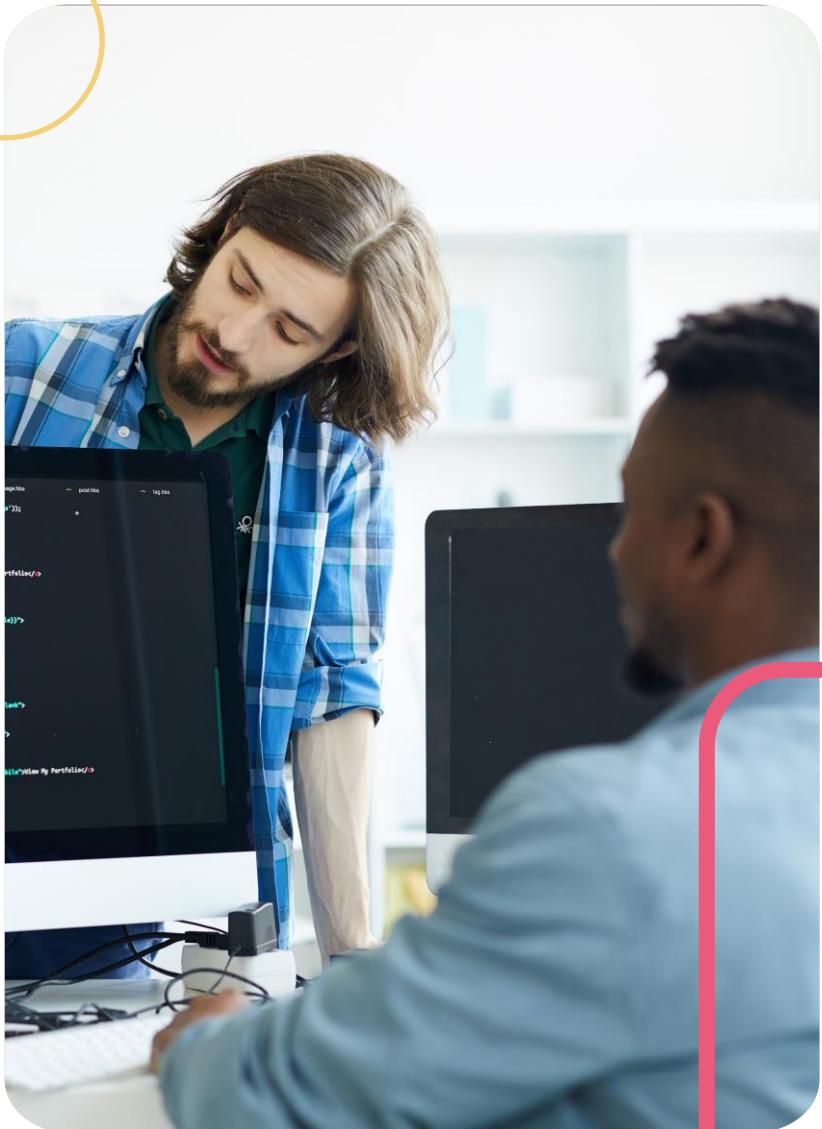


# **Diploma in**

# **Computer Science**

**Data Types**



## Objectives

Describe the data types available in C

Understand the reasoning involved in choosing  
a data type

Appreciate the various data structures available in C

Understand how type qualifiers modify variables



**Data, data...  
and more data**



Data types depend entirely on the hardware you're running your code in.

#### Four classes:

- Basic data types
- Enumerated data types
- Void data types
- Derived data types

# Basic data types

Two categories:

- integer data types
- floating-point data types



# Integer data types

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes or (4bytes for 32 bit OS) 32-bit 32768 to 65536 to 2147483648 to 4294967295	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

Some data types have alternate specifications.

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <float.h>

int main(int argc, char** argv) {

    printf("CHAR_BIT : %d\n", CHAR_BIT);
    printf("CHAR_MAX : %d\n", CHAR_MAX);
    printf("CHAR_MIN : %d\n", CHAR_MIN);
    printf("INT_MAX : %d\n", INT_MAX);
    printf("INT_MIN : %d\n", INT_MIN);
    printf("LONG_MAX : %ld\n", (long) LONG_MAX);
    printf("LONG_MIN : %ld\n", (long) LONG_MIN);
    printf("SCHAR_MAX : %d\n", SCHAR_MAX);
    printf("SCHAR_MIN : %d\n", SCHAR_MIN);
    printf("SHRT_MAX : %d\n", SHRT_MAX);
    printf("SHRT_MIN : %d\n", SHRT_MIN);
    printf("UCHAR_MAX : %d\n", UCHAR_MAX);
    printf("UINT_MAX : %u\n", (unsigned int) UINT_MAX);
    printf("ULONG_MAX : %lu\n", (unsigned long) ULONG_MAX);
    printf("USHRT_MAX : %d\n", (unsigned short) USHRT_MAX);

    return 0;
}
```

Example

```
CHAR_BIT      : 8
CHAR_MAX     : 127
CHAR_MIN     : -128
INT_MAX      : 2147483647
INT_MIN      : -2147483648
LONG_MAX     : 2147483647
LONG_MIN     : -2147483648
SCHAR_MAX    : 127
SCHAR_MIN    : -128
SHRT_MAX     : 32767
SHRT_MIN     : -32768
UCHAR_MAX    : 255
UINT_MAX     : 4294967295
ULONG_MAX    : 4294967295
USHRT_MAX    : 65535
```

```
-----
Process exited after 0.04854 seconds with return value 0
Press any key to continue . . .
```

Example



# The int data type

Represents all real numbers  
that are not fractions

No fractions, so value is absolute

The fractional part will be discarded



## The short integer data type

- So small that isn't used much
- Can substitute it with int

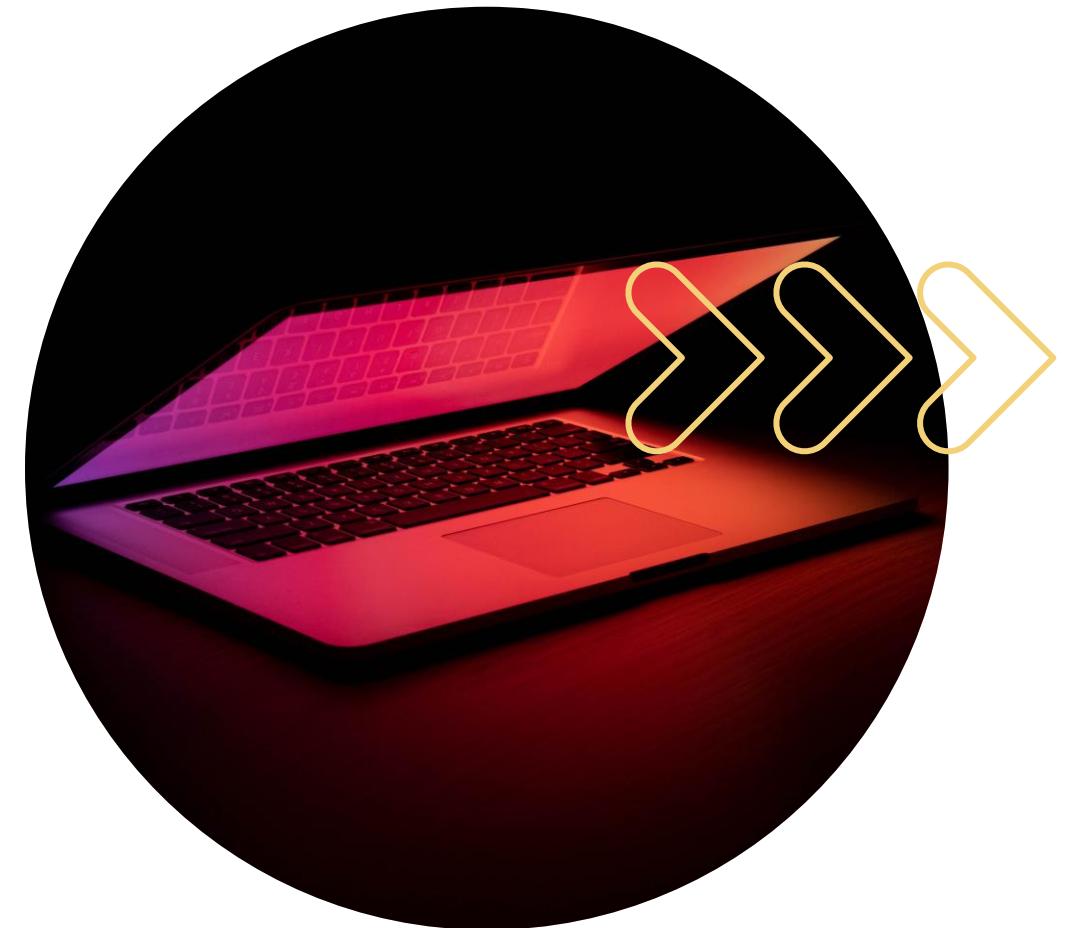
# Long data type

Stores integers

Wider range of values

More memory

Stores at least 32 bits



# Long long integer data type

An overkill

Use it to get -9,223,372,036,854,775,807  
to 9,223,372,036,854,775,807

Best to avoid it





# Boolean data types

- Need to include stdbool.h – compiler needs header file to work
- C Boolean – fake because it uses integers
- True Boolean data type would use logical values
- \_Bool is unsigned integer - can only be assigned values 0 or 1
- Anything else will be stored as 1





Did you know?



The C Boolean type is kind of a fake Boolean as it uses integers. A true Boolean data type would use logical values in the form of "true", and "false".



# Fixed-width integer types





# The void data type

- Function returns as void

`void exit (int status)`

Example

- Function arguments as void

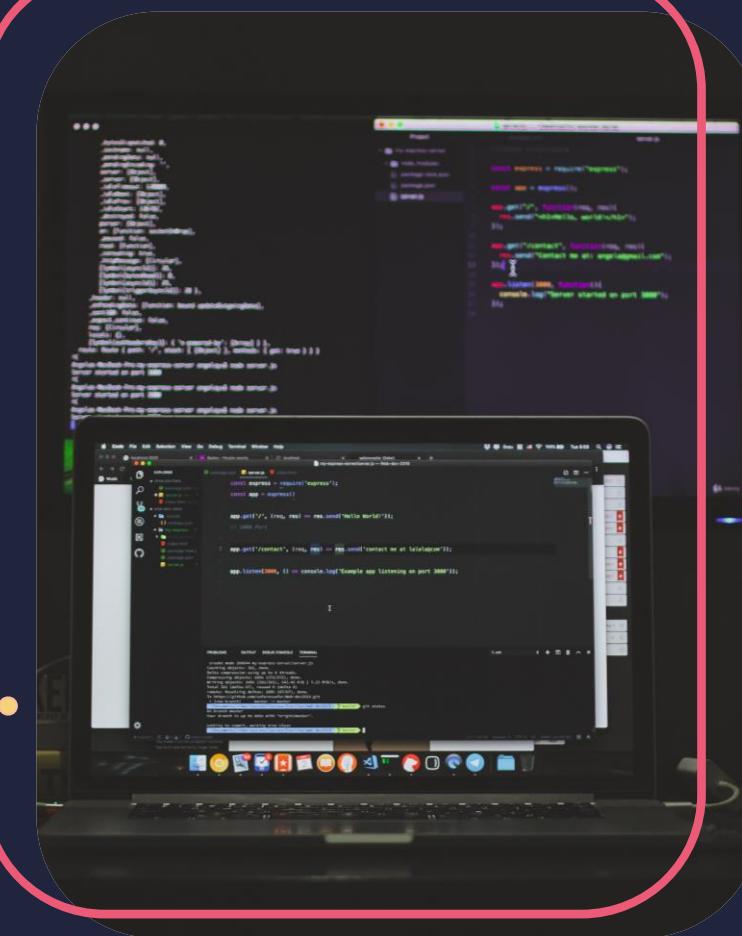
`int rand(void)`

Example

- Pointers to void

`void *malloc( size_t size )` returns a pointer to void which can be cast to any data type

Example





Did you  
know?



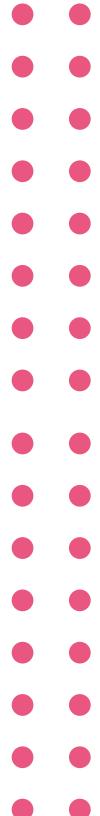
We've just launched an Artificial Intelligence course that would be the perfect next step for you.





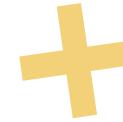
# Characteristics of a data type

- Syntactic
- Representation
- Representation and behaviour
- Value space
- Value space and behaviour



# Choosing the right data type

- Requirements
- Future proofing
- Convenience
- Performance and memory





# Requirements

What you want and need

Often vague and beyond capability  
of computer

Trim to meet technical specs



# Future proofing



Use scalable  
data types

Remember Y2K?

# Convenience

- What's best for your program?
- Good balance between readability and efficiency



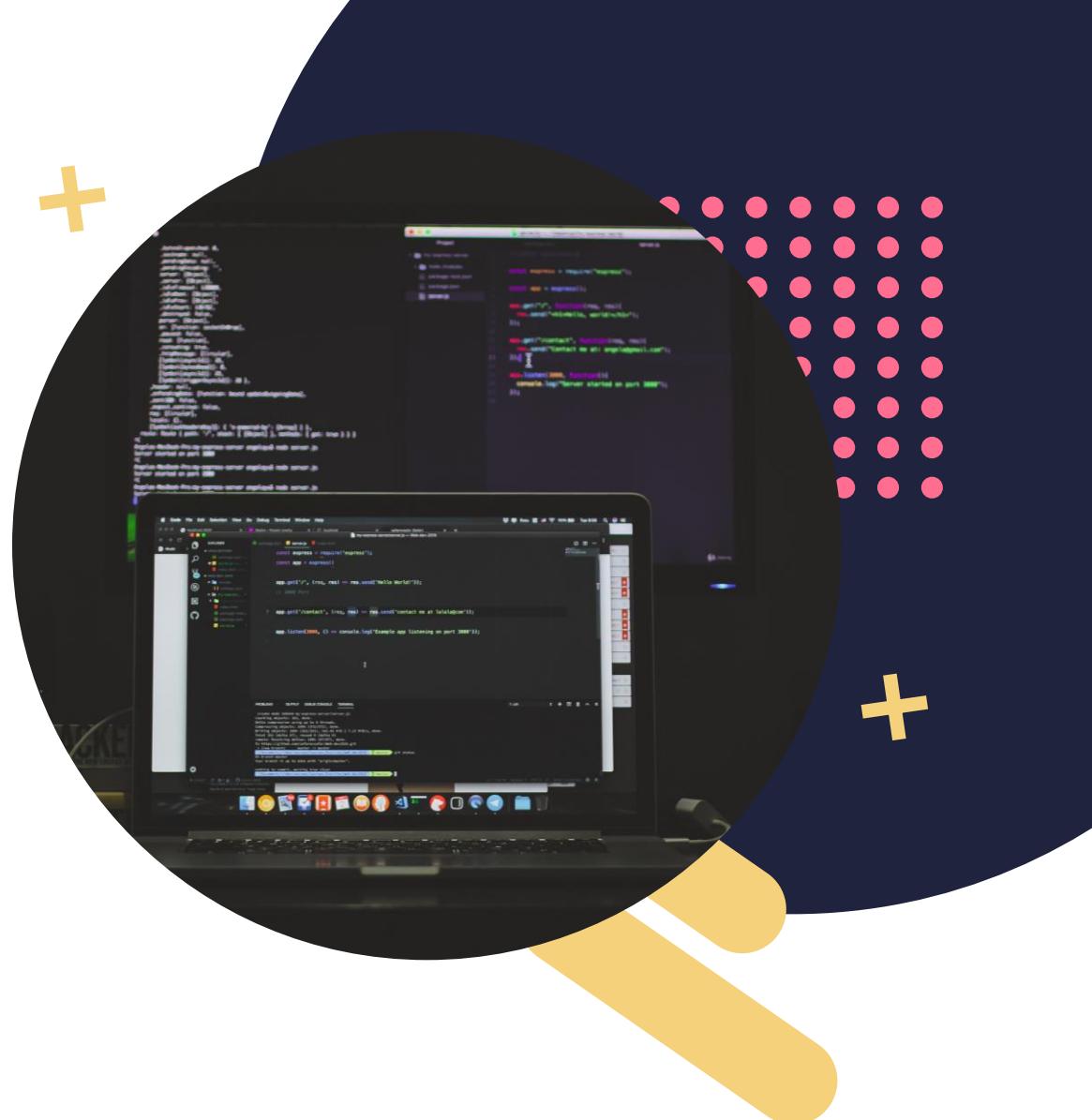
# Performance and memory

- Consider RAM and bits
- Avoid choosing purely on performance
- Consider when memory consumption and performance requirements matter



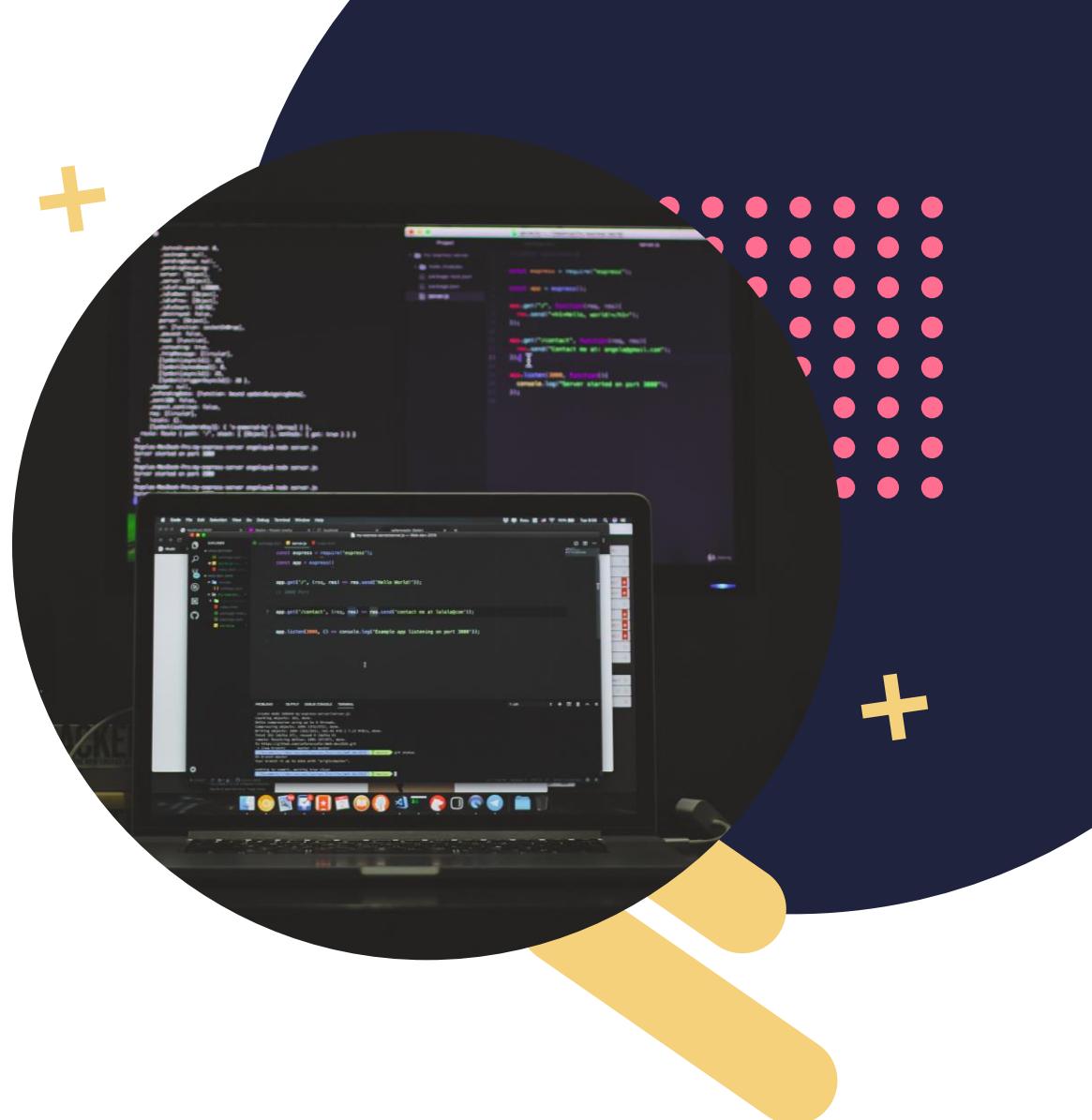
# Choosing the right data type

- C data types allow you to be very specific with the type of data your variables will hold and how they will be manipulated
- Pick the minimum for your task
- This functionality enables C to produce light and efficient code



# Choosing the right data type

- Understand the abilities and limitations of different data types
- Some data types are dependent on hardware
- Use the size of operator to see if the data type is up to the task





# • Advanced data types

# Arrays

A collection of elements of the same data type

An ordered series or arrangement

Pile of books with different titles





# Arrays

- Just declare one array to represent individual variables
- Created in contiguous memory block
- Lowest address corresponds to first element and highest address to last element
- Syntax:  
`datatype arrayName [ arraySize ];`

Char initials[15];

Will hold up to 15 characters

Example

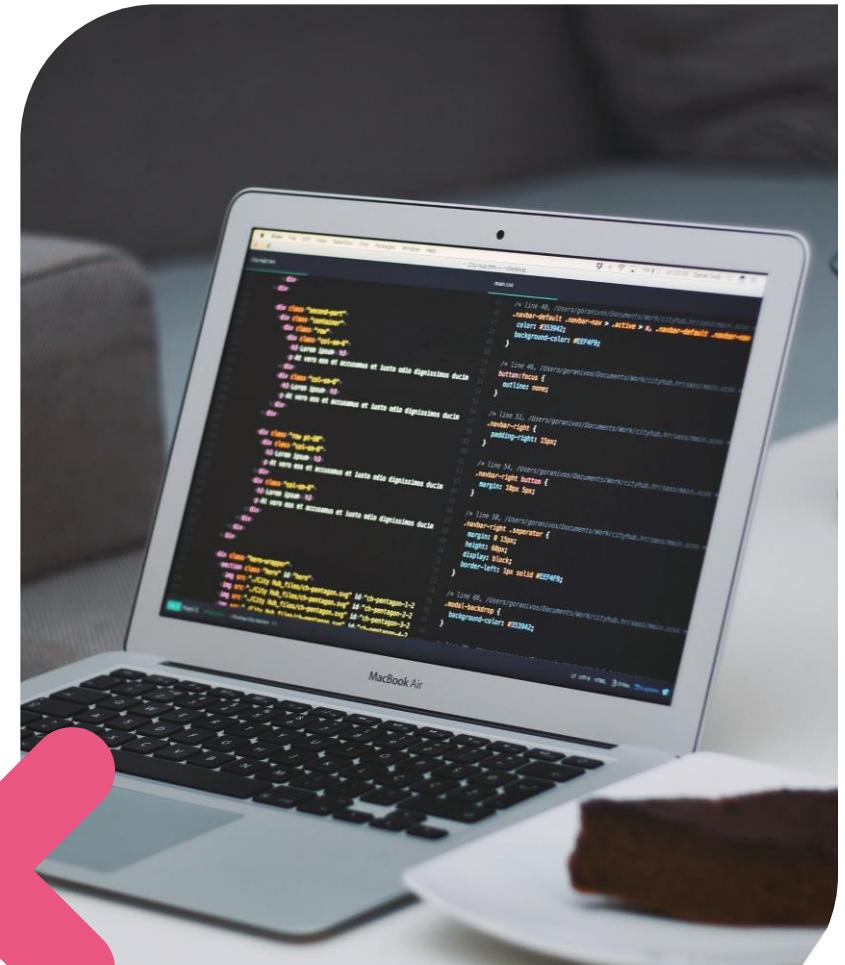
- Number of elements that you assign to the array cannot be larger in size of the array you declared

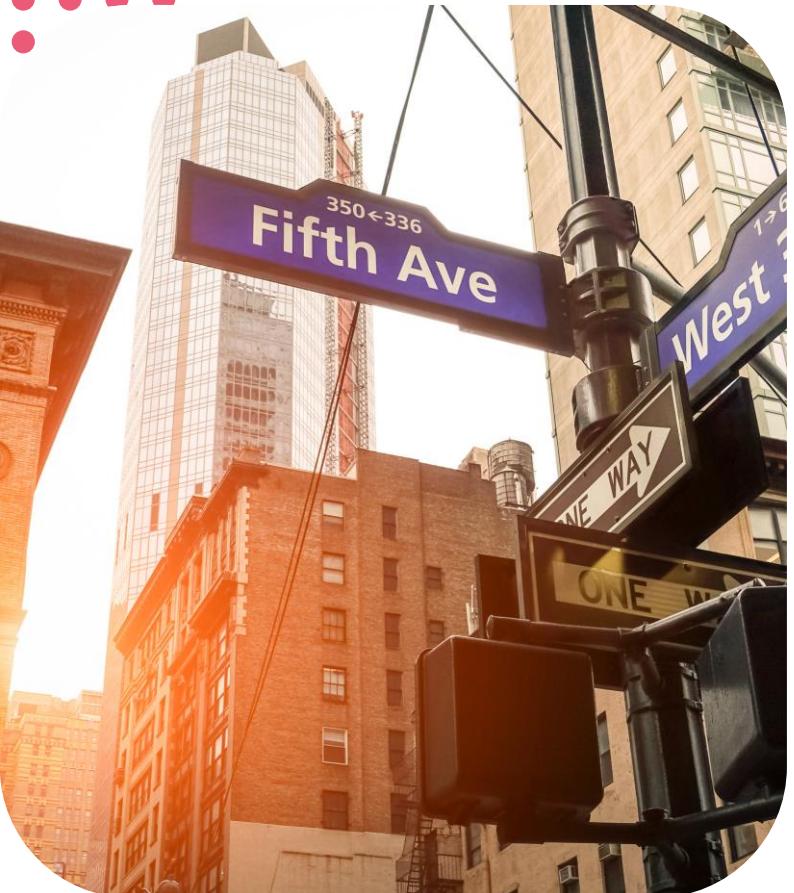
[initials[2]=E;

Letter E will be assigned to 3<sup>rd</sup> position  
in the arrays

Example

- Element accessed specifying the array name and the position of the item in square brackets - item's index.





# Pointers

Variable whose value is the address of another variable

Syntax: type \*var-name;

Always assign a NULL value to a pointer variable if you don't have an address

A NULL pointer is a constant with a value of zero defined in several standard libraries.

# Forms of pointers

Pointer arithmetic

Pointer arrays

Pointers to pointers

Passing pointers to functions

Return pointer from functions



```
index.html
48 <script src="https://www.javascriptkit.com/dhtmltut/clicks/counter1.js" type="text/javascript">
49 <![endif]-->
50 </head>
51 <body>
52 <div class="preloader"></div>
53 <!-- end preloader -->
54 <header class="header">
55 <div class="left-side" data-type="header">
56 <!-- end left-side -->
57 <div class="right-side" data-type="header">
58 <div class="ver-middle">
59 <div class="inner">
60 <h1>Views<br>
61 <b>Discover spots by location</b>
62 <a href="https://itunes.apple.com/app/id1348888888">APP STORE</a>
63 </h1>
64 <img alt="A small icon or logo, possibly a camera or map pin, located below the main title." data-type="header"/>
65 </div>
66 </div>
67 </div>
68 </header>
69 <div class="content">
70 <div class="left-side" data-type="content">
71 <div class="right-side" data-type="content">
72 <div class="ver-middle" data-type="content">
73 <div class="inner" data-type="content">
74 <h2>Recent Views</h2>
75 <ul>
76 <li>View 1</li>
77 <li>View 2</li>
78 <li>View 3</li>
79 <li>View 4</li>
80 <li>View 5</li>
81 </ul>
82 </div>
83 </div>
84 </div>
85 </div>
86 </div>
87 </div>
88 </div>
89 </div>
90 </div>
91 </div>
92 </div>
93 </div>
94 </div>
95 </div>
96 </div>
97 </div>
98 </div>
```



**Pointers are  
powerful but can  
easily create  
messy code if not  
used correctly.**



# Unions

- Store different data types in the same memory location
- Efficient way of using same memory location for different purposes

Syntax:

```
union [union tag] {  
    member definition;  
    member definition;  
    ...  
    member definition;  
} [union variables];
```



# Unions



- Great if running on low memory constraints
- Memory occupied by a union will be large enough to hold largest member of the union
- Member access operator - (.)
- Keyword union defines variables of union type



# Structures

Composite data type declaration that defines group of variables under one name in block of memory

Variables can be accessed via single pointer or struct declared name

Struct data type used for mixed data type records



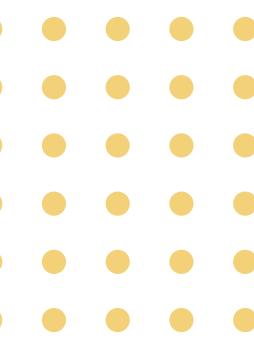
# Structures

C struct directly references contiguous block of physical memory limited by word-length boundaries

Each field located at certain fixed offset from the start

Alignment of particular fields in struct is same as word size of machine





# Three ways to initialise a structure



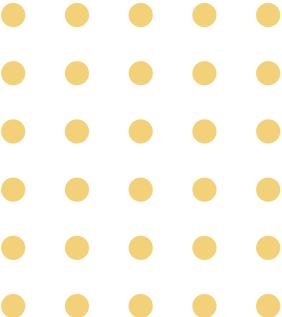
```
// Declare the struct with integer members x, y *
struct point {
    int x;
    int y;
};
```

Example

1. Define a variable p of type point, and initialise its first two members in place

```
struct point p = { 1, 2 };
```

Example



# Three ways to initialise a structure



2. Define a variable p of type point, and set members using designated initialisers

```
struct point p = { .y = 2, .x = 1 };
```

Example

3. Define a variable q of type point, and set members to the same values as p

```
struct point q = p;
```

Example

# Structures

A struct can be assigned to another struct

memcpy()

Example

Use pointers to refer to a struct by its address  
to pass it to a function



A photograph of a lottery ball machine. A large pile of lottery balls is in the foreground, with many more balls visible inside the machine's drum. The balls are white with various colored numbers and patterns.

# Challenge >>

Given a task to create a program that generates lottery numbers for the state lottery, what would be the appropriate data type for the numbers?



# • Type qualifiers

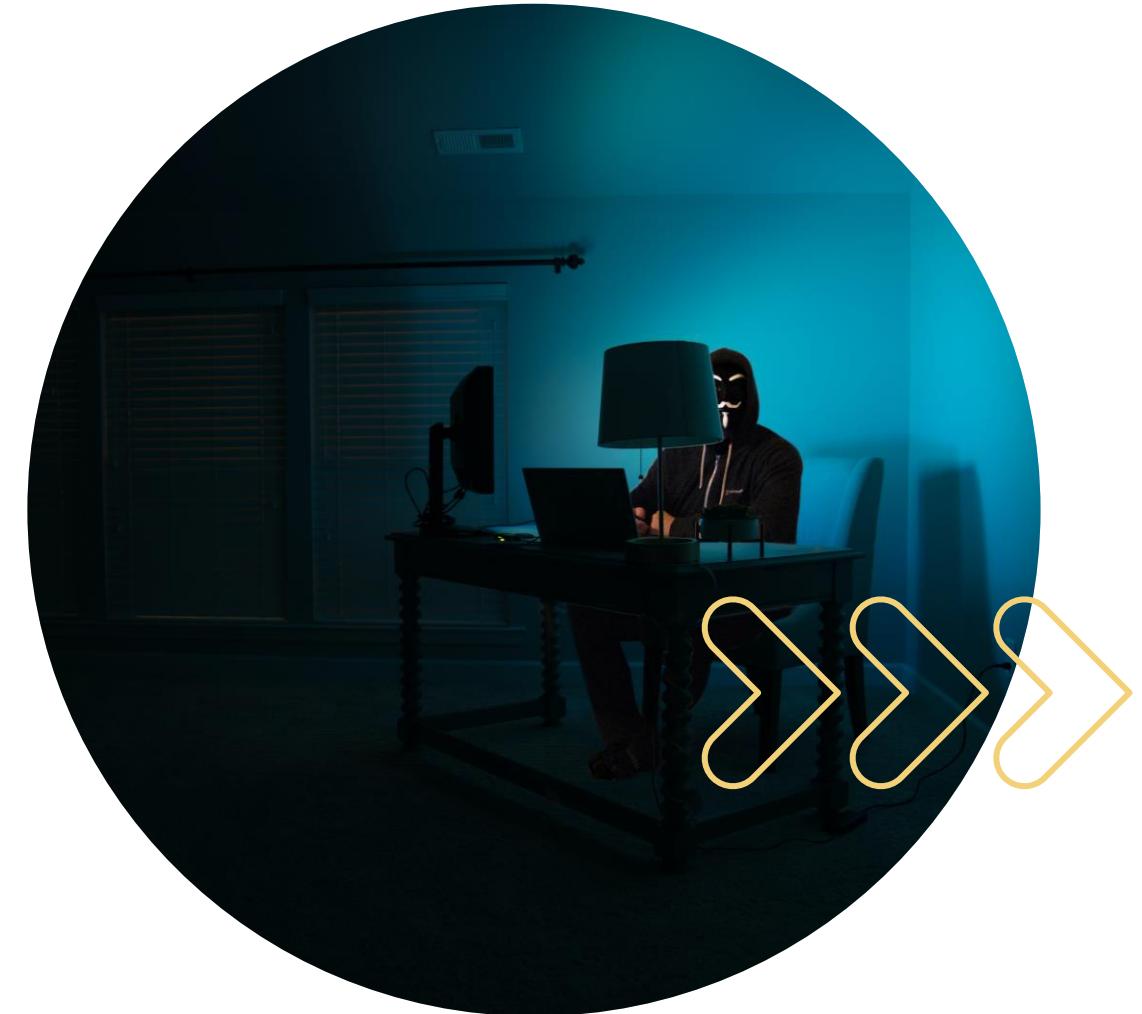
Constants are like  
normal variables, but  
their values can't be  
modified by the  
program once  
defined.



# Constants

Also known as ‘literals’

Can belong to any data type



Variables are **volatile** when the program doesn't change the value of the variable, but it might keep changing without any explicit assignment.





# Volatile

If a global variable's address is passed to clock routine of the operating system to store the system time, the value in this address will keep on changing without any assignment by the program.

Example



# Restrict

- Used in pointer declarations as a type qualifier for pointers
- A way for programmer to instruct compiler to perform optimisations

NOTE: You can't use the restrict and volatile qualifiers at the same time.

