstraction

- `stu.setDoB( 27, 5, 1981 );`
- But internally, the date might be stored as the number of days since, say, 1st January 1964.
- Methods `setDoB` and `getDoB` would have to convert between day/month/year data, and the internal (integer) data.
  - Keeping the actual data private from the user is called **encapsulation**.

THE END