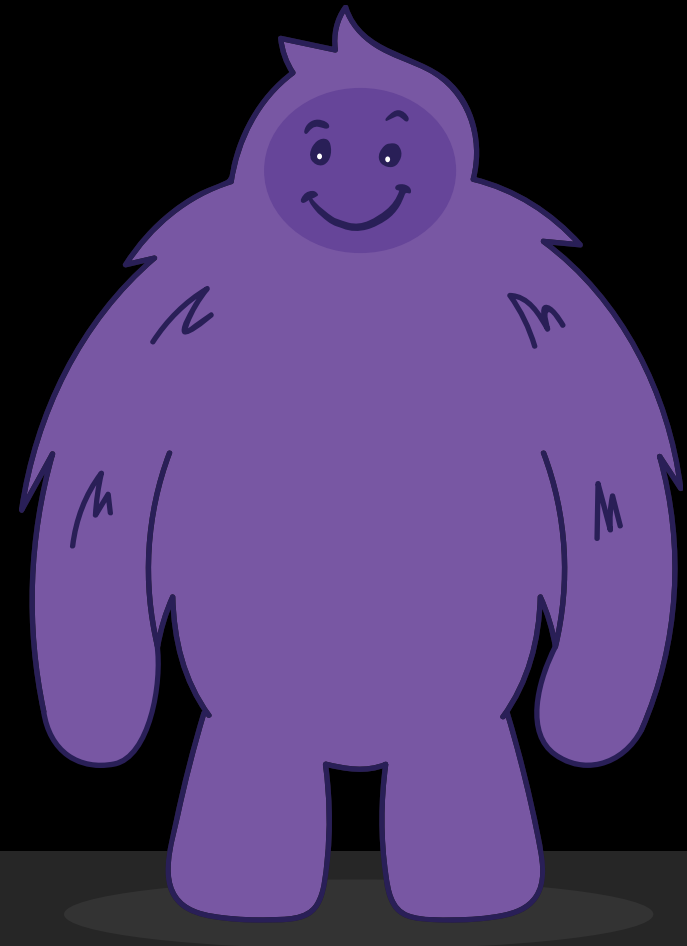


C Intro Pack

Section 1: C, It's Not So Bad After All

The History: B Before C Except After D

- » It all started back in 1969 with the creation of the B Programming Language.
 - Named after the B in “Bell Labs”.
 - It was somewhat similar to C, but contained fewer features and operations.
- » In 1972, Dennis Ritchie and Brian Kernighan released an updated form of B, called C.
 - C is still used today, forming the fundamentals for operating and embedded systems.
- » In the 1980s, Bjarne Stoustrup took the basis of C and developed it into a new object-orientated form called C++.
 - Those of you who know C already will get the name...
- » After C++ came D, which, whilst used, is nowhere near as popular as C or C++.



Time to Get Coding

- » Things you'll need:
 - Ideally some form of Ubuntu Machine/VM.
 - GCC – Run `sudo apt install build-essential` from the terminal to install it.
 - Some form of text editor – I personally recommend something like VSCode but anything that colour codes the C will do.
- » If you're using Windows, I'd recommend getting WSL set up and use that to program C in.



C Programs 101

- » The program starts with `#include <stdio.h>`.
 - This is the **ST**andard **I**nput **O**utput library.
 - It contains functions that allow for input and output from the program.
 - Many more libraries exist, but for now all we need is `stdio`.
- » All C programs must contain a function called `main`.
 - Everything in the function is contained in the curly brackets.
 - Every C program will run the `main` function first.
- » Next, we have a function called `puts`.
 - This prints text to the terminal, followed by a new line.
 - It's like a function called `printf`, but is less powerful.
 - We need a semi-colon at the end of the line – this indicates the end of an instruction.
- » Finally, the `main` function runs `return`.
 - This signifies the end of the function. In `main`, it signifies the end of the program.
 - It returns a number, generally either 1 or 0.
 - 0 = Success, 1 = Fail.
- » Comments allow you to annotate your code.
 - All comments are ignored by the compiler.
 - Single line comments are signified by `//`.
 - Multi-line comments are signified by `/*<comment>*/`.

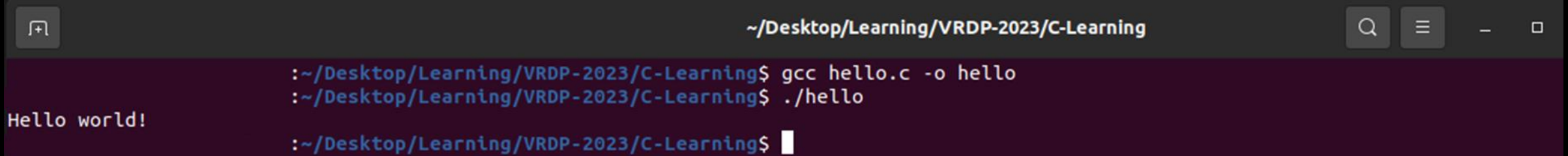
```
1  ✓ /*
2   A C program to print a message to the terminal
3   Also this is an example of a multi-line comment
4   Pretty cool stuff
5   */
6
7   #include <stdio.h>
8
9   int main()
10  {
11      // This prints to the console
12      puts("Hello world!");
13      return(0);
14  }
```

Compile Compile Compile

» It's all well and good writing code, but how do we run it?

» It's a simple three step process!

1. Open a terminal in the folder your program is in.
2. Type in: `gcc <your program name> -o <what you want the executable to be called>.`
 - For example: `gcc hello.c -o hello.`
3. Type `./<executable name>` and it will run!



```
~/Desktop/Learning/VRDP-2023/C-Learning
Hello world!
~/Desktop/Learning/VRDP-2023/C-Learning$ gcc hello.c -o hello
~/Desktop/Learning/VRDP-2023/C-Learning$ ./hello
~/Desktop/Learning/VRDP-2023/C-Learning$
```

Task Time - My First Program

1. Copy the program from slide 4. Compile with GCC and run it. You can call it what you like.
2. Add a line in underneath the first `puts` statement to print “I wrote this in C!”.

Bonus: Use a `printf` statement to print out the line “And I wrote this with printf” after the previous line. Your `printf` must add a new line after the string.

(If you’re struggling with the bonus don’t worry – we’ll go through `printf` later).



Storing Stuff

- » It's all well and good printing stuff to the screen, but ideally, we'd like to store values too.
- » Declaring variables in C is different from languages such as Python, due to you having to declare both the variable name and type.
- » These are the main types of variable you should know – there are more, but for the exercises here this is all you'll need.
 - `int` – Stores whole numbers. 4 bytes in size.
 - `float` – Stores non-whole numbers. 4 bytes in size.
 - `char` – Stores characters. 1 byte in size.
 - `_Bool` – Stores either 1 or 0 – True/False. 1 byte in size.
- » You can declare a variable without giving it a value, then assign it one later!
 - You can also declare multiple variables of the same type on one line.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int emptyNumber, number = 10;
6      float decimal = 3.14;
7      char letter = 'a';
8      _Bool yes = 1;
9
10     emptyNumber = 5;
11 }
```

Wait... Where's the String?

- » You may have noticed that there's no variable type for strings.
- » Well...
- » C has no explicit variable type for strings.
- » Instead, it uses an array of characters to represent strings.
 - All will be explained later. (Ooh, foreshadowing...)
- » For now, if you need to store a string, use this method:
 - `char aString[] = "This is a string"`

```
1  #include <stdio.h>
2
3  int main()
4  {
5      char singleLetter = 'a';
6      char string[] = "I am a string";
7  }
```


Storing Stuff++

- » You can also store variable types that you don't want to change.
- » This is done using the `#define` keyword.
 - This keyword generally goes after all the `#include` lines in a C Program.
 - It assigns a value to a variable, which can't be changed. (At least not easily...)
 - It's useful if you have a value that you never want to be edited, like a mathematical constant.
- » `#define` lines don't need a semi-colon at the end of them, just like `#include` lines.

```
1  #include <stdio.h>
2
3  #define TRUE 1
4  #define FALSE 0
5  #define PI 3.14
6
7  ✓ int main()
8  {
9      float pi = PI;
10     _Bool yes = TRUE;
11 }
```

Printf me Some Stuff!

- » Whilst puts is useful for strings, we ideally want to be able to print more than just strings.
- » This is where `printf` comes in – the **PRINT** Formatted.
- » It allows us to print formatted strings, meaning we can now include integers, floats, hex, etc.
- » Example: Copy the code below. Compile and run it.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      float pi = 3.14;
6
7      printf("Did you know that %d is an integer, ", 20);
8      printf("and that %f is a float?\n", pi);
9      printf("Printf is very powerful!\n");
10
11     return(0);
```

Well, That Was Odd...

- » If you're new to C, you probably expected the output to look something like this:

```
user@machine:~/Desktop/Work/Programs/C/VRDP-2023$ gcc printfExample.c
user@machine:~/Desktop/Work/Programs/C/VRDP-2023$ ./a.out
Did you know that %d is an integer,
and that %f is a float?
Printf is very powerful!
user@machine:~/Desktop/Work/Programs/C/VRDP-2023$
```

- » But what you get is this:

```
user@machine:~/Desktop/Work/Programs/C/VRDP-2023$ ./a.out
Did you know that 20 is an integer, and that 3.140000 is a float?
Printf is very powerful!
user@machine:~/Desktop/Work/Programs/C/VRDP-2023$
```

- » `printf` makes use of conversion characters – these are represented by the `%` symbol followed by a character, and are used to act as placeholders for variables we want to replace them with.
- » It also makes use of escape sequences – these are used to represent characters you can't normally type on a keyboard, like newlines and tabs.

Handy Tables

Escape Sequence	Character Produced
\a	Bell (Computer goes beep!)
\b	Backspace, Non-Erasing
\f	Form Feed – Clear the screen
\n	Newline
\t	Tab
\\	Backslash
\?	Question Mark
\"	Quotation Mark

Conversion Character	Value Held
%c	Char
%s	String/Char[]
%d	Int
%f	Float
%u	Unsigned Int
%x	Hexadecimal

Task Time - Variables

That's enough reading, let's do some programming!

1. Create a program that assigns a float to the variable `jebediah`, a string to the variable `bill`, and an integer to the variable `bob`. Print out these values using `printf`, so the output looks like this:

```
Jebediah holds the float 3.140000
Bill holds the string Hello world!
Bob holds the integer 180
```

2. The circumference of a circle is calculated by multiplying its diameter by pi. Write a program that takes a variable storing a diameter and prints the area of its relative circle. You must use a constant to store the value of pi, which is 3.14159.
3. Modify your above program so that it displays the area of your circle as well.

Bonus: Modify the circle programs again so that the answers are given to two decimal places.

Scan-demonium

- » Often, we need to let the user provide input to the program, rather than relying on variables to store everything we need.
- » For this, we use the `scanf()` function – **SCAN** Formatted.
- » The format of `scanf` is: `scanf("conversion", variable)`, where:
 - `conversion` = A conversion character representing the expected input type.
 - `variable` = The variable we want to store the input in.
- » Unless what you want to read in is a string, the variable is prefixed with `&`.
 - Much like strings being arrays, this will be explained later...
- » See the image for some examples!

```
// Scan to an integer variable
scanf("%d", &myInt);

// Scan to a float variable
scanf("%f", &myFloat)

// Scan to a character variable
scanf("%c", &myChar);

// Scan to a character array (A string)
scanf("%s", myString)
```

Printf & Scanf = Good Times

- » So, how do we use `scanf`?
- » Generally, regarding user input, you'd expect to get some after you have prompted the user for input.
- » For this, we can use a `printf` to print a prompt to the user!
 - Sadly, `puts` can't be used for `scanf` input, but that's okay because `printf` exists!
- » Exercise: Copy the below code, compile, and run. You should be able to input a string!
 - You'll notice that for the string, we need to create a character array.
 - Remember slide 8? Strings in C are character arrays in disguise.
 - So, to store arrays from input, we need to make sure we have an array ready to store it in that's big enough.
 - Using a constant to store the buffer size is a very good idea!

```
1  #include <stdio.h>
2
3  #define BUFF 255
4
5  int main()
6  {
7      char inputBuff[BUFF];
8
9      printf("Enter your name: ");
10     scanf("%s", inputBuff);
11
12     printf("Hello, %s! Hope you're enjoying C!\n", inputBuff);
13     return(0);
14 }
```

Task Time - Scanning

1. Expand on the example from the last slide to calculate the year the user was born. The user should be prompted to enter the year they were born after they have entered their name. The output should look like this:

“Hello <name>! You were born in <year>.”

Hint: You may encounter a problem where the second input is skipped. This is due to the computer treating the enter you press as input and sends that as input to the second prompt. To mitigate this, add `while (getchar() != '\n');` after your `scanf`. This clears the input buffer properly!

2. Expand your circle code to ask the user to input a diameter instead of simply using a variable for it. The user should be able to input a decimal.

Decisions, Decisions...

- » It's a good idea to allow your computer to make decisions at times. Sure, it may not be sentient yet, but it's a powerful way to expand the capabilities of your programs.
- » We'll start with the simplest decision maker – the `if` statement.
 - The `if` statement takes in an evaluation, also known as a condition. This condition can be a comparison, mathematical operation, function result, or other condition.
 - If the condition is true, the program will run the code contained in the curly brackets below it.
 - Any value other than 0 is considered True in C.
 - Conversely, if the statement is false, the program won't run the code.
 - `if` statements can't compare strings sadly – strings need their own special functions for comparisons.
- » Exercise: Copy the example, compile, and run it. Play around with the values and see what happens.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int a = 6;
6      int b = 4;
7
8      if(a > b)
9      {
10         printf("%d is greater than %d", a, b);
11     }
12
13     return(0);
14 }
```

I if, Therefore I else

- » Sometimes, if a condition isn't met, we want to run a piece of code after that lets the user know the condition wasn't met. However, if we have just an if statement, we run into problems.
 - Give it a go yourself: Add a line in your previous program after your if statement (on the line above your return statement) which prints a message saying b is bigger than a. Notice what happens when you compile and run...
- » To get around this, we use what's known as an `else` statement.
 - This is a block of code that runs if the previous if statement returned false.
 - It allows us to control what happens, rather than just simply returning to normal execution.
- » Exercise: Copy the code below to add an `else` statement to the program. Compare what happens now to what happened without the else statement.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int a = 6;
6      int b = 4;
7
8      if(a > b)
9      {
10         printf("%d is greater than %d", a, b);
11     }
12     else
13     {
14         printf("%d is smaller than %d", a, b);
15     }
16
17     return(0);
18 }
```

I if, Therefore I else if, Therefore I else

- » Sure, it's good to compare against one condition, but what if we want to go crazy and compare more than one condition?
- » Thankfully, we can chain together `if` statements one after the other using an `else if` block.
 - This allows us to specify another condition check after the first one.
 - We could use another single `if` statement, but this is a more graceful.
- » Exercise: Copy the code to add a second check to see if the two numbers are equal. Compile and run.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int a = 6;
6      int b = 4;
7
8      if(a > b)
9      {
10         printf("%d is greater than %d", a, b);
11     }
12     else if(a == b)
13     {
14         printf("The two numbers are equal");
15     }
16     else
17     {
18         printf("%d is smaller than %d", a, b);
19     }
20
21     return(0);
22 }
```

Handy Tables: Comparisons

Code	Comparison
<code>a > b</code>	Is a greater than b?
<code>a < b</code>	Is a smaller than b?
<code>a == b</code>	Is a equal to b?
<code>a != b</code>	Is a not equal to b?
<code>a >= b</code>	Is a greater than or equal to b?
<code>a <= b</code>	Is a less than or equal to b?
<code>a < c && b < c</code>	True when both comparisons are true
<code>a < c b < c</code>	True when either comparison is true
<code>!a</code>	True when a is false (0)

Task Time - Decision Making

Now we know a bit more about comparisons, let's put your knowledge into action!

- » Create a program where the user must guess a secret number.
 - The program should store the secret number as a constant.
 - The program should ask the user to guess a number.
 - If the user guesses the number, the program should print out a message saying they've won.
 - If the user's guess is within 5 of the actual number, the program should let the user know their guess was close.
 - For any other number, the program should tell the user to try again.

Switching Things Up

- » The `switch-case` statement is like an if statement, however we can code decisions based on a single value.
 - But wait – can't we do this with if statements?
 - Well, we can, but only if things are true or false – `switch-case` works for multiple single values!
- » Exercise: Copy the code below and run it a few times with different values – we'll go through it on the next slide.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      char code;
6
7      printf("Please enter a letter (A-C): ");
8      scanf("%c", &code);
9
10     switch(code)
11     {
12         case 'A':
13             puts("Jebediah's rocket blew up on launch. Oops.");
14             break;
15         case 'B':
16             puts("Bill got to space but then exploded. Whoops.");
17             break;
18         case 'C':
19             puts("Bob actually made it to the moon! But you forgot his oxygen. Uh oh.");
20             break;
21         default:
22             puts("That's not even a rocket!");
23     }
24
25     return(0);
26 }
```

(Nintendo) Switch Explained

- » The `switch-case` starts with the `switch` statement.
 - This takes in the value to evaluate in brackets.
 - Everything in the curly brackets is encompassed in the statement.
- » Each `case` shows a single value followed by a colon.
 - If the value matches the one in the switch statement, the code underneath it will run!
- » Between each `case` statement is a `break`.
 - This stops the program flowing through every other option.
- » Each `switch` statement should also contain a `default` statement.
 - If none of the case statements match the input, the code in the `default` section is run.

```
10  switch(code)
11  {
12      case 'A':
13          puts("Jebediah's rocket blew up on launch. Oops.");
14          break;
15      case 'B':
16          puts("Bill got to space but then exploded. Whoops.");
17          break;
18      case 'C':
19          puts("Bob actually made it to the moon! But you forgot his oxygen. Uh oh.");
20          break;
21      default:
22          puts("That's not even a rocket!");
23  }
```

Time to Get Loopy

- » Often when programming, we want to be able to run certain sections of code multiple times over.
- » We *could* copy the section we want to repeat over and over, but surely there's a more elegant solution...
- » This is where the `for` statement comes in!
 - They're a little different to loops in other languages, but we'll go through them.
- » For now, copy down the example below – we'll go through it in the next slide!
 - You'll notice that the loop doesn't use curly brackets – what's going on here?!
 - In C, if your loop or decision (if statement) only has one line of code to run, you can omit the curly brackets!
 - Don't try and get away with more than one line though, C doesn't know how to handle that and will throw an error at you.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int x;
6
7      for(x = 0; x < 10; x++)
8          printf("This is loop number %d\n", x + 1);
9
10     return(0);
11 }
```


10: Loop Sweet | 20: goto 10

» Let's break down that `for` statement:

- The `x = 0;` is the initialisation – most of the time, it's where the variable used to keep track of the loop is set or initialised.
- The `x < 10;` is the exit condition – this is a test to determine if the loop should stop, and takes the form of a comparison.
- The `x++;` is the repeat statement – this is run once every iteration.
 - This one adds one to the value of `x` each loop.
 - But why is it not `x = x + 1`?
 - C programmers are lazy, and to save time they shortened down common operations. In this case, `++` means “Add one to the variable”.
 - Conversely, `--` means “Subtract one from the variable”.

```
7   for(x = 0; x < 10; x++)
8       printf("This is loop number %d\n", x + 1);
```

Getting Even Loopier

- » As well as sitting on their own, we can also combine for loops into what's called a nested for loop.
 - This is just a fancy term for a for loop sitting inside a for loop.
- » These are very useful for running an overall loop, where something loops faster than the other loop.
- » As an example, copy the below code and run it. Note what the output looks like.
 - You should notice that the letters increment slower than the numbers.
 - Wait... incrementing characters? But they're not numbers!
 - C can treat a character as either a letter or its integer representation.
 - If you're still confused, have a look at an ASCII table – each letter has a numerical equivalent.
 - Ergo, we can loop on characters too!

```
1  #include <stdio.h>
2
3  int main()
4  {
5      for(char x = 'A'; x < 'D'; x++)
6      {
7          for(int y = 0; y < 4; y++)
8              printf("%c%d\t", x, y);
9          putchar('\n');
10     }
11
12     return(0);
13 }
```

While We're on Loops...

- » Along with for loops, there's another called the `while` loop.
- » These are easier to form than for loops, but require a more careful setup...
- » Exercise: Copy the below example. There's not much to explain so we'll do that here.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int x = 0;
6
7      while(x < 10)
8      {
9          printf("This is loop %d\n", x + 1);
10         x++;
11     }
12
13     return(0);
14 }
```

- » The `while` loop takes a comparison and checks if the condition has been met.
 - If the condition has not been met, the `while` loop runs the code in the brackets, then loops to check the condition.
 - Once the condition has been met, the code moves on.
 - Be very careful with `while` loops – you need to make sure the condition is met at some point, or the code will run forever!

While's Sibling - Do

- » Sometimes with a `while` loop, we want the loop to occur at least once, even if the condition isn't met.
- » To achieve this, we can use something called a `do...while` loop.
 - This is exactly like a while loop, except the code inside the loop is run at least once.
- » Take a look at the example below – feel free to copy and run it too!

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int x = 0;
6
7      do
8      {
9          printf("This is loop %d\n", x + 1);
10     } while (x != 0);
11
12
13     return(0);
14 }
```

- » The print statement, if it was in a `while` loop, wouldn't run at all as the condition is immediately met.
- » However, since it sits in a `do...while` loop, the program will run at least once!

Task Time - Looping

That's loops in the bag now – time for some quick exercises!

1. Remember the secret number guesser? Modify that program so that it runs until the user guesses the secret number.
2. A Fibonacci sequence is a sequence where the next number in the sequence is formed by adding together the two numbers that come before it. Write a program which prints out a list of Fibonacci numbers. The user should be able to input how many numbers in the sequence to print.
3. Write a program which prints out each three-letter combination of A – Z (AAA, AAB, AAC ... ZZX, ZZY, ZZZ). Each combination should be separated by a newline.

Challenge One

- » Write a program to find the most chicken nuggets you can buy with a particular budget from McDonald's.
- » Currently, the prices of chicken nuggets are:
 - 6 Chicken Nuggets = £4.69
 - 9 Chicken Nuggets = £5.19
 - 20 Chicken Nuggets = £5.99
- » The program must take in a user input for their budget.
- » The program must output how many chicken nuggets the user can buy with that budget.
- » The program must tell the user how much change they will get from their order.
- » The program must be able to catch errors, such as an invalid budget, or if no chicken nuggets can be bought.

