

# INDIVIDUAL ASSESSMENT COVER SHEET

Faculty of Design and Creative Technologies

AUT

TE WĀNANGA ARONUI  
O TĀMAKI MAKAU RAU

First Name	Francisca	Family Name	Nel	Student ID No	21145382
Course Name	Programming Concepts and Techniques	Course Code	COMP500	Assignment Due Date	8/04/2022
Lecturer	Steffan Hooper	Tutorial Day	Thursday	Date Submitted	8/04/2022
Tutor	Xinyu Hu	Tutorial Time	12:10pm	No.Words/Pages	16,047 words/ 121 pages

In order to ensure fair and honest assessment results for all students, it is a requirement that the work that you hand in for assessment is your own work. If you are uncertain about any of these matters, then please discuss them with your lecturer.

Plagiarism and Dishonesty are methods of cheating for the purposes of General Academic Regulations (GAR)  
<http://www.aut.ac.nz/calendar>

**Assignments will not be accepted if this section is not completed and signed.**

Please read the following and tick ✓ to indicate your understanding:

1. I understand it is my responsibility to keep a copy of my assignment.
2. I have signed and read the **Student's Statement below**.
3. I understand that a software programme (Turnitin) that detects plagiarism and copying may be used on my assignment.

Yes

No

Yes

No

Yes

No

## Student's Statement:

This assessment is entirely my own work and has not been submitted in any other course of study. I have submitted a copy of this assessment to Turnitin, if required. In this assessment I have acknowledged, to the best of my ability:

- The source of direct quotes from the work of others.
- The ideas of others (includes work from private or professional services, past assessments, other students, books, journals, cut/paste from internet sites and/or other materials).
- The source of diagrams or visual images.

Student's Signature:



Date:

8/04/2022

The information on this form is collected for the primary purpose of submitting your assignment for assessment. Other purposes of collection include receiving your acknowledgement of plagiarism policies and attending to administrative matters. If you choose not to complete all questions on this form, it may not be possible for the Faculty of Design and Creative Technologies to accept your assignment.

**Timesheet**

Entry ID	Week	Date	Start Time	Duration	Session Type	Location
1	1	28/02/2022	8:00am	1 hr	Lecture and Q&A	Online/Home
2	1	28/02/2022	5:38pm	0.5 hr	Self-directed	Home
3	1	2/03/2022	11:00am	1 hr	Lecture and Q&A	Online/Home
4	1	2/03/2022	3:30pm	6 hr	Self-directed	Home
5	1	3/03/2022	12:10pm	3 hr	Lab Tutorial	Online/Home
6	1	4/03/2022	3:00pm	1 hr	Lecture (pre-rec)	Online/Home
7	2	7/03/2022	12:00pm	1 hr	Lecture (pre-rec)	Online/Home
8	2	7/03/2022	5:00pm	5.5 hr	Self-directed	Home
9	2	9/03/2022	11:00am	1 hr	Lecture (pre-rec)	Online/Home
10	2	9/03/2022	3:00pm	3.5 hr	Self-directed	Home
11	2	10/03/2022	12:10	3 hr	Lab tutorial	Online/Home
12	2	11/03/2022	3:00pm	1 hr	Lecture (pre-rec)	Online/Home
13	3	14/03/2022	10:00am	1 hr	Lecture (pre-rec)	Online/Home
14	3	14/03/2022	6:00pm	2.5 hr	Self-directed	Home
15	3	16/03/2022	11:00am	1 hr	Lecture (pre-rec)	Online/Home
16	3	17/03/2022	12:10pm	3 hr	Lab Tutorial	Online/Home
17	3	25/03/2022	1:00pm	1 hr	Lecture (pre-rec)	Online/Home
18	4	21/03/2022	11:00am	1 hr	Lecture (pre-rec)	Online/Home
19	4	23/03/2022	3:00pm	1 hr	Lecture (pre-rec)	Online/Home
20	4	23/03/2022	5:00pm	4.5 hr	Self-directed	Home
21	4	24/03/2022	12:10pm	3 hr	Lab tutorial	Online/Home
22	4	25/03/2022	4:00pm	1 hr	Lecture (pre-rec)	Online/Home
23	4	25/03/2022	5:30pm	2.5 hr	Self-directed	Home
24	5	28/03/2022	10:00am	1 hr	Lecture (pre-rec)	Online/Home
25	5	30/03/2022	11:00am	1 hr	Lecture (pre-rec)	Online/Home
26	5	31/03/2022	12:10am	3 hr	Lab tutorial	Online/Home
27	5	1/04/2022	8:40am	1 hr	Lecture (pre-rec)	Online/Home
28	5	2/04/2022	3:00pm	3.5 hr	Self-directed	Home
29	6	4/04/2022	2:00pm	1 hr	Lecture (pre-rec)	Online/Home
30	6	6/04/2022	9:00am	1 hr	Lecture (pre-rec)	Online/Home
31	6	7/04/2022	12:10pm	3 hr	Lab Tutorial	WZ514
32	6	7/04/2022	5:30pm	6.5 hr	Self-directed	Home
33	6	8/04/2022	12:00pm	1 hr	Self-directed	Home
<b>TOTAL</b>				<b>71 hr</b>		

# WEEK 1

## Week 1 lecture 001 Notes- Sequence, Text Output, Visual Studio

WRITTEN 7/02/2022- Monday, Week 2

Notes Retrieved from Steffan Hooper, W01 Lec 001, 2022

USE VISUAL STUDIO ENTERPRISE 2017

Always start program with:

- `#include <stdio.h>` for access to print
- Next, I need to write '`int main(void)`' followed by curly brackets {}. My code needs to be contained within these brackets
- The code is ended with '`return 0;`'

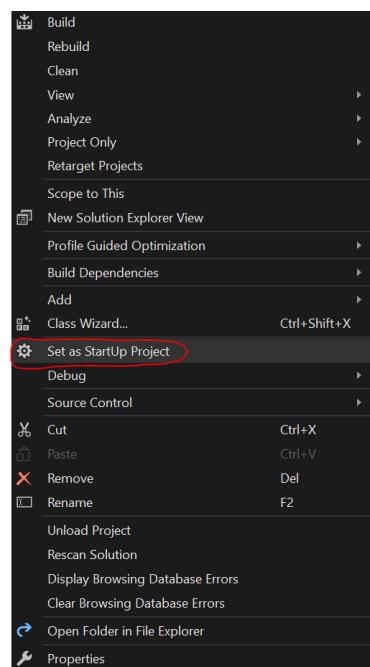
Printf is used to print. Ex: `printf("Hello world!");`

New lines are ended with the semicolon ';' as opposed to python which did not have this

Escape sequences need to be used to print certain characters properly. Example of escape sequences:

Escape Sequence	Character Representation
<code>\n</code>	newline
<code>\</code>	?
<code>\"</code>	Double quotes
<code>\'</code>	Single quote

SCREENSHOTTED FROM OWN INSTALLATION OF VISUAL STUDIO ENTERPRISE 2017:



Right click a project and click "set up as startup project" to run the specific project you are working on

Note: I completed all lab 1's work before week 1's lab

## LAB 1: ASCII ART

COMPLETED 28/02/2022 –Monday, Week 1

WRITTEN 7/02/2022- Monday, Week 2

Source Code (Input)	Output & discussion
1x01a – Welcome Text	
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5 6      printf("Hello\n"); 7      printf("    Programming\n"); 8      printf("        Students\n"); 9 10 } 11 </pre>	 <p>Output:</p> <p>Hello Programming Students</p> <p>Note: I have to use spaces instead of tabs in the print because the output uses spaces only</p>
1x02a – Centered Triangle	
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      printf("    /\\"\\n"); 6      printf("    /##\\\"\\n"); 7      printf("    /####\\\"\\n"); 8      printf("    /#####\\\"\\n"); 9      printf("    /#####\\\"\\n"); 10 11 } 12 </pre>	 <p>Output:</p> <p>/\\" /##\" /###\" /####\" /#####\" /#####\"</p> <p>Note: remember "\\" is the escape sequence for outputting single backslash</p>
1x03a – The Rocket	
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5 6      printf("    /\\"\\n"); 7      printf("    /  \\\"\\n"); 8      printf("    /    \\\"\\n"); 9      printf("    +-----+\\n"); 10     printf("            \\n"); 11     printf("            \\n"); 12     printf("            \\n"); 13     printf("    +-----+\\n"); 14     printf("            \\n"); 15     printf("      N    \\n"); 16     printf("        Z  \\n"); 17     printf("     Rocket \\n"); 18     printf("    +-----+\\n"); 19     printf("            \\n"); 20     printf("            \\n"); 21     printf("            \\n"); 22     printf("    +-----+\\n"); 23     printf("    /  \\\"\\n"); 24     printf("    /  \\\"\\n"); 25 26 } 27 </pre>	 <p>Output:</p> <p>A diagram of a rocket ship with a dashed outline. The word "Rocket" is printed inside the body of the rocket.</p>

## Wednesday Week 1 lecture 002 Notes- Solving Programming Problems

Notes Retrieved from Steffan Hooper, W01 Lec 002, 2022

**WRITTEN 7/02/2022- Monday, Week 2**

Input → Process → Output

Think of the input and the output first, then figure out the process. The process is the steps required to reach the goal (output). Example table:

Input-Process-Output of Fahrenheit to Celsius		
Input	Process	Output
User inputs degrees Fahrenheit as a real number	Convert using formula $(Fahrenheit - 32) / (9/5)$	Program displays user input as degrees Celsius

## LAB 1: ASCII ART

**COMPLETED 2/03/2022 - Wednesday, Week 1. WRITTEN 7/02/2022- Monday, Week 2**

**LAB 1: ASCII ART**

COMPLETED 2/03/2022 - Wednesday, Week 1. WRITTEN 7/02/2022- Monday, Week 2

Source Code (Input)	Output & discussion
<pre> 1   #include &lt;stdio.h&gt; 2 3   int main(void) 4   { 5       printf("\n Auckland\n\n"); 6       printf("University\n\n"); 7       printf("    of\n\n"); 8       printf("Technology\n\n"); 9 10      return 0; 11  } </pre>	<p align="center"><b>1x01c – Print AUT</b></p>  <p>I got confused on this exercise because I forgot the new line before the first word! Remember to check for newlines.</p>
<pre> 1   #include &lt;stdio.h&gt; 2 3   int main(void) 4   { 5       printf("C source code has an entry point named main.\n\n"); 6 7       printf("The main has { and } braces, inside of which there are\n"); 8       printf("instructions for the machine to run in sequence.\n\n"); 9 10      printf("The printf functionality sends text to the screen. To call\n"); 11      printf("printf the programmer must use (and) brackets after the\n"); 12      printf("printf, inside of which there is a piece of text. The text\n"); 13      printf("must be enclosed in \" quotes.\n\n"); 14 15      printf("Each instruction ends with a ; followed by a newline.\n\n"); 16 17      printf("Source code must successfully compile before the program can\n"); 18      printf("be run. Programmers test their programs by running them to\n"); 19      printf("ensure they function as expected!\n\n"); 20 21      return 0; 22  } </pre>	<p>C source code has an entry point named main.</p> <p>The main has { and } braces, inside of which there are instructions for the machine to run in sequence.</p> <p>The printf functionality sends text to the screen. To call printf the programmer must use (and) brackets after the printf, inside of which there is a piece of text. The text must be enclosed in " quotes.</p> <p>Each instruction ends with a ; followed by a newline.</p> <p>Source code must successfully compile before the program can be run. Programmers test their programs by running them to ensure they function as expected!</p> <p>This exercise did not have feedback mode, but I still know when I have made an error in the output log due to the build failing when using incorrect syntax</p>
<pre> 1   #include &lt;stdio.h&gt; 2 3   int main(void) 4   { 5       printf("COMP500: Programming 1\n"); 6       printf("ENSE501: Programming for Engineering Applications\n\n"); 7 8       printf("COURSE DESCRIPTOR: An introduction to the basics of computer\n"); 9       printf("programming to equip students for a career in any branch of\n"); 10      printf("IT, the sciences, data analysis or engineering. The\n"); 11      printf("fundamentals of writing, designing and testing programs will\n"); 12      printf("be developed.\n\n"); 13 14      printf("WORKLOAD: As a general guide, the average student should\n"); 15      printf("expect to spend about 150 hours on a 15 point course. This\n"); 16      printf("averages to be approximately 10 hours per week, for 12 weeks\n"); 17      printf("of teaching and a 3-week exam period per semester\n\n"); 18 19      return 0; 20  } </pre>	<p>COMP500: Programming 1 ENSE501: Programming for Engineering Applications</p> <p><b>COURSE DESCRIPTOR:</b> An introduction to the basics of computer programming to equip students for a career in any branch of IT, the sciences, data analysis or engineering. The fundamentals of writing, designing and testing programs will be developed.</p> <p><b>WORKLOAD:</b> As a general guide, the average student should expect to spend about 150 hours on a 15 point course. This averages to be approximately 10 hours per week, for 12 weeks of teaching and a 3-week exam period per semester</p> <p>This exercise had feedback mode, so it was easier to see grammatical accuracy and syntax mistakes.</p>

**LAB 1: ASCII ART**

COMPLETED 2/03/2022 - Wednesday, Week 1. WRITTEN 7/02/2022- Monday, Week 2

Source Code (Input)	Output & discussion
<b>1x02b – X in Square</b>	
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      printf("#-----#\n"); 6      printf("   x       x  \n"); 7      printf("   x       x  \n"); 8      printf("   x   x    \n"); 9      printf("   x        \n"); 10     printf("   x   x    \n"); 11     printf("   x       x  \n"); 12     printf("             x  \n"); 13 14 } 15 16 }</pre>	
<b>1x02c – Right Angle Triangle</b>	
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      printf("#\n"); 6      printf("##\n"); 7      printf("###\n"); 8      printf("####\n"); 9      printf("#####\n"); 10 11 } 12 13 }</pre>	
<b>1x02d – Small Mouse</b>	
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      printf("(\\____/\n"); 6      printf("=(o.o)= \n"); 7      printf("(\")_(_\" ) \n"); 8 9 10 } 11 }</pre>	

**LAB 1: ASCII ART**

COMPLETED 2/03/2022 - Wednesday, Week 1. WRITTEN 7/02/2022- Monday, Week 2

Source Code (Input)	Output & discussion
<b>1x02e – Small Cat</b>	
<pre> 1   #include &lt;stdio.h&gt; 2 3   int main(void) 4   { 5       printf(" /\_/\_\ \n"); 6       printf("( 0.0 ) \n"); 7       printf(" &gt; ^ &lt; \n"); 8 9 10    return 0; 11 }</pre>	 <p>The output shows a small cat's head and body made of ASCII characters. It has a triangular face with a circle for an eye, a vertical line for a nose, and a curved line for a mouth. Its body is a simple triangle.</p>
<b>1x03b – Three Big Letters</b>	
<pre> 1   #include &lt;stdio.h&gt; 2 3   int main(void) 4   { 5       printf("*****      *****      **      **\n"); 6       printf("**      **      **      **\n"); 7       printf("**      **      **      **\n"); 8       printf("**      **      **      **\n"); 9       printf("*****      **      *****\n"); 10      printf("      **      **      **\n"); 11      printf("      **      **      **\n"); 12      printf("      **      **      **\n"); 13      printf("*****      *****      **      **\n"); 14 15 16    return 0; 17 }</pre>	<p><b>Feedback Mode:</b> Output appears to be correct. Well done</p> <p><b>Debug:</b></p>  <p>The output shows three large letters 'A', 'B', and 'C' composed of asterisks (*). Each letter is a 5x5 grid of asterisks, with the central column being shorter than the outer columns.</p>
<b>1x03c – Mirrored Triangles</b>	
<pre> 1   #include &lt;stdio.h&gt; 2 3   int main(void) 4   { 5       printf(" /\_/\_\ \n"); 6       printf(" / \ \ \ \ \n"); 7       printf(" / \ \ \ \ \n"); 8       printf(" / \ \ \ \ \n"); 9       printf("----- \n"); 10      printf(" \n"); 11      printf(" \n"); 12      printf(" _____ \n"); 13      printf(" \ \ \ / \ \ \ \ \n"); 14      printf(" \ \ \ / \ \ \ \ \n"); 15      printf(" \ \ \ / \ \ \ \ \n"); 16      printf(" \ \ \ / \ \ \ \ \n"); 17      printf(" \ \ \ / \ \ \ \ \n"); 18 19 20    return 0; 21 }</pre>	 <p>The output shows a series of mirrored triangles and horizontal lines. It includes a large inverted triangle at the top, followed by several smaller triangles pointing downwards, and a horizontal line near the bottom.</p> <p>I could have used 4 newlines “\n\n\n\n” in a single line between the mirrored triangles to shorten the amount of lines, but it is easier to see what the end result would look like when the line formatting is accurate to what the output is supposed to look like.</p>

**LAB 1: ASCII ART**

COMPLETED 2/03/2022 - Wednesday, Week 1. WRITTEN 7/02/2022- Monday, Week 2

Source Code (Input)	Output & discussion
<b>1x03d – Pattern Printer</b>	
<pre> 1   #include &lt;stdio.h&gt; 2 3   int main(void) 4   { 5       printf(" * P R O G R A M M I N G 1 * \n"); 6       printf("C           E \n"); 7       printf("O           N \n"); 8       printf("M           S \n"); 9       printf("P           E \n"); 10      printf("5           5 \n"); 11      printf("0           0 \n"); 12      printf("0           1 \n"); 13      printf("           * \n"); 14 15 16      return 0; 17 }</pre>	
<b>1x03e - Snowflake</b>	
<pre> 1   #include &lt;stdio.h&gt; 2 3   int main(void) 4   { 5       printf("          /\ \n"); 6       printf("         //\\ \n"); 7       printf("        \\\\" \n"); 8       printf("        \\\\ \n"); 9       printf("        \\\\ \n"); 10      printf("        \\\\ \n"); 11      printf("        \\\\ \n"); 12      printf("        \\\\ \n"); 13      printf("        \\\\ \n"); 14      printf("        \\\\ \n"); 15      printf("        \\\\ \n"); 16      printf("        \\\\ \n"); 17      printf("        \\\\ \n"); 18      printf("        \\\\ \n"); 19      printf("        \\\\ \n"); 20      printf("        \\\\ \n"); 21      printf("        \\\\ \n"); 22      printf("        \\\\ \n"); 23 24      return 0; 25 }</pre>	<p>This exercise was very hard due to the large number of escape sequences. CTRL+F was my best friend to find the mistakes.</p>
<b>1x04b – Sailboat</b>	
<pre> 1   #include &lt;stdio.h&gt; 2 3   int main(void) 4   { 5       printf("          + \n"); 6       printf("         /  \n"); 7       printf("        /  + \n"); 8       printf("        /   \\ \n"); 9       printf("        /    \\ \n"); 10      printf("        +----+----+ \n"); 11      printf("        \% _____+ \n"); 12      printf("        \\\ FRAN'S BOAT / \n"); 13      printf("        \\\_____\n"); 14 15      return 0; 16 17 18 19 }</pre>	<p>Of all things I could print into this boat, I chose to write "FRAN'S BOAT"</p>

**LAB 1: ASCII ART**

COMPLETED 2/03/2022 - Wednesday, Week 1. WRITTEN 7/02/2022- Monday, Week 2

Source Code (Input)	Output & discussion
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      printf("    /\\"    /\\"    \n"); 6      printf("   //\\__/_//\\\"    \n"); 7      printf("   /           \\\"    \n"); 8      printf(" /     (o)   (o)   \\\"    \n"); 9      printf("         Y           \n"); 10     printf(" \\ == = _ _==    / \n"); 11     printf(" \\\"  \\-----/ \n"); 12     printf("   \\-----/ \n"); 13     printf("   ---(P%1)--- \n"); 14     printf(" //-----\\\"\\\" \n"); 15 16 17 18     return 0; 19 }</pre>	<p style="text-align: center;"><b>1x04c – The Cat</b></p>
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      printf(" _____ \n"); 6      printf(" / . . \\\" \n"); 7      printf(" / . . \\\" \n"); 8      printf("-\"-\\'-\"-\\'-\"- \n"); 9      printf("\\===== \n"); 10     printf(" \\_____\n"); 11 12 13     return 0; 14 }</pre>	<p style="text-align: center;"><b>1x04d – Cheese Burger</b></p>
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      printf(" _[u]_ \n"); 6      printf(" [ununu] \n"); 7      printf(" [ununun] \n"); 8      printf(" [unununun] \n"); 9      printf(" [ununununun] \n"); 10     printf(" [ununununun] \n"); 11     printf(" [unununununun] \n"); 12     printf(" [unununununun] \n"); 13     printf(" [ununununununun] \n"); 14     printf(" [unununununununun] \n"); 15     printf(" [ununununununununun] \n"); 16 17 18     return 0; 19 }</pre>	<p style="text-align: center;"><b>1x04e - Cityscape</b></p>

This exercise was difficult due to the ASCII art becoming displaced in the source code due to the escape sequences. To make it easier, I did the ASCII art without escape sequences and did the escape sequences last so that I know that it looks right before displacing it.

**LAB 1: ASCII ART**

COMPLETED 2/03/2022 - Wednesday, Week 1. WRITTEN 7/02/2022- Monday, Week 2

Source Code (Input)	Output & discussion
<b>1x05b – Alien Face</b>	
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      printf("   _____ \n"); 6      printf("  / \\\n"); 7      printf("  \\\n"); 8      printf("   +--+ +--+   \n"); 9      printf(" (   %%   %%   ) \n"); 10     printf("(y  +--+ +--+  y)\n"); 11     printf(" (   oo   ) \n"); 12     printf("         \n"); 13     printf(" \\ +-----+ / \n"); 14     printf(" \\_____/ \n"); 15 16 17     return 0; 18 }</pre>	
<b>1x05d – Pyramid</b>	
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5 6      printf("         \n"); 7      printf("        /=\\"\\ \n"); 8      printf("       /==\\ \\ \n"); 9      printf("      /====\\' \\ \n"); 10     printf("     /=====\\'' \\ \n"); 11     printf("    /=====\\' ' \\ \n"); 12     printf("   /=====\\' ' ' \\ \n"); 13     printf("  /=====\\' ' ' ' \\ \n"); 14     printf(" /=====\\' ' ' ' ' \\ \n"); 15     printf(" /=====\\' ' ' ' ' ' \\ \n"); 16     printf(" /=====\\' ' ' ' ' ' ' \\ \n"); 17     printf(" /=====\\' ' ' ' ' ' ' ' \\ \n"); 18     printf(" /=====\\' ' ' ' ' ' ' ' ' \\ \n"); 19     printf(" /=====\\' ' ' ' ' ' ' ' ' ' \\ \n"); 20     printf(" /=====\\' ' ' ' ' ' ' ' ' ' ' \\ \n"); 21     printf(" /=====\\' ' ' ' ' ' ' ' ' ' ' ' \\ \n"); 22     printf(" /=====\\' ' ' ' ' ' ' ' ' ' ' ' ' \\ \n"); 23 24     return 0; 25 }</pre>	
<b>1x05e – Cat Fish</b>	
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5 6      printf("   \\__/_  %% \n"); 7      printf("  (_ ^_ ) %% \n"); 8      printf("  _ ) ( )) [^^^] \n"); 9      printf(" ((/_ \\"\\\" \\\\" ( (         \n"); 10     printf(" ( )_\\\\_\\\\_\\\\_\\( ))          \n"); 11     printf(" ( _ _ _ _ _ &gt;++%%&gt;           \n"); 12     printf(" ===== \n"); 13 14     return 0; 15 }</pre>	

## Week 1 work analysis: 1x05c Log Cabin

COMPLETED 2/03/2022 - Wednesday, Week 1. WRITTEN 9/02/2022- Wednesday, Week 2

Aim	Testing/errors/debugging
The aim was to replicate the cabin ASCII art in the task	<p>While attempting to build, I came across errors a few times due to missing % or double backslash \\ escape sequences.</p> <p>To easily identify where these mistakes are, I used 'ctrl+f' to get the find tool, and I inputted '\ or '%' to highlight where I may have missed using proper escape sequences for these symbols.</p> <p>I also made spacing and symbol placement mistakes while trying to make the ASCII art accurate to the example. At times, these were hard to identify because of how subtle the mistake is (like missing a single space)</p>

Implementation	
Source Code (Input)	Output
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  // Log Cabin 5  { 6      printf("      ( ) \n"); 7      printf("      ) ) \n"); 8      printf("      ( ( \n"); 9      printf("      () \n"); 10     printf("      _____[ ]_____ \n"); 11     printf("      /\\ \\\\ /\\ \\\\ /\\ \\\\ /\\ \\\\ \n"); 12     printf("      //---\\ \\\\ /\\ \\\\ /\\ \\\\ /\\ \\\\ \n"); 13     printf("      //----\\ \\\\ /\\ [ ] \\\\ /\\ \\\\ /\\ \\\\ \n"); 14     printf("      //-----\\ \\\\ /\\ [ ] \\\\ /\\ \\\\ /\\ \\\\ \n"); 15     printf("      /xxxxxxxxxxxxxx\\ \\\\ /\\ \\\\ \n"); 16     printf("      / I I I I \\\\ /\\ \\\\ /\\ \\\\ /\\ \\\\ \n"); 17     printf("      I   I [ ]   [ ] I \n"); 18     printf("      I   I [ ]   [ ] I \n"); 19     printf("      I   I %%%%%%%\n"); 20     printf("      I   I I \n"); 21     printf("#####\n"); 22     printf("#####\n"); 23 24     return 0; 25 }</pre>	

It was difficult to see what the art looked like properly in the source code due to the escape sequences, especially the % symbols having to be doubled- thus displacing the source code ASCII art

### Lessons learnt:

- Utilising the find tool (ctrl+f) makes it easier to find escape sequence mistakes
- Implement the escape sequences last so that I can see what the ASCII art looks properly before debugging. When doing this for long rows of %%%%%% (for example), simply copy and paste them to make them double.

## Friday Week 1 lecture 003 Notes- Variables, Assignment, printf

Notes Retrieved from Steffan Hooper, W01 Lec 003, 2022

**WRITTEN 9/02/2022- Wednesday, Week 2**

Variables store information, and they must be named. Variables cannot start with a digit. Each variable must have a type of information specified to it. These types include- float (6 decimal places), decimal (15 decimal places), void (no value).  
Variable declaration: statement that tells compiler type of information to be stored.  
Variable initialisation: setting value to variable when declaring it. Variables don't have a default value when declared.  
Symbols for mathematical operations : add (+), subtract (-), multiply (\*), divide (/), and brackets () may be used. The % operator works out the remainder of two integer values after division.

Compound assignment shortens lines of code. Ex:

```
int f = 10;  
f += 5; (instead of f = f + 5;)
```

Other compound assignment operators: add (+=), subtract (-=), multiply (\*=), divide (/=).

Increment : take value, add 1, and save result back to the variable

```
int e = 10;  
e++ (instead of e = e + 1;)
```

Postfix: for example, the ++ increment happens after a variable is assigned to something:

```
int h = g++; (g is assigned into h and the increment happens after)
```

Prefix: for example, the ++ increment happens before a variable is assigned to something

```
int j = ++i; (increment of i happens and the result is saved to variable j)
```

For input in printf, remember to state the **variable I want to print**, ex:

```
printf("result currently holds: %d.\n", result);
```

Literals/constants are fixed values that cannot change, for example, pi (3.1415...) in a program would be treated as a constant.

Sizeof operation is used to check the number of bytes a variable type has

# WEEK 2

**Week 2 lecture 004 Notes- Addressof, scanf, const, Overflow, Truncation**

Notes Retrieved from Steffan Hooper, W02 Lec 004, 2022

07/03/2022 - Monday

Format specifiers most likely to be used	
Specifier	Output
%d or %i	Prints signed decimal (base 10) integer
%c	Prints single character (ASCII)
%f	Prints decimal floating-point
%s	Prints string of characters

Format Specifiers are placeholders in printing for specifying the type of value I want printed (which I must define with a variable).

Start simple: focus on using the %d and %f format specifiers for now.

The width option controls how many character spaces something takes up when printed. For example:

```
printf("%5d\n%5d\n%5d\n%5d\n", 10, 555, example, 4);
          Constants, variable
```

The '5' before the format specifier is the width option. It creates 5 empty spaces before the digits of the number (but if the number digits are > 5, no empty spaces will be printed)

The Justify option: values can be printed left or right aligned. They are printed right-aligned by default, and left-aligned is activated by adding minus to format specifier, ex: %-5d

The zero-fill option: adding zero to format specifier forces zeros to be printed before it, ex: %05d will print the number with 5 digits, but if value is less than 5 digits, zeroes will be printed before it.

The positive sign is not shown by default (as opposed to negative sign) but can be forced by adding + sign before format specifier. Ex. %+5d

Float numbers: format specifier for float numbers is %f. Decimal places are specified by adding a decimal point before format specifier (followed by number of decimal places wanted), ex: %.f is no decimal points and %.2f is 2 decimal places.

Address-of operator: the & symbol will ask the program for the address and location of memory of variable. Example of address output: 4192392 (decimal address when using %d), and 003FF888 (hexadecimal address when using %p).

scanf: like input() in python. Scanf reads user input, where the value type needs to be specified, then it needs to be saved to a variable. **KEEP MIND: & symbol needs to be placed behind variable.**

ALSO: don't scan for \n. Ex, don't put \n next to specifier like "%d \n".

```
scanf("%d", &a);
```

Use const to avoid magic numbers (values with unnecessary multiple occurrences), ex:

```
const float PI = 3.14159f;
```

Truncation: shorten or cut-off. To truncate a float, for example:

```
float pi = 3.14f;
```

```
int truncated_pi = pi; (truncated_pi will only hold the value 3)
```

int values cannot be divided without being truncated, so use floats when dividing.

- X.0f is 32-bit float, X.0 is 64-bit double (X being any value)

Integer overflow: when numerical value is too large to be represented by the variable type.

## LAB 2: 2x01a – A Question of Age

7/03/2022 - Monday, Week 2.

Program aim		
Prompt user their age and print: "You are x years old", x = age user input		
Input	Process	Output
User Inputs age: ex. 19	Age is saved to integer variable called age	Program prints "You are 19 years old"
Implementation		
Source Code (Input)	Output/s	
<pre>1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      int age = 0; 6 7      printf("What is your age? "); 8 9      scanf("%d", &amp;age); 10 11     printf("\nYou are %d years old.", age); 12 13     return 0; 14 }</pre>	<pre>What is your age? 80 You are 80 years old.</pre> <pre>What is your age? 14 You are 14 years old.</pre> <pre>What is your age? 27 You are 27 years old.</pre>	

**LAB 2: 2x02a – Three Number Reverse**

7/03/2022 - Monday, Week 2

Program aim		
Ask user to input 3 whole numbers, then output the numbers in entry order, followed by reverse order.		
Input-Process-Output plan		
Input	Process	Output
User Inputs three whole numbers (integers).  Numbers are saved to 3 integer variables: <b>a</b> , <b>b</b> , <b>c</b> .	Print using %d (integer format specifier) three times in one line each followed by \n. To print them in order, variables need to be sorted <b>a</b> , <b>b</b> , <b>c</b> . To be reversed, variables need to be sorted <b>c</b> , <b>b</b> , <b>a</b> .	Program prints the numbers in entry order, then reverse order
Implementation		
Source Code (Input)	Output/s	
<pre> 1   #include &lt;stdio.h&gt; 2 3   int main(void) 4   { 5       int a = 0; 6       int b = 0; 7       int c = 0; 8 9       printf("Please enter number 1: "); 10      scanf("%d", &amp;a); 11 12      printf("Please enter number 2: "); 13      scanf("%d", &amp;b); 14 15      printf("Please enter number 3: "); 16      scanf("%d", &amp;c); 17 18      printf("\nYour numbers in entry order:"); 19      printf("\n%d \n%d \n%d", a, b, c); 20 21      printf("\n\nYour numbers reversed:"); 22      printf("\n%d \n%d \n%d", c, b, a); 23 24      return 0; 25 }</pre>	<pre> Please enter number 1: 65 Please enter number 2: 93 Please enter number 3: 32  Your numbers in entry order: 65 93 32  Your numbers reversed: 32 93 65  Please enter number 1: 83 Please enter number 2: 43 Please enter number 3: 18  Your numbers in entry order: 83 43 18  Your numbers reversed: 18 43 83 </pre>	

## LAB 2: 2x03a - Cubing

7/03/2022 - Monday, Week 2

Program aim				
Input-Process-Output plan				
Input	Process	Output		
User inputs a whole number. Value saved to variable called <b>value</b>	Create variable <b>result</b> which is the result of the cubed number using the formula: <b>value * value * value</b>	Program outputs cubed number.		
Testing/errors/debugging				
10     int result = value^3;	← It seems like using the power symbol (^) does not work, as it may have a different function to powers.			
For example, changing the formula of <b>results</b> to <b>value^3</b> causes the program to output the incorrect answer for 57 (the answer is supposed to be 185,193) → I did not receive any output log errors however.	Please enter a whole number: 57 57 cubed is: 58			
Implementation				
Source Code (Input)	Output/s			
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      int value = 0; 6 7      printf("Please enter a whole number: "); 8      scanf("%d", &amp;value); 9 10     int result = value * value * value; 11 12     printf("%d cubed is: %d", value, result); 13 14     return 0; 15 }</pre>	<pre> Please enter a whole number: 45 45 cubed is: 91125  Please enter a whole number: 25 25 cubed is: 15625  Please enter a whole number: 5 5 cubed is: 125</pre>			
Lessons learned				
The math operation '^' does not work as it has a different purpose. When solving powers, using " <b>x * x * x</b> " instead. (x being the number I want to be powered).				

**LAB 2: 2x04a – Square Calculator**

7/03/2022 - Monday, Week 2

Program aim		
Ask user for the length of a square. Then, calculate perimeter (length x 4) and area of the square ( $\text{length}^2$ ).		
Input-Process-Output plan		
Input	Process	Output
User inputs number (length of a square). Save number to variable <b>length</b>	Calculate perimeter using formula <b>length * 4</b> and save result to variable <b>perimeter</b> . Calculate area using formula <b>length * length</b> and save result to variable <b>area</b> .	Program outputs the square's perimeter and the square's area.
Implementation		
Source Code (Input)	Output/s	
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      int length = 0; 6 7      printf("What is the length of one side? "); 8      scanf("%d", &amp;length); 9 10     int perimeter = length * 4; 11     int area = length * length; 12 13     printf("\nThe square's perimeter is: %d \n\n", perimeter); 14     printf("The square's area is: %d", area); 15 16     return 0; 17 }</pre>	<pre>What is the length of one side? 28 The square's perimeter is: 112 The square's area is: 784</pre> <pre>What is the length of one side? 47 The square's perimeter is: 188 The square's area is: 2209</pre> <pre>What is the length of one side? 79 The square's perimeter is: 316 The square's area is: 6241</pre>	

**LAB 2: 2x05a – Circle Calculator**

7/03/2022 - Monday, Week 2

**Program aim**

Ask user to enter radius of a circle (real number). Calculate the area ( $\pi \times \text{radius}^2$ ) and circumference ( $2 \times \pi \times \text{radius}$ ) of the circle using the radius- then output this to the user. Then calculate the area and circumference of a circle (and print to user) twice the radius the user originally inputted.

**Input-Process-Output plan**

Input	Process	Output
User inputs radius of a circle (real number). Input saved to float variable <b>radius</b>	Calculate area: <b>pi*radius*radius</b> and calculate circumference: <b>2*pi*radius</b> . Double the original radius and save it to variable <b>radius2</b> . Calculate area and circumference of <b>radius2</b> .	Program prints the original radius's area and perimeter to user, then the area and perimeter of double the radius.

**Implementation**

Source Code (Input)	Output/s
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      const float pi = 3.14159f; 6      float radius = 0.0f; 7 8      printf("Circle Calculator:\n*****\n"); 9 10     printf("Please enter the radius of a circle: "); 11     scanf("%f", &amp;radius); 12 13     printf("\nCalculating...\n\n"); 14 15     float area = pi * radius * radius; 16     float circumference = 2 * pi * radius; 17 18     printf("A circle with radius %f has: \n\n", radius); 19 20     printf("  - An area of: %f \n", area); 21     printf("  - A circumference of: %f \n\n", circumference); 22 23     float radius2 = radius * 2; 24     float area2 = pi * radius2 * radius2; 25     float circumference2 = 2 * pi * radius2; 26 27     printf("Also, did you know a circle with radius %f has: \n\n", radius2); 28 29     printf("  - An area of: %f\n", area2); 30     printf("  - A circumference of: %f \n", circumference2); 31 32     return 0; 33 }</pre>	<pre> Circle Calculator: ***** Please enter the radius of a circle: 79.528000 Calculating... A circle with radius 79.528000 has:   - An area of: 19869.623047   - A circumference of: 499.688751  Also, did you know a circle with radius 159.056000 has:   - An area of: 79478.492188   - A circumference of: 999.377502</pre>
	<pre> Circle Calculator: ***** Please enter the radius of a circle: 3.438000 Calculating... A circle with radius 3.438000 has:   - An area of: 37.133106   - A circumference of: 21.601574  Also, did you know a circle with radius 6.876000 has:   - An area of: 148.532425   - A circumference of: 43.203148</pre>

**Lessons learned**

- Remember to use 'const' to define constants to avoid "magic numbers" (repeating numbers)
  - Remember to set the initial value of float variables to '0.0f'

**LAB 2: 2x06a – Temperature Converter**

7/03/2022 - Monday, Week 2

**Program aim**

Convert user input: Fahrenheit (real number) into Celsius using formula

$$\text{Celsius} = (5 \div 9) \times (\text{Fahrenheit} - 32). \text{ Display the results.}$$

**Input-Process-Output plan****Input****Process****Output**

User inputs Fahrenheit as a real number. User input saved into float variable <b>fahrenheit</b>	Convert Fahrenheit input into Celsius using formula: <b>(fahrenheit - 32) * 5/9</b> Save result to variable <b>celsius</b>	Program prints the conversion of user input (Fahrenheit) to Celsius.
---	--	--

**Testing/errors/debugging**

In attempting to compute the formula exactly in code form, even though the order of operations was the same, it did not output the correct answer as expected.

For example, this line: 17 | `float celsius = (5 / 9) * (fahrenheit - 32);`

Outputted this:

```
62.419998 degrees Fahrenheit is the same as...
0.000000 degrees Celsius.
```

I played around with a different order of operations and the formula: `(fahrenheit-32) * 5/9` worked.

**Implementation****Source Code (Input)**

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     float fahrenheit = 0;
6
7     printf("-----+\n");
8     printf("| Temperature Converter |\n");
9     printf("-----+\n\n");
10
11    printf("Please enter a temperature in Fahrenheit: ");
12    scanf("%f", &fahrenheit);
13
14    printf("\nConverting...\n\n");
15
16    // Fahrenheit to Celsius formula: Celsius = (Fahrenheit - 32) x (5 ÷ 9)
17    float celsius = (fahrenheit - 32) * 5 / 9;
18
19    printf("%f degrees Fahrenheit is the same as...\n", fahrenheit);
20    printf("%f degrees Celsius.", celsius);
21
22
23 }
```

**Output/s**

```
+-----+
| Temperature Converter |
+-----+
Please enter a temperature in Fahrenheit: 42.219002
Converting...
42.219002 degrees Fahrenheit is the same as...
5.677223 degrees Celsius.

+-----+
| Temperature Converter |
+-----+
Please enter a temperature in Fahrenheit: 90.179001
Converting...
90.179001 degrees Fahrenheit is the same as...
32.321671 degrees Celsius.
```

**Lessons learned**

Watch out for order of operations (BEDMAS).

**LAB 2: 2x07a – Reals Sum and Average**

7/03/2022 - Monday, Week 2

Program aim		
Ask user for 6 float numbers, calculate, and output the sum and average of the 6 inputted numbers.		
Input-Process-Output plan		
Input	Process	Output
User inputs 6 float values. The input is saved to 6 float variables: num1, num2, num3, num4, num5, and num6.	Calculate sum of inputs: num1 + num2 + num3 + num4 + num5 + num6 and save to float variable sum. Then calculate average: sum/6 and save to float variable average	Program prints the sum and the average of the 6 float values inputted by the user.
Implementation		
Source Code (Input)	Output/s	
<pre> 3 int main(void) 4 { 5     float num1 = 0; 6     float num2 = 0; 7     float num3 = 0; 8     float num4 = 0; 9     float num5 = 0; 10    float num6 = 0; 11 12    printf("Please enter six real numbers: \n\n"); 13 14    printf("Number 1? "); 15    scanf("%f", &amp;num1); 16 17    printf("Number 2? "); 18    scanf("%f", &amp;num2); 19 20    printf("Number 3? "); 21    scanf("%f", &amp;num3); 22 23    printf("Number 4? "); 24    scanf("%f", &amp;num4); 25 26    printf("Number 5? "); 27    scanf("%f", &amp;num5); 28 29    printf("Number 6? "); 30    scanf("%f", &amp;num6); 31 32    float sum = num1 + num2 + num3 + num4 + num5 + num6; 33    float average = sum / 6; 34 35    printf("\nThe sum is: %.2f\n", sum); 36    printf("The average is: %.2f", average); 37 38    return 0; 39 }</pre>	<pre> Please enter six real numbers: Number 1? 16.165001 Number 2? 50.410000 Number 3? 83.079002 Number 4? 25.308001 Number 5? 74.132004 Number 6? 56.862000  The sum is: 305.96 The average is: 50.99</pre> <pre> Please enter six real numbers: Number 1? 59.179001 Number 2? 50.917999 Number 3? 51.969002 Number 4? 13.152000 Number 5? 62.855999 Number 6? 49.359001  The sum is: 287.43 The average is: 47.91</pre> <pre> Please enter six real numbers: Number 1? 46.530998 Number 2? 21.450001 Number 3? 100.494003 Number 4? 100.710999 Number 5? 97.998001 Number 6? 56.743000  The sum is: 423.93 The average is: 70.65</pre>	

**LAB 2: 2x08a – Seconds Converter**

7/03/2022 - Monday, Week 2

Program aim				
Convert user input of seconds (integer) and print it's equivalent time in hours, minutes, and seconds.				
Input-Process-Output plan				
Input	Process	Output		
User inputs seconds (integer). The user input is saved to integer variable called <b>seconds</b>	Calculate hours using formula <b>seconds/3600</b> and save result to integer variable <b>hours</b> . Then calculate minutes using formula <b>seconds/60%60</b> and save result to integer variable <b>minutes</b> . Finally, calculate remainder seconds using formula <b>seconds % 60</b> and save result to integer variable <b>seconds2</b> .	Print the results into the format: x hour(s), x minute(s), x second(s) (x being the result)		
Testing/errors/debugging				
I had a lot of trouble getting the computation of minutes to output the correct answer, due to my confusion with the use of the modulus (%) operator.				
Implementation				
Source Code (Input)	Output/s			
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      int seconds = 0; 6 7      printf("Number of seconds? "); 8      scanf("%d", &amp;seconds); 9 10     int hours = seconds / 3600; 11     int minutes = (seconds / 60) % 60; 12     int seconds2 = seconds % 60; 13 14     printf("\n%d hour(s), %d minute(s), %d second(s).", hours, minutes, seconds2); 15 16     return 0; 17 }</pre>	<p>Number of seconds? 28872 8 hour(s), 1 minute(s), 12 second(s).</p> <p>Number of seconds? 4358 1 hour(s), 12 minute(s), 38 second(s).</p> <p>Number of seconds? 527 0 hour(s), 8 minute(s), 47 second(s).</p>			
Lessons learned				
REMEMBER: the % symbol (modulus) is used as a math operator for finding the remainder of division.				

**LAB 2: 2x09a – Enhance the User Interface**

7/03/2022 - Monday, Week 2

**Program aim**

From the given source code, add printf statements to clarify the purpose of the program to the user

**Purpose of program**

From reading the source code, I have concluded that the user must input 3 decimal values. The program must then find the sum and average of these numbers and output only the average result to the user.

**Implementation (comparison to given source code)**

Given Source Code	My Source Code
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      float a = 0.0f; 6      float b = 0.0f; 7      float c = 0.0f; 8 9      printf("&gt; "); 10     scanf("%f", &amp;a); 11     printf("&gt; "); 12     scanf("%f", &amp;b); 13     printf("&gt; "); 14     scanf("%f", &amp;c); 15 16     float d = a + b + c; 17     float e = d / 3.0f; 18 19     printf("%f\n", e); 20 21     return 0; 22 }</pre>	<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      // input 3 float variables. Find the sum, and find the average. 6 7      float a = 0.0f; 8      float b = 0.0f; 9      float c = 0.0f; 10 11     printf("-----+ \n"); 12     printf(" \ Average of 3 Numbers  \n"); 13     printf("-----+ \n\n"); 14 15     printf("Number 1: "); 16     scanf("%f", &amp;a); 17     printf("Number 2: "); 18     scanf("%f", &amp;b); 19     printf("Number 3: "); 20     scanf("%f", &amp;c); 21 22     float d = a + b + c; 23     float e = d / 3.0f; 24 25     printf("\nThe average is: %f\n", e); 26 27     return 0; 28 }</pre>

**Output comparison**

Given output	My output
No context is given, the user wouldn't know what to input and why!	<pre> +-----+   Average of 3 Numbers   +-----+ Number 1: 4.45 Number 2: 2.222 Number 3: 3.7 The average is: 3.457333 </pre>

**LAB 2: 2x10a – P1 Result Calculator**

7/03/2022 - Monday, Week 2

**Program aim**

Program must ask Semester 1, 2019 COMP500/ENSE501 student to input assessment results and then calculate the contribution per assessment item and overall result.

**Input-Process-Output plan**

Input	Process	Output
User must input their reporting journal, practical test 1, 2 and 3, and final practical exam results as real numbers. These inputs will be saved to float variables: <code>journal, test1, test2, test3, and exam.</code>	Calculate percentage contribution to the following scores: journal result is <code>0.1*journal</code> , test 1 result is <code>0.1*test1</code> , test 2 result is <code>0.1*test2</code> , test3 is <code>0.15*test3</code> , exam result is <code>0.5*exam</code> , and overall result is the sum of percentage contribution of each result. Save results to variables: <code>journal_result, test1_result, test2_result, test3_result, and overall_result.</code>	The program prints the results of the percentage contribution of each score to the user.  The program then prints to the user additional information about how to pass the paper.

**Testing/errors/debugging**

```
ects\comp500\lab 02 vss\2x10a - p1 result calculator\p1_intermediatebuildstep.c(27): error C2220: warning treated as error - no 'object' file generated
ects\comp500\lab 02 vss\2x10a - p1 result calculator\p1_intermediatebuildstep.c(27): warning C4244: 'initializing': conversion from 'double' to 'float', possible loss of data
ects\comp500\lab 02 vss\2x10a - p1 result calculator\p1_intermediatebuildstep.c(28): warning C4244: 'initializing': conversion from 'double' to 'float', possible loss of data
ects\comp500\lab 02 vss\2x10a - p1 result calculator\p1_intermediatebuildstep.c(29): warning C4244: 'initializing': conversion from 'double' to 'float', possible loss of data
ects\comp500\lab 02 vss\2x10a - p1 result calculator\p1_intermediatebuildstep.c(30): warning C4244: 'initializing': conversion from 'double' to 'float', possible loss of data
ects\comp500\lab 02 vss\2x10a - p1 result calculator\p1_intermediatebuildstep.c(31): warning C4244: 'initializing': conversion from 'double' to 'float', possible loss of data
P1 Result Calculator.vcxproj" -- FAILED.
```

The error output log above (lines 27-32) occurs when I define the result calculations as 'floats'. This is due to truncation (loss of data). Changing them to double type fixes this error.

**Implementation**

Source Code (lines 5-46)	Output/s (lines 14-40)
<pre> 5   float journal = 0.0f; 6   float test1 = 0.0f; 7   float test2 = 0.0f; 8   float test3 = 0.0f; 9   float exam = 0.0f; 10 11  printf("COMP500/ENSE501 Result Calculator: \n"); 12  printf("-----\n"); 13 14  printf("Enter the score (out of 100) for the Reporting Journal: "); 15  scanf("%f", &amp;journal); 16  printf("Enter the score (out of 100) for the Practical Test 1: "); 17  scanf("%f", &amp;test1); 18  printf("Enter the score (out of 100) for the Practical Test 2: "); 19  scanf("%f", &amp;test2); 20  printf("Enter the score (out of 100) for the Practical Test 3: "); 21  scanf("%f", &amp;test3); 22  printf("Enter the score (out of 100) for the Final Practical Exam: "); 23  scanf("%f", &amp;exam); 24 25  printf("\nCalculating...\n"); 26 27  double journal_result = 0.15 * journal; 28  double test1_result = 0.10 * test1; 29  double test2_result = 0.10 * test2; 30  double test3_result = 0.15 * test3; 31  double exam_result = 0.50 * exam; 32  double overall_result = journal_result + test1_result + test2_result + test3_result + exam_result; 33 34  printf("Reporting Journal (worth 15%) contributes: %f \n", journal_result); 35  printf("Practical Test 1 (worth 10%) contributes: %f \n", test1_result); 36  printf("Practical Test 2 (worth 10%) contributes: %f \n", test2_result); 37  printf("Practical Test 3 (worth 15%) contributes: %f \n", test3_result); 38  printf("Final Practical Exam (worth 50%) contributes: %f \n\n", exam_result); 39 40  printf("Overall result total: %f%%\n\n", overall_result); 41 42  printf("Remember, to pass the paper, a student must achieve: \n"); 43  printf(" - At least 80% attendance and participation in the \n"); 44  printf(" individual's scheduled lab tutorial stream, AND \n"); 45  printf(" - A minimum mark of 40% for the Final Practical Exam, AND \n"); 46  printf(" - A minimum C- (50%) overall grade. \n"); </pre>	<p>(Assuming I have already printed the title and additional information correctly)</p> <pre> Enter the score (out of 100) for the Reporting Journal: 46 Enter the score (out of 100) for the Practical Test 1: 13 Enter the score (out of 100) for the Practical Test 2: 35 Enter the score (out of 100) for the Practical Test 3: 75 Enter the score (out of 100) for the Final Practical Exam: 51  Calculating...  Reporting Journal (worth 15%) contributes: 6.900000 Practical Test 1 (worth 10%) contributes: 1.300000 Practical Test 2 (worth 10%) contributes: 3.500000 Practical Test 3 (worth 15%) contributes: 11.250000 Final Practical Exam (worth 50%) contributes: 25.500000  Overall result total: 48.450000 </pre> <pre> Enter the score (out of 100) for the Reporting Journal: 29 Enter the score (out of 100) for the Practical Test 1: 20 Enter the score (out of 100) for the Practical Test 2: 59 Enter the score (out of 100) for the Practical Test 3: 36 Enter the score (out of 100) for the Final Practical Exam: 10  Calculating...  Reporting Journal (worth 15%) contributes: 4.350000 Practical Test 1 (worth 10%) contributes: 2.000000 Practical Test 2 (worth 10%) contributes: 5.900000 Practical Test 3 (worth 15%) contributes: 5.400000 Final Practical Exam (worth 50%) contributes: 5.000000  Overall result total: 22.650000% </pre>

**LAB 2: 2x10a – P1 Result Calculator Continued**

7/03/2022 - Monday, Week 2

Trace table trial using the first output values from the implementation table

```
Enter the score (out of 100) for the Reporting Journal: 46
Enter the score (out of 100) for the Practical Test 1: 13
Enter the score (out of 100) for the Practical Test 2: 35
Enter the score (out of 100) for the Practical Test 3: 75
Enter the score (out of 100) for the Final Practical Exam: 51
```

From the following source code (lines 5-23):

```
5   float journal = 0.0f;
6   float test1 = 0.0f;
7   float test2 = 0.0f;
8   float test3 = 0.0f;
9   float exam = 0.0f;
10
11  printf("COMP500/ENSE501 Result Calculator: \n");
12  printf("-----\n");
13
14  printf("Enter the score (out of 100) for the Reporting Journal: ");
15  scanf("%f", &journal);
16  printf("Enter the score (out of 100) for the Practical Test 1: ");
17  scanf("%f", &test1);
18  printf("Enter the score (out of 100) for the Practical Test 2: ");
19  scanf("%f", &test2);
20  printf("Enter the score (out of 100) for the Practical Test 3: ");
21  scanf("%f", &test3);
22  printf("Enter the score (out of 100) for the Final Practical Exam: ");
23  scanf("%f", &exam);
```

**Trace Table 1 (lines 5-23)**

Line Number	Output				
	Variable: journal	Variable: test1	Variable: test2	Variable: test3	Variable: exam
5	?				
6	?	?			
7	?	?	?		
8	?	?	?	?	
9	?	?	?	?	?
10	?	?	?	?	?
11	?	?	?	?	?
12	?	?	?	?	?
13	?	?	?	?	?
14	?	?	?	?	?
15	<b>46</b>	?	?	?	?
16	<b>46</b>	?	?	?	?
17	<b>46</b>	<b>13</b>	?	?	?
18	<b>46</b>	<b>13</b>	?	?	?
19	<b>46</b>	<b>13</b>	<b>35</b>	?	?
20	<b>46</b>	<b>13</b>	<b>35</b>	?	?
21	<b>46</b>	<b>13</b>	<b>35</b>	<b>75</b>	?
22	<b>46</b>	<b>13</b>	<b>35</b>	<b>75</b>	?
23	<b>46</b>	<b>13</b>	<b>35</b>	<b>75</b>	<b>51</b>

## LAB 2: 2x10a – P1 Result Calculator Continued

7/03/2022 - Monday, Week 2

Trace table trial using the first output values from the implementation table

```
Reporting Journal (worth 15%) contributes: 6.900000
Practical Test 1 (worth 10%) contributes: 1.300000
Practical Test 2 (worth 10%) contributes: 3.500000
Practical Test 3 (worth 15%) contributes: 11.250000
Final Practical Exam (worth 50%) contributes: 25.500000

Overall result total: 48.450000
```

From the following source code (lines 27-40):

```
27     double journal_result = 0.15 * journal;
28     double test1_result = 0.10 * test1;
29     double test2_result = 0.10 * test2;
30     double test3_result = 0.15 * test3;
31     double exam_result = 0.50 * exam;
32     double overall_result = journal_result + test1_result + test2_result + test3_result + exam_result;
```

Trace Table 2 (lines 27-40)

Line Number	Output					
	Variable: journal_result	Variable: test1_result	Variable: test2_result	Variable: test3_result	Variable: exam_result	Variable: overall_result
27	6.900000					
28	6.900000	1.300000				
29	6.900000	1.300000	3.500000			
30	6.900000	1.300000	3.500000	11.250000		
31	6.900000	1.300000	3.500000	11.250000	25.500000	
32	6.900000	1.300000	3.500000	11.250000	25.500000	48.450000

**Week 2 lecture 005 Notes- Characters, ASCII, scanf with char, Formatted Text Input**

Notes Retrieved from Steffan Hooper, W02 Lec 005, 2022

**09/03/2022 - Wednesday**

Char (character) data type can store whole numbers and single characters (a,b,c...). Usually they hold characters. Characters are outputted with the placeholder: %c

Signed variables hold positive, negative, or zero values. Unsigned variables can only hold store positive values or zero. Simply declaring a variable as char will automatically create a signed char.

ASCII – American Standard Code for Information Interchange. ASCII encodes 128 characters into 7-bit integers.

ASCII TABLE			
Decimal Value	Hexadecimal Value	ASCII character	C char Literal
0	0x0	NULL	'\0'
...			
7	0x7	Alarm	'\a'
8	0x8	Backspace	'\b'
9	0x9	Horizontal Tab	'\t'
10	0xA	Line Feed	'\n'
11	0xB	Vertical Tab	'\v'
12	0xC	Form Feed	'\f'
13	0xD	Carriage Return	'\r'
...			
32	0x20	Space	' '

**Characters of decimal value 32 and above are printable. This includes all symbols, numbers, and letters.**

Using character literal '\0' evaluates the value to 0.

Scarf means "scan formatted". Buffered means data from the keyboard is stored in a temporary place in the memory before the program gets access to it.

1. Program only gets access to input buffer when the user presses enter while the program is running scanf.
2. Scarf converts the buffer into the format requested by the program.

Scarf and whitespace input: remember to put a space before "%c" so the scan does not hold the \n when the user presses enter (for example, user inputs "a \n"). The leading whitespace will cause scarf to consume any whitespace left in the input buffer.

Remember, do not scan for \n: for example, don't do this: `scanf("%d\n", &user_input)`

Reading multiple inputs in one `scanf` line example:

```
scanf("%d/%d/%d", &day, &month, &year);
```

Program will scan an int then '/' then int then '/' then int from the keyboard input buffer...

If I get a security warning (C4996) because of `scanf`, add `#define _CRT_SECURE_NO_WARNINGS` to the top of program.

## LAB 2: 2x01b – Next Year

9/03/2022 - Wednesday, Week 2

Program aim				
Input-Process-Output plan				
Input	Process	Output		
User inputs age as integer (whole number). Input saved to integer variable	Increment age variable using <code>age++</code>	Print the age the user will be next year.		
Testing/errors/debugging				
I initially used <code>age = age + 1;</code> to add 1 to the age variable, but then I remembered that incrementation exists, so I changed the line of code to <code>age++;</code>				
Implementation				
Source Code (Input)	Output/s			
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      int age = 0; 6 7      printf("What is your age? "); 8      scanf("%d", &amp;age); 9 10     age++; 11 12     printf("\nNext year you will be %d.", age); 13 14     return 0; 15 }</pre>	<p>What is your age? 96</p> <p>Next year you will be 97.</p> <p>What is your age? 6</p> <p>Next year you will be 7.</p> <p>What is your age? 29</p> <p>Next year you will be 30.</p>			
Lessons learned				
REMEMBER: to increment a variable's value by 1, use <code>variable++</code>				

**LAB 2: 2x01c – Exam Time Remaining**

9/03/2022 - Wednesday, Week 2

Program aim		
Ask user how many minutes have elapsed since the start of exam. Calculate and output the time remaining in minutes.		
Input-Process-Output plan		
Input	Process	Output
User inputs minutes elapsed since start of exam (as integer). Input is saved to variable called <code>minutes_start</code>	Remaining time is calculated using formula <code>150 - minutes_start</code> . Result is saved to variable <code>minutes_left</code>	Program outputs the calculated time remaining of the exam.
Testing/errors/debugging		
Using compound assignment for subtraction does not work as using the following line of code:		
<pre>10   150 -= minutes_start;</pre>		
Outputs the following error:		
<pre>exam time remaining\p1_intermediatebuildstep.c(10): error C2106: '=': left operand must be l-value</pre>		
So to fix this, I just created a new variable called <code>minutes_left</code> to hold the result of <code>150 - minutes_start</code>		
Implementation		
Source Code (Input)		Output/s
<pre>1 #include &lt;stdio.h&gt; 2 3 int main(void) 4 { 5     int minutes_start = 0; 6 7     printf("How many minutes since the start of the exam? "); 8     scanf("%d", &amp;minutes_start); 9 10    int minutes_left = 150 - minutes_start; 11 12    printf("\nYou have %d minutes left!!!", minutes_left); 13 14    return 0; 15 }</pre>		<pre>How many minutes since the start of the exam? 59 You have 91 minutes left!!!  How many minutes since the start of the exam? 86 You have 64 minutes left!!!  How many minutes since the start of the exam? 137 You have 13 minutes left!!!</pre>
Lessons learned		
Compound assignment does not necessarily need to be used all the time.		

**LAB 2: 2x01d – Degree Points Left**

9/03/2022 - Wednesday, Week 2

**Program aim**

Ask user how many points they have achieved in their bachelor's degree so far. A bachelor's degree is 360 points in total. Then, output how many points are left.

**Input-Process-Output plan**

Input	Process	Output
User inputs how many points they have (integer). Input saved to variable <code>points</code>	Calculate points left using formula <code>360 - points</code> , save result to variable called <code>points_left</code> .	Program outputs how many points are left to the user.

**Implementation**

Source Code (Input)	Output/s
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      int points = 0; 6 7      printf("How many points have you achieved thus far? "); 8      scanf("%d", &amp;points); 9 10     int points_left = 360 - points; 11 12     printf("\nYou have %d points left!!!", points_left); 13 14     return 0; 15 }</pre>	<pre>How many points have you achieved thus far? 253 You have 107 points left!!!  How many points have you achieved thus far? 167 You have 193 points left!!!  How many points have you achieved thus far? 9 You have 351 points left!!!</pre>

## LAB 2: 2x02b – Simple Multiplier

9/03/2022 – Wednesday, Week 2

Program aim		
Input-Process-Output plan		
Input	Process	Output
User inputs two whole numbers that are saved to integer variables <b>a</b> and <b>b</b>	Product of numbers saved to int variable <b>c</b> $c = a * b$	Output product of the 2 numbers
Implementation		
Source Code (Input)	Output/s	
<pre> 1  #include &lt;stdio.h&gt; 2 3  // Simple Multiplier 4 5  int main(void) 6  { 7      int a = 0; 8      int b = 0; 9 10     printf("Please enter a whole number: "); 11     scanf("%d", &amp;a); 12 13     printf("Please enter another whole number: "); 14     scanf("%d", &amp;b); 15 16     int c = a * b; 17 18     printf("%d times %d is: %d", a, b, c); 19 20     return 0; 21 }</pre>	<pre> Please enter a whole number: 89 Please enter another whole number: 21 89 times 21 is: 1869</pre> <pre> Please enter a whole number: 100 Please enter another whole number: 21 100 times 21 is: 2100</pre> <pre> Please enter a whole number: 12 Please enter another whole number: 1 12 times 1 is: 12</pre>	

**LAB 2: 2x03b – Megabytes Used**

9/03/2022 – Wednesday, Week 2

**Program aim**

Ask user how many megabytes of data they have used on their mobile phone this month and output how many megabytes are left. Their monthly mobile quota is 1500 megabytes.

**Input-Process-Output plan**

Input	Process	Output
User inputs <code>mb_used</code> as integer.	Create Integer variable <code>mb_left</code> from formula $1500 - mb\_used$	Print the megabytes left for the month to the user.

**Implementation**

Source Code (Input)	Output/s
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      // Calculate a person's monthly mobile data quota of 1500 megabytes minus how much they used 6 7      { 8          int mb_used = 0; 9 10         printf("How many megabytes have you used thus far? "); 11         scanf("%d", &amp;mb_used); 12 13         int mb_left = 1500 - mb_used; 14 15         printf("\nYou have %d megabytes left!!!", mb_left); 16 17         return 0; 18     } </pre>	<pre>How many megabytes have you used thus far? 189 You have 1311 megabytes left!!!</pre>
	<pre>How many megabytes have you used thus far? 1168 You have 332 megabytes left!!!</pre>
	<pre>How many megabytes have you used thus far? 1500 You have 0 megabytes left!!!</pre>

**LAB 2: 2x04b – Rectangle Calculator**

9/03/2022 – Wednesday, Week 2

Program aim		
Ask user to input height and width of rectangle, then calculate perimeter and area of the rectangle.		
Input-Process-Output plan		
Input	Process	Output
User inputs <b>width</b> and <b>height</b> of a rectangle (integer).	Integer variable <b>perimeter</b> is the result of formula $2 * (\text{width} + \text{height})$ , and integer variable <b>area</b> is the result of formula $\text{width} * \text{height}$	Print the rectangle's calculated perimeter and area to the user
Implementation		
Source Code (Input)	Output/s	
<pre> 1 #include &lt;stdio.h&gt; 2 3 int main(void) 4 { 5     int width = 0; 6     int height = 0; 7 8     printf("Rectangle Calculator: \n"); 9     printf("----- \n\n"); 10 11    printf("What is the width of the rectangle? "); 12    scanf("%d", &amp;width); 13    printf("What is the height of the rectangle? "); 14    scanf("%d", &amp;height); 15 16    int perimeter = 2 * (width + height); 17    int area = width * height; 18 19    printf("\nThe rectangle's perimeter is: %d \n", perimeter); 20    printf("The rectangle's area is: %d \n", area); 21 22 23 } 24 </pre>	<p><b>Rectangle Calculator:</b> -----</p> <p>What is the width of the rectangle? 51 What is the height of the rectangle? 29</p> <p>The rectangle's perimeter is: 160 The rectangle's area is: 1479</p> <p><b>Rectangle Calculator:</b> -----</p> <p>What is the width of the rectangle? 5 What is the height of the rectangle? 61</p> <p>The rectangle's perimeter is: 132 The rectangle's area is: 305</p> <p><b>Rectangle Calculator:</b> -----</p> <p>What is the width of the rectangle? 1 What is the height of the rectangle? 72</p> <p>The rectangle's perimeter is: 146 The rectangle's area is: 72</p>	

**LAB 2: 2x05b – Trapezium Calculator**

9/03/2022 – Wednesday, Week 2

Program aim		
Create a program that calculates a trapezium's area from two bases and height		
Input-Process-Output plan		
Input	Process	Output
User inputs <b>base1</b> , <b>base2</b> , and <b>height</b> as float values.	Use formula $0.5f * (\text{base1} + \text{base2}) * \text{height}$ to calculate the trapezium area, save result to float variable <b>area</b> .	Print the trapezium area to the user.
Testing/errors/debugging		
The truncation warning below occurred because I did not include '.0f' when defining the variables and '0.5' in the formula		
warning C4244: 'initializing': conversion from 'double' to 'float', possible loss of data		
I tried resolving this by turning the ' <b>float</b> 's into ' <b>double</b> 's, but this resulted in the area equating to 0 due to truncation. I fixed this error by reverting the values back to ' <b>float</b> ' and adding ' <b>.0f</b> ' to each variable and the <b>0.5</b> literal		
Implementation		
Source Code (Input)	Output/s	
<pre>2x05b - Trapezium Calculator 1 #include &lt;stdio.h&gt; 2 3 int main(void) 4 { 5 6     // Trapezium area calculator 7 8     float base1 = 0.0f; 9     float base2 = 0.0f; 10    float height = 0.0f; 11 12    printf("What is the length of the trapezium's first base? "); 13    scanf("%f", &amp;base1); 14 15    printf("What is the length of the trapezium's second base? "); 16    scanf("%f", &amp;base2); 17 18    printf("What is the height of the trapezium? "); 19    scanf("%f", &amp;height); 20 21    float area = 0.5f * (base1 + base2) * height; 22 23    printf("\n\nThe trapezium's area is: %.2f", area); 24 25    return 0; 26 }</pre>	<p>What is the length of the trapezium's first base? 35.820000      What is the length of the trapezium's second base? 90.260002      What is the height of the trapezium? 94.589996      The trapezium's area is: 5962.95</p> <p>What is the length of the trapezium's first base? 96.279999      What is the length of the trapezium's second base? 100.589996      What is the height of the trapezium? 36.470001      The trapezium's area is: 3589.92</p> <p>What is the length of the trapezium's first base? 84.860001      What is the length of the trapezium's second base? 45.990002      What is the height of the trapezium? 14.540000      The trapezium's area is: 951.28</p>	

**Lessons learned**

When defining float values, especially when they are to be used in a math formula, always include a decimal place followed by 'f' (.0f), including the literal values to be used in the formula.

## LAB 2: 2x01e – Attendance

10/03/2022 – Thursday (lab) , Week 2

### Program aim

Ask user the week number, whether they were late, and their attendance percentage. Then output the information in the format: “Week: X, Late: Y, Attendance percentage: Z%”.

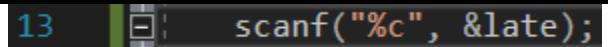
### Input-Process-Output plan

User Input (variables)	Process	Output
<b>week</b> number as integer	Save user inputs to variables <b>week</b> , <b>late</b> , <b>attendance</b> .	Results printed to user in format: “Week: X, Late: Y, Attendance percentage: Z%”
<b>late</b> (y/n) as character		
<b>attendance</b> (%) as float		

### Testing/errors/debugging

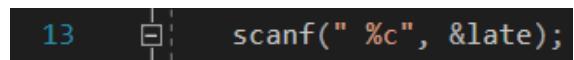
If I forget to put a space before "%c" in line 13 like this →

```
Week? 2
Late? Attendance percentage? 89.33
Week: 2, Late:
, Attendance percentage: 89.33%
```



← I will get this incorrect output because the character variable will take the enter button as it's input instead and skip the user's input.

To fix this error, I simply need to put a space behind "%c" like this →



### Implementation

Source Code (Input)	Output/s
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      int week = 0; 6      char late = 0; 7      float attendance = 0.0f; 8 9      printf("Week? "); 10     scanf("%d", &amp;week); 11 12     printf("Late? "); 13     scanf(" %c", &amp;late); // IMPORTANT: ALWAYS PUT SPACE BEFORE CHAR SCANF 14     // (since the enter from the previous input is taken as input) 15 16     printf("Attendance percentage? "); 17     scanf("%f", &amp;attendance); 18 19     printf("\nWeek: %d, Late: %c, Attendance percentage: %.2f%%", week, late, attendance); 20 21     return 0; 22 }</pre>	<pre>Week? 1 Late? n Attendance percentage? 0.000  Week: 1, Late: n, Attendance percentage: 0.00%</pre> <pre>Week? 1 Late? n Attendance percentage? 61.16  Week: 1, Late: n, Attendance percentage: 61.16%</pre> <pre>Week? 12 Late? n Attendance percentage? 93.6  Week: 12, Late: n, Attendance percentage: 93.60%</pre>

### Lessons learned

Always put a space before character input "%c" to prevent the variable from taking the enter button as its input!

## LAB 2: 2x02c – Math Operation Test

10/03/2022 – Thursday (lab) , Week 2

### Program aim

Ask the user for 2 whole numbers. The program must then perform addition, subtraction, multiplication, integer division, float division, and remainder of integer division (modulus) on the two numbers.

### Input-Process-Output plan

User Input (variables)	Process / Steps	Output
<b>A</b> and <b>B</b> (float)	<b>A + B</b>	Sum of <b>A + B</b>
	<b>A - B</b>	Difference of <b>A - B</b>
	<b>A * B</b>	Product of <b>A * B</b>
	<b>INT A / B</b>	Quotient of <b>A / B</b> no decimal places
	<b>FLOAT A / B</b>	Quotient of <b>A / B</b> full decimal places
	<b>A % B</b>	Remainder of <b>A % B</b>

### Implementation

Source code before feedback	Feedback (lab zoom meeting)	Source code after feedback
<pre>2x02c - Math Operation Test (Global Scope) 1 #include &lt;stdio.h&gt; 2 3 int main(void) 4 { 5     int a = 0; 6     int b = 0; 7 8     printf("Math Operation Test: \n"); 9     printf("----- \n\n"); 10 11    printf("Please enter the first whole number: "); 12    scanf("%d", &amp;a); 13 14    printf("Please enter the second whole number: "); 15    scanf("%d", &amp;b); 16 17    printf("\nCalculating... \n\n"); 18 19    int addition = a + b; 20    int subtraction = a - b; 21    int multiplication = a * b; 22    int int_division = a / b; 23    float float_division = (float)a / (float)b; 24    int remainder = a % b; 25 26    printf("%d + %d is %d \n\n", a, b, addition); 27    printf("%d - %d is %d \n\n", a, b, subtraction); 28    printf("%d * %d is %d \n\n", a, b, multiplication); 29 30    printf("With integer division: \n\n"); 31    printf("%d / %d is %d \n\n", a, b, int_division); 32 33    printf("With floating point division: \n\n"); 34    printf("%d / %d is %f \n\n", a, b, float_division); 35 36    printf("The remainder of %d divided by %d is %d", a, b, remainder); 37 38    return 0; 39 }</pre>	<p>Me to Everyone</p> <p>In 2x02c, Int variables are used in addition and subtraction, but the float division needs them to be a float. How do you convert them?</p> <p>FN</p> <p>Me to Everyone</p> <p>idk.PNG 5.26 KB</p> <p>BG</p> <p>Bernie Garnell to Everyone</p> <p>It might be better to create two float variables for the division or just use all floats.</p> <p>TR</p> <p>Tarun Ramachandran to Everyone</p> <p>you could use floats instead of integers for the numbers themselves, and use the printf formatting</p> <p>XH</p> <p>Xinyu Hu to Everyone</p> <p>or you can do float float_number = int_number * 1.0f;</p> <p>FN</p> <p>Me to Everyone</p> <p>I'll try these, thank you!</p>	<pre>2x02c - Math Operation Test (Global Scope) 1 #include &lt;stdio.h&gt; 2 3 int main(void) 4 { 5     float a = 0.0f; 6     float b = 0.0f; 7 8     printf("Math Operation Test: \n"); 9     printf("----- \n\n"); 10 11    printf("Please enter the first whole number: "); 12    scanf("%f", &amp;a); 13 14    printf("Please enter the second whole number: "); 15    scanf("%f", &amp;b); 16 17    printf("\nCalculating... \n\n"); 18 19    float addition = a + b; 20    float subtraction = a - b; 21    float multiplication = a * b; 22    int int_division = a / b; 23 24    float float_division = a / b; 25    int remainder = (int)a % (int)b; 26 27    printf("%.f + %.f is %.f \n\n", a, b, addition); 28    printf("%.f - %.f is %.f \n\n", a, b, subtraction); 29    printf("%.f * %.f is %.f \n\n", a, b, multiplication); 30 31    printf("With integer division: \n\n"); 32    printf("%.f / %.f is %d \n\n", a, b, int_division); 33 34    printf("With floating point division: \n\n"); 35    printf("%.f / %.f is %f \n\n", a, b, float_division); 36 37    printf("The remainder of %.f divided by %.f is %d", a, b, remainder); 38 39    return 0; 40 }</pre>

**LAB 2: 2x02c – Math Operation Test (Continued)**

10/03/2022 – Thursday (lab) , Week 2

Output Comparison	
Output before feedback	Output after feedback
<pre>Math Operation Test: ----- Please enter the first whole number: 50 Please enter the second whole number: 26 Calculating... 50 + 26 is 76 50 - 26 is 24 50 * 26 is 1300 With integer division: 50 / 26 is 1 With floating point division: 50 / 26 is 1073741824 The remainder of 50 divided by 26 is 24</pre>	<pre>Math Operation Test: ----- Please enter the first whole number: 6 Please enter the second whole number: 74 Calculating... 6 + 74 is 80 6 - 74 is -68 6 * 74 is 444 With integer division: 6 / 74 is 0 With floating point division: 6 / 74 is 0.081681 The remainder of 6 divided by 74 is 6</pre>
Lessons learned	
<ul style="list-style-type: none"> <li>- Even if the initial value is entered as a whole (integer) number, if float division is to be used later on, it is much easier to set the input variables as floats instead of integers.</li> <li>- REMEMBER to shorten the decimal places when outputting real numbers, use ex: .2f to simplify float to 2 decimal places or simply .f to remove decimal places.</li> </ul>	

## LAB 2: 2x02d – Simple Addition

10/03/2022 – Thursday (lab) , Week 2

Program aim				
Input-Process-Output plan				
User Input (variables)	Process	Output		
a as integer b as integer c as integer	<b>a + b + c</b>  Results are saved to Integer variable: <b>ans</b>	Print the sum of the three inputted values to the user		
Implementation				
Source Code (Input)	Output/s			
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      int a; 6      int b; 7      int c; 8 9      printf("Please enter a whole number: "); 10     scanf("%d", &amp;a); 11 12     printf("Please enter another whole number: "); 13     scanf("%d", &amp;b); 14 15     printf("Please enter another whole number: "); 16     scanf("%d", &amp;c); 17 18     int ans = a + b + c; 19 20     printf("\n%d plus %d plus %d is %d", a, b, c, ans); 21 22     return 0; 23 }</pre>	<pre> Please enter a whole number: 2 Please enter another whole number: 65 Please enter another whole number: 83   2 plus 65 plus 83 is 150  Please enter a whole number: 12 Please enter another whole number: 14 Please enter another whole number: 100   12 plus 14 plus 100 is 126  Please enter a whole number: 11 Please enter another whole number: 96 Please enter another whole number: 2   11 plus 96 plus 2 is 109 </pre>			
Lessons learned				
NOTE FOR NEXT TIME: abbreviated/unclear variable names are not recommended				

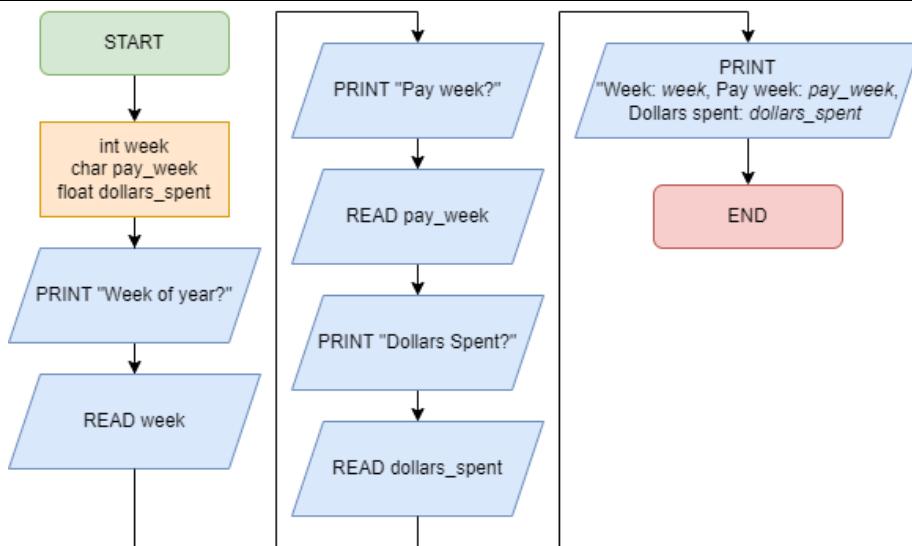
## LAB 2: 2x02e – Pay Week

10/03/2022 – Thursday (lab) , Week 2

### Program aim

Ask user for week number, whether they were paid that week (y/n), and how many dollars they spent. The program must then output the information in the style:  
 “Week: X, Pay week: Y, Dollars spent: Z”

### Flowchart Design (Created with draw.io)



### Implementation

#### Source Code (Input)

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     int week;
6     char pay_week;
7     float dollars_spent = 0.0f;
8
9     printf("Week of year? ");
10    scanf("%d", &week);
11
12    printf("Pay week? ");
13    scanf(" %c", &pay_week);
14
15    printf("Dollars spent? ");
16    scanf("%f", &dollars_spent);
17
18    printf("\nWeek: %d, Pay week: %c, Dollars spent: $%.2f", week, pay_week, dollars_spent);
19
20    return 0;
21 }
  
```

#### Output/s

Week of year? 1 Pay week? n Dollars spent? 0.00  Week: 1, Pay week: n, Dollars spent: \$0.00
Week of year? 1 Pay week? y Dollars spent? 200.00  Week: 1, Pay week: y, Dollars spent: \$200.00
Week of year? 29 Pay week? y Dollars spent? 134.63  Week: 29, Pay week: y, Dollars spent: \$134.63

### Lessons learned

I learned how to design a program using a flowchart for the first time! It's a fun way to design program algorithms and it helps see the bigger picture.

**LAB 2: 2x03d – Simple Multiplication Table**

10/03/2022 – Thursday (lab) , Week 2

Program aim		
Ask user for a whole number. Calculate and output a times table from the user's input from 1-12.		
User Input (variables)	Process (steps)	Output
num as integer	<pre> times_1 = num * 1 times_2 = num * 2 times_3 = num * 3 times_4 = num * 4 times_5 = num * 5 times_6 = num * 6 times_7 = num * 7 times_8 = num * 8 times_9 = num * 9 times_10 = num * 10 times_11 = num * 11 times_12 = num * 12 </pre>	The times table of the inputted number up to times 12 is printed to the user.
Implementation		
Source Code (lines 5-38)	Output/s	
<pre> 5   int num; 6 7   printf("Please enter a whole number: "); 8   scanf("%d", &amp;num); 9 10 // calculating multiplication table 11 12   int times_1 = num * 1; 13   int times_2 = num * 2; 14   int times_3 = num * 3; 15   int times_4 = num * 4; 16   int times_5 = num * 5; 17   int times_6 = num * 6; 18   int times_7 = num * 7; 19   int times_8 = num * 8; 20   int times_9 = num * 9; 21   int times_10 = num * 10; 22   int times_11 = num * 11; 23   int times_12 = num * 12; 24 25 // print multiplication table 26 27   printf("\n1 x %d = %d \n", num, times_1); 28   printf("2 x %d = %d \n", num, times_2); 29   printf("3 x %d = %d \n", num, times_3); 30   printf("4 x %d = %d \n", num, times_4); 31   printf("5 x %d = %d \n", num, times_5); 32   printf("6 x %d = %d \n", num, times_6); 33   printf("7 x %d = %d \n", num, times_7); 34   printf("8 x %d = %d \n", num, times_8); 35   printf("9 x %d = %d \n", num, times_9); 36   printf("10 x %d = %d \n", num, times_10); 37   printf("11 x %d = %d \n", num, times_11); 38   printf("12 x %d = %d", num, times_12); </pre>	<pre> Please enter a whole number: 30 1 x 30 = 30 2 x 30 = 60 3 x 30 = 90 4 x 30 = 120 5 x 30 = 150 6 x 30 = 180 7 x 30 = 210 8 x 30 = 240 9 x 30 = 270 10 x 30 = 300 11 x 30 = 330 12 x 30 = 360 </pre> <pre> Please enter a whole number: 2 1 x 2 = 2 2 x 2 = 4 3 x 2 = 6 4 x 2 = 8 5 x 2 = 10 6 x 2 = 12 7 x 2 = 14 8 x 2 = 16 9 x 2 = 18 10 x 2 = 20 11 x 2 = 22 12 x 2 = 24 </pre> <pre> Please enter a whole number: -86 1 x -86 = -86 2 x -86 = -172 3 x -86 = -258 4 x -86 = -344 5 x -86 = -430 6 x -86 = -516 7 x -86 = -602 8 x -86 = -688 9 x -86 = -774 10 x -86 = -860 11 x -86 = -946 12 x -86 = -1032 </pre>	

## Week 2 lecture 006 Notes- Arrays, Memory Window, Math, Random Numbers

Notes Retrieved from Steffan Hooper, W02 Lec 006, 2022

**11/03/2022 - Friday**

Arrays are like lists in python. They are declared like this: `int city_ages[360]`, the number between the [] square brackets is the array length. Arrays start from 0. Arrays can be in chars, floats, and ints. An element of an array is accessed like this: `city_ages[0]` and values can be assigned to it like this: `city_ages[0] = 18`

Arrays can be initialised like this: `int data[8] = {5, 10, 7, 1, -3, 5, -7, 7}`

Array elements are printed like this: `printf("data[0] is %d\n", data[0])`

Visual Studio Debugger: Memory Window can be used to see the elements of an array at different stages of the program.

KEYWORDS: `array`- a block of memory, `element`- a single array variable, `size`- number of elements array can store, `index`- a position in an array (index starts from 0 and end at size -1)

Variables can be used as the index of an array, example:

`printf("data[%d] holds %d\n", index, data[index++])` ← this is a post-increment

Array bugs- calling elements beyond the size of the array will cause errors. REMEMBER array sizes should be 1+ larger than the number of elements needed.

The ^ symbol is not the power (or exponent) in C.

Mathematical function: use `#include <math.h>` for access to several math functions:

`powf(2, 3)` → '2' is the base, '3' is the power. This is the same as  $2^3$ .

`Sqrtf(16.0f)` → same as  $\sqrt{16}$

`expf(1)` → same as  $e^1$

`logf(2.7f)` → same as  $\ln(2.7)$

`log10f(1000)` → same as  $\log_{10}(1000)$

`fabsf(-5)` → same as  $|-5|$

`floorf(3.75f)` → Computes the floor of 3.75f

`ceilf(3.75f)` → Computes the floor of 3.75

Trigonometric functions: `sinf(x), cosf(x), tanf(x)` → results are in radians

These are for float values (for double, remove the 'f' at the end, ex: `pow()`):

Process of generating random numbers:

1. Use `#include <stdlib.h>` and `#include <time.h>` at the start of the program.
2. Call `srand()` at the start of program (from `<stdlib.h>`)
3. Pass in the result using `time()` function (from `<time.h>`)
4. To get random number, call `rand()` function, example: `max random number, minimum`  
`int dice_roll = (rand() % 6) + 1`

REMINDER: only call `srand` once

# WEEK 3

## Week 3 lecture 007 Notes- Character Arrays, C-Strings

Notes Retrieved from Steffan Hooper, W03 Lec 007, 2022

14/03/2022 - Monday

Character Arrays can store a string of characters to represent word/s. Each element is a single ASCII value. Example:

char my_name[5]	
Array Index	Variable's Value
my_name[0]	'F' 70
my_name[1]	'r' 114
my_name[2]	'a' 97
my_name[3]	'n' 110
my_name[4]	'\0' 0 (character array terminated here)

%s (string) specifier will cause printf to print from a specified array of characters until it finds an element in the char array with the value 0. Null-terminator- the null character has the value 0 and is ASCII '\0'

Note: every other array is not terminated with zero

Initialising character array example:

```
char greeting[6] = {'h', 'e', 'l', 'l', 'o', '\0'} ← null-terminated using '\0'  
Initialising character array a simpler way:
```

```
char greeting[] = "hello" ← automatically null-terminated and array-sized (5) by compiler  
printing character array:  
printf("%s", greeting) ← Output: hello
```

Use **scanf** and **%s** to read user input into char array: **scanf ("%s", &username)**.

- There is no need for the address-of operator (&) when using arrays and scanf, and scanf will automatically null-terminate the C-String input.

**scanf ("%10s", &username)**. ← **restrict**: only read 10 characters from input

- When scanning with %s, scanf will stop parsing when it encounters white space input

**scanf ("%[A-Z]", &upper\_case\_only)** ← Stops scanning into the array when the next item is not an uppercase character.

**scanf ("%[A-Za-z]", &letter\_only)** ← Stops scanning into the array when next input is not an uppercase or lowercase character

**scanf ("%[0-9]", &numbers\_only)** ← Stop scanning into the array when the next input is not a digit character

**scanf ("%[^\\n]", &allowing\_spaces)** ← Stop scanning into the array when the next input is newline

Scanset Notation: \* (asterisk) in scanf specifier will stop scanf from saving input into variable

**scanf ("%\*[^\\n]")** ← \* means suppress assignment of the data read from the input buffer

## Week 3 lecture 007 Notes- Character Arrays, C-Strings (Continued)

Notes Retrieved from Steffan Hooper, W03 Lec 007, 2022

14/03/2022 - Monday

`#include <string.h>` to access :

the `strlen` function. It is like `len()` in python!

`int length = strlen(example_string)` ← `strlen` takes in a char array and returns the number of characters in the C-String

The `strcpy` function. Copy arrays.

`strcpy(destination, source)` ← `strcpy` takes in two char array parameters, the first one is the destination to copy to, the second one is the source to copy from.

The `strcat` function. Links c-strings together. Example of its use with `strcpy`:

```
strcpy(full_name, first_name); ← Firstly, copy into the full_name array  
strcat(full_name, " "); ← Then concatenate a space on the end  
strcat(full_name, last_name); ← Then concatenate another string on the end  
printf("Full Name: %s\n", full_name); ← Result is printed like this
```

Note: The length of a C-String is not the same as the size of the char array in which it is stored

the `atoi()` function stands for “ASCII to int”. How ASCII is converted to int in code:

```
char digits[10];  
scanf("%9s", &digits); ← Read in ASCII characters  
int number = 0;  
number = atoi(digits) ← Convert ASCII characters into an integer
```

the `atof()` function stands for “ASCII to floating-point”. Example conversion:

```
number = atof(digits);
```

`sprintf()` is used to print char array and convert an integer number into a C-String. Example:

```
sprintf(digits, "%d", number) ← 'digits' is the array to print into, 'number' is the integer variable
```

**LAB 3: 3x01a – Input char, Output ASCII**

14/03/2022 – Monday, Week 3

Program aim		
Ask user to input a single letter, digit, or symbol and output the ASCII value of the character in decimal (base-10) and hexadecimal (base-16)		
Input-Process-Output plan		
User Input (variables)	Process	Output
value as char	Output as decimal using %d	The ASCII value of the user-inputted character is printed as a decimal and hexadecimal
	Output as hexadecimal using %x	
Implementation		
Source Code (Input)		Output/s
<pre> 1 #include &lt;stdio.h&gt; 2 3 int main(void) 4 { 5     char value; 6 7     printf("Please input a letter, digit or symbol: "); 8     scanf(" %c", &amp;value); 9 10    printf("\nThe ASCII value for %c in base-10 is %d \n", value, value); 11    printf("The ASCII value for %c in base-16 is 0x%x ", value, value); 12 13    return 0; 14 }</pre>		<pre> Please input a letter, digit or symbol: ! The ASCII value for ! in base-10 is 33 The ASCII value for ! in base-16 is 0x21</pre> <pre> Please input a letter, digit or symbol: 5 The ASCII value for 5 in base-10 is 53 The ASCII value for 5 in base-16 is 0x35</pre> <pre> Please input a letter, digit or symbol: f The ASCII value for f in base-10 is 102 The ASCII value for f in base-16 is 0x66</pre>
Lessons learned		
REMEMBER: %x is the formatting placeholder for printing hexadecimal values.		

**LAB 3: 3x02a – Lower to Upper**

14/03/2022 – Monday, Week 3

Program aim			
Ask user to input single lowercase letter, then output the letter as a capital and output the decimal ASCII values of the letter in lowercase and uppercase.			
User Input (variables)		Process	Output
<code>letter</code> as char		Convert to uppercase using formula: <code>letter - 32</code> . Save result to variable <code>upper_letter</code>	Print the uppercase version of the letter user input, and print the value of the input/uppercase version as base-10
Test Cases for alphabet			
Case#	Input letter	Output	
		<code>upper_letter</code> as char	<code>letter</code> as decimal
1	a	A	97
2	b	B	98
3	c	C	99
4	d	D	100
5	e	E	101
6	f	F	102
7	g	G	103
8	h	H	104
9	i	I	105
10	j	J	106
11	k	K	107
12	l	L	108
13	m	M	109
14	n	N	110
15	o	O	111
16	p	P	112
17	q	Q	113
18	r	R	114
19	s	S	115
20	t	T	116
21	u	U	117
22	v	V	118
23	w	W	119
24	x	X	120
25	y	Y	121
26	z	Z	122

**LAB 3: 3x02a – Lower to Upper (Continued)**

14/03/2022 – Monday, Week 3

Implementation	
Source Code (Input)	Output/s
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      char letter; 6 7      printf("===== \n"); 8      printf("Lowercase to Uppercase Converter: \n"); 9      printf("===== \n\n"); 10 11     printf("Please input a lowercase letter: "); 12     scanf("%c", &amp;letter); 13 14     char upper_letter = letter - 32; 15     printf("\nThe uppercase version is %c. \n\n", upper_letter); 16 17     printf("The ASCII value of %c is %d in base-10. \n\n", letter, letter); 18 19     printf("The ASCII value of %c is %d in base-10.", upper_letter, upper_letter); 20 21     return 0; 22 }</pre>	<pre> ===== Lowercase to Uppercase Converter: =====  Please input a lowercase letter: a  The uppercase version is A.  The ASCII value of a is 97 in base-10.  The ASCII value of A is 65 in base-10.  ===== Lowercase to Uppercase Converter: =====  Please input a lowercase letter: l  The uppercase version is L.  The ASCII value of l is 108 in base-10.  The ASCII value of L is 76 in base-10.  ===== Lowercase to Uppercase Converter: =====  Please input a lowercase letter: z  The uppercase version is Z.  The ASCII value of z is 122 in base-10.  The ASCII value of Z is 90 in base-10.</pre>
(The output was correct for all 26 alphabet letters)	

**LAB 3: 3x03a – Average using an Array**

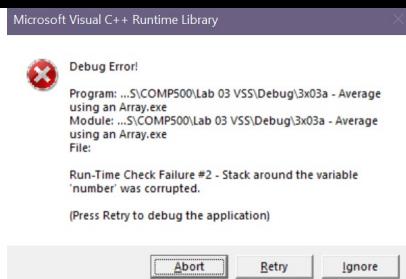
14/03/2022 – Monday, Week 3

**Program aim**

Ask user to input 5 float values, use float array to store and output the numbers. Calculate and output the average of the numbers.

**Input-Process-Output plan**

User Input number [5]	Process	Output
number [0] as float	Average of 5 float values: $(\text{number}[0]+\text{number}[1]+\text{number}[2]+\text{number}[3]+\text{number}[4]) / 5$	Input is printed back to the user followed by the average of the 5 inputted float values.
number [1] as float		
number [2] as float		
number [3] as float		
number [4] as float	Result is saved to float variable <b>average</b>	

**Testing/errors/debugging**

Even though the code runs all the way through correctly, I get the following debug error when it is finished. This happened when I set the number array to **number [4]** instead of **number [5]**. (line 5)

This error happened because even though there are 5 inputs (arrays start from 0, so 4 would be 5 inputs), it is not considered big enough to store the data in the array.

**Implementation**

Source Code (lines 5-33)	Output/s
<pre> 5   float number[5]; 6 7   printf("Please enter five real numbers: \n\n"); 8 9   printf(" First number: "); 10  scanf("%f", &amp;number[0]); 11 12  printf(" Second number: "); 13  scanf("%f", &amp;number[1]); 14 15  printf(" Third number: "); 16  scanf("%f", &amp;number[2]); 17 18  printf(" Fourth number: "); 19  scanf("%f", &amp;number[3]); 20 21  printf(" Fifth number: "); 22  scanf("%f", &amp;number[4]); 23 24  printf("\nThe user entered: \n\n"); 25  printf(" %f then, \n", number[0]); 26  printf(" %f then, \n", number[1]); 27  printf(" %f then, \n", number[2]); 28  printf(" %f then, \n", number[3]); 29  printf(" %f \n\n", number[4]); 30 31  float average = (number[0]+number[1]+number[2]+number[3]+number[4]) / 5; 32 33  printf("The average of these five numbers is: %f", average); </pre>	<pre> Please enter five real numbers: First number: 77.519997 Second number: 51.090000 Third number: 49.169998 Fourth number: 37.590000 Fifth number: 8.320000  The user entered: 77.519997 then, 51.090000 then, 49.169998 then, 37.590000 then, 8.320000  The average of these five numbers is: 44.737999  Please enter five real numbers: First number: 30.180000 Second number: 89.589996 Third number: 38.169998 Fourth number: 90.160004 Fifth number: 61.480000  The user entered: 30.180000 then, 89.589996 then, 38.169998 then, 90.160004 then, 61.480000  The average of these five numbers is: 61.916004 </pre>

**Lessons learned**

Always set the array sizes 1 larger than expected to prevent the debug error: "stack around variable was corrupted"

**LAB 3: 3x04a – Buggy Array**

14/03/2022 – Monday

**Program aim**

Find the bug in the given source code, and fix it

**Purpose of program**

Halve the number 1, and halve each result four times, and print the output of each halve.

**Implementation (comparison to given source code)****Given Source Code**

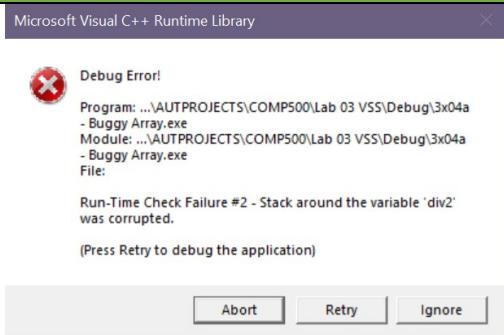
```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     float div2[5];
6
7     div2[1] = 1.0f;
8     div2[2] = div2[1] / 2.0f;
9     div2[3] = div2[2] / 2.0f;
10    div2[4] = div2[3] / 2.0f;
11    div2[5] = div2[4] / 2.0f;
12
13    printf("Starting at %.2f\n", div2[1]);
14    printf("Halved... %.2f\n", div2[2]);
15    printf("Halved... %.2f\n", div2[3]);
16    printf("Halved... %.2f\n", div2[4]);
17    printf("Halved... %.2f\n", div2[5]);
18
19    printf("\n");
20
21    return 0;
22 }
```

**My Source Code**

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     float div2[6]; // < The bug was that the array was not long enough
6
7     div2[1] = 1.0f;
8     div2[2] = div2[1] / 2.0f;
9     div2[3] = div2[2] / 2.0f;
10    div2[4] = div2[3] / 2.0f;
11    div2[5] = div2[4] / 2.0f;
12
13    printf("Starting at %.2f\n", div2[1]);
14    printf("Halved... %.2f\n", div2[2]);
15    printf("Halved... %.2f\n", div2[3]);
16    printf("Halved... %.2f\n", div2[4]);
17    printf("Halved... %.2f\n", div2[5]);
18
19    printf("\n");
20
21 }
```

**Output comparison****Given output****My output**

```

Starting at 1.00
Halved... 0.50
Halved... 0.25
Halved... 0.13
Halved... 0.06

```

**Bug**

The bug is that the array **div2 [5]** was too short, thus causing the error: "stack around the variable 'div2' was corrupted". To fix this, I simply had to increase the array size to 6: **div2 [6]**.

## Week 3 lecture 008 Notes- Coding Standards, Commenting, Flowcharts, Pseudo Code

Notes Retrieved from Steffan Hooper, W03 Lec 008, 2022

16/03/2022 – Wednesday

Consider good practice in coding: White Space, Layout, Naming, Declaration, and using `const` to avoid ‘magic numbers’. To view whitespace in code, go Edit -> Advanced -> View White Space.

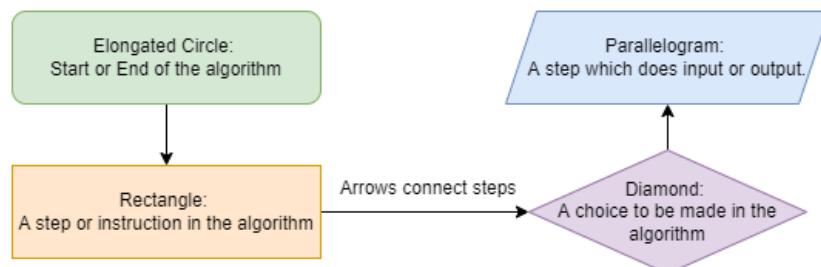
Avoid declaring many variables in one statement: `int x, y, z;`

Avoid making similar-looking variables as this can get confusing

### **Commenting styles:**

```
/*
    Comment style #1
    this can go over several lines
*/
// Comment style #2- this is a single-line comment
```

### **Flowchart symbols (made using draw.io as I am already familiar with this website)**



Flowchart Practice: Can flow top to bottom or left to right of page, always have one start and end, process blocks always have one arrow in and one arrow out, decision arrows are always labelled with choices, input/output blocks are labelled READ or WRITE.

Design Practices: input-process-output tables, numbered lists, flowcharts, pseudocode.

### **Pseudocode example**

Do COMP500/ENSE501 Lab Activities:

```
Get pen.  
Read the lab slides.  
Go through all problem exercises:  
    Complete problem.  
    While the problem is incomplete or wrong:  
        Write journal notes, try again.  
Proofread reporting journal entry.
```

## Week 3 lecture 008 Notes- Coding Standards, Commenting, Flowcharts, Pseudo Code (continued)

Notes Retrieved from Steffan Hooper, W03 Lec 008, 2022

**16/03/2022 – Wednesday**

Pseudocode Practice: use clear, simple steps. Choices (IF, ENDIF) are indented. Actions to be repeated (WHILE, ENDWHILE, REPEAT, UNTIL) are indented. Pseudocode can be written as comments, then code can be implemented using the pseudocode comments as a guide.

Commenting practice: avoid TOO many comments. Every line of source code doesn't need comments. Some lines of code benefit from a short description of what it does.

**// TODO: comments are meant to set reminders of what to develop. Example:**

```
// TODO: Declare an int variable.  
// TODO: Print out a greeting.  
// TODO: Ask for the user input.  
// TODO: Read in the user input.  
// TODO: Compute the square of the user input.  
// TODO: Print out the result.
```

To view the TODO comment task list in the IDE, go view > other windows > task list or press ctrl+\, T

Debugging: “Rubber ducking” involves finding the cause of a bug by explaining my source code step-by-step until I realise what the error is!

## LAB 3: 3x05a – Powers Printer

17/03/2022 – Thursday (lab) , Week 3

Program aim		
Input-Process-Output plan		
User Input (variables)	Process	Output
<b>Base</b> as float	Calculate powers of <b>base</b> from 0-8 using <b>powf</b> : <b>powf(base, 0)</b> to <b>powf(base, 8)</b>	Powers 0-8 of <b>base</b> is printed to the user
Testing/errors/debugging		
<p>Although the input and output are whole numbers, I needed to make the variable <b>base</b> a float in order to use the <b>math.h</b> operation <b>powf</b>. Otherwise, if I make <b>base</b> int, I get the truncation error:</p> <pre>warning C4244: 'function': conversion from 'int' to 'float', possible loss of data</pre>		
Without #include <math.h>		
<pre>1 #include &lt;stdio.h&gt; 2 3 int main(void) 4 { 5     int base; 6     int powers[9]; 7 8     printf("Please enter a base: "); 9     scanf("%d", &amp;base); 10 11    powers[0] = 1; // power to 0 is always 1 12    powers[1] = base; // power to 1 is always the base number 13    powers[2] = base * base; 14    powers[3] = base * base * base; 15    powers[4] = base * base * base * base; 16    powers[5] = base * base * base * base * base; 17    powers[6] = base * base * base * base * base * base; 18    powers[7] = base * base * base * base * base * base * base; 19    powers[8] = base * base; 20 21    printf("\nThe powers of %d are: \n\n", base); 22 23    printf("%d ^ 0 is %d \n", base, powers[0]); 24    printf("%d ^ 1 is %d \n", base, powers[1]); 25    printf("%d ^ 2 is %d \n", base, powers[2]); 26    printf("%d ^ 3 is %d \n", base, powers[3]); 27    printf("%d ^ 4 is %d \n", base, powers[4]); 28    printf("%d ^ 5 is %d \n", base, powers[5]); 29    printf("%d ^ 6 is %d \n", base, powers[6]); 30    printf("%d ^ 7 is %d \n", base, powers[7]); 31    printf("%d ^ 8 is %d \n", base, powers[8]); 32 33    return 0; 34 }</pre>		
With #include <math.h>		
<pre>1 #include &lt;stdio.h&gt; 2 #include &lt;math.h&gt; 3 4 int main(void) 5 { 6     float base; 7 8     printf("Please enter a base: "); 9     scanf("%f", &amp;base); 10 11     printf("\nThe powers of %.f are: \n\n", base); 12 13     printf("%.f ^ 0 is %.f \n", base, powf(base, 0)); 14     printf("%.f ^ 1 is %.f \n", base, powf(base, 1)); 15     printf("%.f ^ 2 is %.f \n", base, powf(base, 2)); 16     printf("%.f ^ 3 is %.f \n", base, powf(base, 3)); 17     printf("%.f ^ 4 is %.f \n", base, powf(base, 4)); 18     printf("%.f ^ 5 is %.f \n", base, powf(base, 5)); 19     printf("%.f ^ 6 is %.f \n", base, powf(base, 6)); 20     printf("%.f ^ 7 is %.f \n", base, powf(base, 7)); 21     printf("%.f ^ 8 is %.f \n", base, powf(base, 8)); 22 23     return 0; 24 }</pre>		
<p>I accidentally went ahead with the program while forgetting about <b>&lt;math.h&gt;</b></p>		
<i>Much fewer lines than before!</i>		

**LAB 3: 3x05a – Powers Printer (Continued)**

17/03/2022 – Thursday (lab) , Week 3

Implementation	
Source Code (Input)	Output/s
<pre> 1  #include &lt;stdio.h&gt; 2  #include &lt;math.h&gt; 3 4  int main(void) 5  { 6      float base; 7 8      printf("Please enter a base: "); 9      scanf("%f", &amp;base); 10 11     printf("\nThe powers of %.f are: \n\n", base); 12 13     printf("%.f ^ 0 is %.f \n", base, powf(base, 0)); 14     printf("%.f ^ 1 is %.f \n", base, powf(base, 1)); 15     printf("%.f ^ 2 is %.f \n", base, powf(base, 2)); 16     printf("%.f ^ 3 is %.f \n", base, powf(base, 3)); 17     printf("%.f ^ 4 is %.f \n", base, powf(base, 4)); 18     printf("%.f ^ 5 is %.f \n", base, powf(base, 5)); 19     printf("%.f ^ 6 is %.f \n", base, powf(base, 6)); 20     printf("%.f ^ 7 is %.f \n", base, powf(base, 7)); 21     printf("%.f ^ 8 is %.f \n", base, powf(base, 8)); 22 23     return 0; 24 }</pre>	<pre> Please enter a base: 0 The powers of 0 are: 0 ^ 0 is 1 0 ^ 1 is 0 0 ^ 2 is 0 0 ^ 3 is 0 0 ^ 4 is 0 0 ^ 5 is 0 0 ^ 6 is 0 0 ^ 7 is 0 0 ^ 8 is 0 Please enter a base: 14 The powers of 14 are: 14 ^ 0 is 1 14 ^ 1 is 14 14 ^ 2 is 196 14 ^ 3 is 2744 14 ^ 4 is 38416 14 ^ 5 is 537824 14 ^ 6 is 7529536 14 ^ 7 is 105413504 14 ^ 8 is 1475789056 Please enter a base: 7 The powers of 7 are: 7 ^ 0 is 1 7 ^ 1 is 7 7 ^ 2 is 49 7 ^ 3 is 343 7 ^ 4 is 2401 7 ^ 5 is 16807 7 ^ 6 is 117649 7 ^ 7 is 823543 7 ^ 8 is 5764801 </pre>

**Lessons learned**

When using the `powf` operation from `math.h`, ensure the values used in the math operation are floats.

Because the '^' symbol has a different usage, Power formulas cannot be written like 'number ^ 3', but only as 'number \* number \* number'

When using `#include <math.h>`, `powf(x, y)` can be used to calculate powers, which is much simpler than calculating the value of a large power number with the original method

I also learned that calculations can be integrated into the `printf` lines instead separate lines, thus simplifying the code structure into fewer lines.

## LAB 3: 3x06a – D20 Dice Roller

17/03/2022 – Thursday (lab) , Week 3

Program aim	
Input-Process-Output plan	
Process	Output
Get access to <code>srand</code> function: <code>#include &lt;stdlib.h&gt;</code>	Output to the user two random die rolls with values being between 1 and 20
Pass result from <code>time()</code> function from: <code>#include &lt;time.h&gt;</code>	
'seed' random number generator: <code>srand(time(0))</code>	
Generate random number between 1-20 twice: <code>(rand() % 20) + 1</code>	
Save result to integer variables: <code>roll1</code> and <code>roll2</code>	
Implementation	
Source Code (Input)	Output/s
<pre> 1  #include &lt;stdio.h&gt; 2  #include &lt;stdlib.h&gt; 3  #include &lt;time.h&gt; 4 5  int main(void) 6  { 7      printf("D20 Dice Roller: \n"); 8      printf("-----\n\n"); 9 10     // seed random generator 11     srand(time(0)); 12 13     // generate first and second die roll 14     int roll1 = (rand() % 20) + 1; 15     int roll2 = (rand() % 20) + 1; 16 17     printf("The first die rolls the value: %d", roll1); 18     printf("\n\nThen... \n\n"); 19     printf("The second die rolls the value: %d", roll2); 20 21     return 0; 22 }</pre>	<pre>D20 Dice Roller: ----- The first die rolls the value: 3 Then... The second die rolls the value: 10</pre> <pre>D20 Dice Roller: ----- The first die rolls the value: 16 Then... The second die rolls the value: 14</pre> <pre>D20 Dice Roller: ----- The first die rolls the value: 8 Then... The second die rolls the value: 20</pre>
Lessons learned	
I learned how to use random for the first time. I have to remember to use <code>#include &lt;stdlib.h&gt;</code> and <code>&lt;time.h&gt;</code> to use the random number generator.	

**LAB 3: 3x07a – Five Dice**

17/03/2022 – Thursday (lab) , Week 3

Program aim																																																													
Simulate the rolling of five six-sided dice and output in a specific style																																																													
Input-Process-Output plan																																																													
Process	Output																																																												
Get access to <code>srand</code> function: <code>#include &lt;stdlib.h&gt;</code>	Output to the user 5 random die rolls with values being between 1 and 6																																																												
Pass result from <code>time()</code> function from: <code>#include &lt;time.h&gt;</code>																																																													
'seed' random number generator: <code>srand(time(0))</code>																																																													
Generate random number between 1-6 five times: <code>(rand() % 6) + 1</code>																																																													
Save result to integer variables: <code>roll1 ...to... roll5</code>																																																													
Implementation																																																													
Source Code (Input)	Output/s																																																												
<pre> 1  #include &lt;stdio.h&gt; 2  #include &lt;stdlib.h&gt; 3  #include &lt;time.h&gt; 4 5  int main(void) 6  { 7      // seed random generator 8      srand(time(0)); 9 10     // generate 5 die rolls 11     int roll1 = (rand() % 6) + 1; 12     int roll2 = (rand() % 6) + 1; 13     int roll3 = (rand() % 6) + 1; 14     int roll4 = (rand() % 6) + 1; 15     int roll5 = (rand() % 6) + 1; 16 17     printf("Dice1  Dice2  Dice3  Dice4  Dice5 \n"); 18     printf("+-+-+ +-++ +---+ +--+ +---+ \n"); 19     printf("  %d     %d     %d     %d   \n", roll1, roll2, roll3, roll4, roll5); 20     printf("+-+-+ +-++ +---+ +--+ +---+ \n"); 21 22     return 0; 23 }</pre>	<table border="1"> <thead> <tr> <th>Dice1</th><th>Dice2</th><th>Dice3</th><th>Dice4</th><th>Dice5</th></tr> </thead> <tbody> <tr> <td>+</td><td>-</td><td>-</td><td>-</td><td>-</td></tr> <tr> <td>  5  </td><td>  5  </td><td>  2  </td><td>  5  </td><td>  2  </td></tr> <tr> <td>+</td><td>-</td><td>-</td><td>-</td><td>-</td></tr> </tbody> </table> <table border="1"> <thead> <tr> <th>Dice1</th><th>Dice2</th><th>Dice3</th><th>Dice4</th><th>Dice5</th></tr> </thead> <tbody> <tr> <td>+</td><td>-</td><td>-</td><td>-</td><td>-</td></tr> <tr> <td>  2  </td><td>  4  </td><td>  2  </td><td>  5  </td><td>  1  </td></tr> <tr> <td>+</td><td>-</td><td>-</td><td>-</td><td>-</td></tr> </tbody> </table> <table border="1"> <thead> <tr> <th>Dice1</th><th>Dice2</th><th>Dice3</th><th>Dice4</th><th>Dice5</th></tr> </thead> <tbody> <tr> <td>+</td><td>-</td><td>-</td><td>-</td><td>-</td></tr> <tr> <td>  3  </td><td>  1  </td><td>  4  </td><td>  5  </td><td>  4  </td></tr> <tr> <td>+</td><td>-</td><td>-</td><td>-</td><td>-</td></tr> </tbody> </table>	Dice1	Dice2	Dice3	Dice4	Dice5	+	-	-	-	-	5	5	2	5	2	+	-	-	-	-	Dice1	Dice2	Dice3	Dice4	Dice5	+	-	-	-	-	2	4	2	5	1	+	-	-	-	-	Dice1	Dice2	Dice3	Dice4	Dice5	+	-	-	-	-	3	1	4	5	4	+	-	-	-	-
Dice1	Dice2	Dice3	Dice4	Dice5																																																									
+	-	-	-	-																																																									
5	5	2	5	2																																																									
+	-	-	-	-																																																									
Dice1	Dice2	Dice3	Dice4	Dice5																																																									
+	-	-	-	-																																																									
2	4	2	5	1																																																									
+	-	-	-	-																																																									
Dice1	Dice2	Dice3	Dice4	Dice5																																																									
+	-	-	-	-																																																									
3	1	4	5	4																																																									
+	-	-	-	-																																																									

## LAB 3: 3x08a – Inputting a C-string

17/03/2022 – Thursday (lab) , Week 3

Program aim	
Input-Process-Output plan	
User Input (variables)	Output
fave_colour as string (limit to 10 chars)	"You like the colour fave_colour"
Implementation	
Source Code (Input)	Output/s
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      char fave_colour[10] = ""; 6 7      printf("What is your favourite colour? "); 8      scanf("%s", &amp;fave_colour); 9 10     printf("You like the colour %s.", fave_colour); 11 12     return 0; 13 }</pre>	<pre> What is your favourite colour? red You like the colour red.  What is your favourite colour? nothing You like the colour nothing.  What is your favourite colour? purple You like the colour purple.  What is your favourite colour? orange You like the colour orange.</pre>
Lessons learned	
I learned to implement string input and output for the first time!	

## LAB 3: 3x09a – String Length

17/03/2022 – Thursday (lab) , Week 3

### Program aim

Ask user to input their first name. Read this input into a char array. Ensure the char array is at least 20 elements in size.

### Input-Process-Output plan

User Input (variables)	Output
<code>first_name</code> as string (limit to 20 chars)	" <code>first_name</code> contains <code>name_length</code> characters"

### Testing/errors/debugging

I forgot to create the size of the `first_name` array, thus creating the error: *run-time check failure- stack around the variable 'first\_name' was corrupted*. I made this mistake due to being in the habit of not having to declare the size of lists in python.

### Implementation

Source Code (Input)	Output/s
<pre> 1  #include &lt;stdio.h&gt; 2  #include &lt;string.h&gt; 3 4  int main(void) 5  { 6      char first_name[20] = ""; 7 8      printf("What is your first name? "); 9      scanf("%s", &amp;first_name); 10 11     int name_length = strlen(first_name); 12 13     printf("%s contains %d characters", first_name, name_length); 14 15     return 0; 16 }</pre>	<p>What is your first name? Steffan Steffan contains 7 characters</p> <p>What is your first name? Xinyu Xinyu contains 5 characters</p> <p>My own name: What is your first name? Francisca Francisca contains 9 characters</p>

### Lessons learned

I learnt to implement `strlen` function from `string.h` for the first time!

Remember to declare the size of the array when formatting char arrays like this:

`Char char_array[size] = ""`

## LAB 3: 3x10a – C-String to int

17/03/2022 – Thursday (lab) , Week 3

### Program aim

Based upon the following numbered list, which outlines the program's design, implement the program

### Plan steps (given by exercise)

1. Declare an integer variable named **whole\_number**.
2. Assign a value of zero to **whole\_number**.
3. Declare a char array named **text** of size 80.
4. Print "> " to the console.
5. Read in keyboard input as text, using **%79s**, saving into the text array.
6. Call **atoi()** to convert the ASCII **text** values into an integer, save the result of this conversion into the **whole\_number** variable.
7. Print out "C-String: ".
8. Print out the **text** array using **%s**.
9. Print a newline.
10. Print out "int: ".
11. Print out the **whole\_number** integer using **%d**.
12. Print a newline.

### Testing/errors/debugging

*Place a breakpoint on line 12 of the numbered list. Run the program to this breakpoint with the debugger.*

*Place the variables **whole\_number** and **text** into the Watch Window – notice their types, and the data they store.*

Source code & breakpoint	Watch Window																								
<pre> 1  #include &lt;stdio.h&gt; 2  #include &lt;stdlib.h&gt; 3 4  int main(void) 5  { 6      int whole_number = 0; 7      char text[80] = ""; 8 9      printf("&gt; "); 10     scanf("%79s", &amp;text); 11 12     whole_number = atoi(text); 13 14     printf("C-String: %s \nint: %d \n", text, whole_number); 15 16     return 0; 17 }</pre>	<p>When nothing is entered:</p> <p>Watch 1</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Name</th> <th>Value</th> <th>Type</th> </tr> </thead> <tbody> <tr> <td>whole_number</td> <td>0</td> <td>int</td> </tr> <tr> <td>text</td> <td>0x00d5f90c ""</td> <td>char[80]</td> </tr> <tr> <td>[0]</td> <td>0 '\0'</td> <td>char</td> </tr> <tr> <td>[1]</td> <td>0 '\0'</td> <td>char</td> </tr> <tr> <td>[2]</td> <td>0 '\0'</td> <td>char</td> </tr> <tr> <td>[3]</td> <td>0 '\0'</td> <td>char</td> </tr> <tr> <td>[4]</td> <td>0 '\0'</td> <td>char</td> </tr> </tbody> </table> <p>Whole_number has the set '0' int value.</p> <p>The entire list array seems to have automatically been arranged null '\0' to all 80 elements</p>	Name	Value	Type	whole_number	0	int	text	0x00d5f90c ""	char[80]	[0]	0 '\0'	char	[1]	0 '\0'	char	[2]	0 '\0'	char	[3]	0 '\0'	char	[4]	0 '\0'	char
Name	Value	Type																							
whole_number	0	int																							
text	0x00d5f90c ""	char[80]																							
[0]	0 '\0'	char																							
[1]	0 '\0'	char																							
[2]	0 '\0'	char																							
[3]	0 '\0'	char																							
[4]	0 '\0'	char																							

### LAB 3: 3x10a – C-String to int (Continued)

17/03/2022 – Thursday (lab) , Week 3

Implementation	
Source Code (Input)	Output/s
<pre>1 #include &lt;stdio.h&gt; 2 #include &lt;stdlib.h&gt; 3 4 int main(void) 5 { 6     int whole_number = 0; 7 8     char text[80] = ""; 9 10    printf("&gt; "); 11    scanf("%79s", &amp;text); 12 13    whole_number = atoi(text); 14 15    printf("C-String: %s \nint: %d \n", text, whole_number); 16 17    return 0; 18 }</pre>	<pre>&gt; 18 C-String: 18 int: 18</pre>
	<pre>&gt; 4 C-String: 4 int: 4</pre>
	<pre>&gt; hi C-String: hi int: 0</pre>
	<pre>&gt; 82917391 C-String: 82917391 int: 82917391</pre>

#### Lessons learned

I learnt how to implement `atoi()` from `<stdlib.h>` for the first time!  
`atoi()` is used for converting c-strings to integers.

## Week 3 lecture 009 Notes- Selection, if, else, Relational Operators

Notes Retrieved from Steffan Hooper, W03 Lec 009, 2022

**18/03/2022 - Friday**

C programming **if**, **else** functions similarly to python. Basic structure:

IF "condition is true",  
THEN "perform action X",  
ELSE "perform action Y".

If else statements can be displayed via pseudocode and flowcharts, for example:

```
PRINT How old are you?  
IF Age greater than 18? THEN  
PRINT You may enter the club!  
ELSE  
PRINT You may not enter the club!  
ENDIF
```

If else statements are contained within scope braces {}, expression is contained within () brackets.

Relational Operators- You can compare literals and variables	
a == b	a equals b
a != b	a does not equal b
a > b	a greater than b
a >= b	a greater than or equal to b
a < b	a less than b
a <= b	a less than or equal to b

Char literals: " single quote, string literals: "" double quote

Comparing C-strings: **strcmp** is from **#include <string.h>**. It is used to compare strings: Give strcmp two C-Strings, if they match, it evaluates to zero. Example of use:

```
if (strcmp(firstname, "Fran") == 0)  
{  
    printf("It's me!\n");  
}  
else  
{  
    printf("Hello %s.\n", firstname);  
}
```

# WEEK 4

## Week 4 lecture 010 Notes- Nested Selection, else if, Logical Operators

Notes Retrieved from Steffan Hooper, W04 Lec 010, 2022

21/03/2022 - Monday

Nested if statements: to place an if else statement within an if else statement:

```
if (x < 50) ← Outer if
{
    printf("x is less than 50");

    if (x > 10) ← inner if
    {
        printf(" and x is greater than 10.\n");
    }
}
```

The diamond symbol in flowcharts can be used to indicate nested if statements.

### **Nested if statements in Pseudocode:**

```
START
    PRINT Tell me a number...
    IF Is number equal to 0? THEN
        PRINT The number is zero!
    ELSE
        IF Is number less than 0? THEN
            PRINT The number is negative!
        ELSE
            PRINT The number is positive!
        ENDIF
    ENDIF
END
```

Note the use of indentation.

Logical Boolean operators: AND is **&&**, OR is **||**, NOT is **!**

### **Examples of Logical Boolean operators use:**

AND: Is **number** greater than zero, but also less than 10?

```
if (number > 0 && number < 10)
{
    // Execute this if true...
}
```

OR and NOT: Is **number** NOT less than or equal to 0, or greater than or equal to 10?

```
if (!(number <= 0 || number >= 10))
{
    // Execute this if true...
}
```

DON'T DO THIS: **10 < x < 100** as it evaluates as **((10 < x) < 100)**

Use logical operators instead: **10 < x && x < 100** as it evaluates as **((10 < x) && (x < 100))**

## Week 4 lecture 011 Notes- Common Selection Bugs, switch, Conditional Operator

Notes Retrieved from Steffan Hooper, W04 Lec 011, 2022

23/03/2022 - Wednesday

**REMINDER:** Don't do logical operators like this: `input == 'y' || 'Y'`

Do them like this: `input == 'y' || input == 'Y'`

**Null statement:** ;;;;; or {}{}{}{}{} are empty statements the program will run and compile.

### **Keywords**

- Switch: Checks value of variable/expression against a list of case values
- Case: like 'if', it checks whether a variable/expression matches to a certain value.
- Break: each case needs to be terminated using 'break'
- Conditional operator: a syntax style for the selection operation

### EXAMPLE OF SWITCH & CASE FORMATTING

```
int i = 0;  
switch (i) → Switch on the value of the variable i  
{  
case 0: → If i holds the value 0...  
printf("i holds zero... \n");  
break;  
...  
When switch finishes, program continues.  
Switches can be nested.
```

Fall through is where a case falls through to the next case:

`Case 'Y': // Fall Through` → REMEMBER to comment this

`Case 'y':` → The 'Y' case falls through to the 'y' case

**Default:** like `else`, if no cases are executed, `default` case will be executed instead. This comes last.

**COMPARISON:** `if` allows you to evaluate a relational (ex. `>` or `<` or `>=`), and utilise logical operators (`&&`, `||`, `!`) to compare multiple comparison results while switch simply does a one-to-one comparison

#### **Example of conditional operator syntax:**

`condition ? value if condition is true : value if condition is false;` SAME AS:

If `condition` is TRUE

Execute `value if condition is true;`

Else

Execute `value if condition is false;`

Or use mnemonic `W ? T : F = What ? True : False`

Conditional breakpoints (debugger stops when condition is met ex. `variable == 10`):

Right click breakpoint > condition... > breakpoint condition.

**LAB 4: 4x01a – Pseudo Code to Source Code**

23/03/2022 – Wednesday , Week 4

**Program aim**

Implement a C program based upon the following pseudo code design:

**Given Pseudocode (from exercise)**

```

PRINT 'Enter the x value: '
READ x
PRINT 'Enter the y value: '
READ y
COMPUTE a AS x * y
COMPUTE b AS x + y
COMPUTE result AS b^2 + a * (b - x) * (a + y)
PRINT 'Computed: '
PRINT result

```

**Implementation****Source Code (lines 4-33)**

```

4 int main(void)
5 {
6     float x;
7     float y;
8
9     // PRINT 'Enter the x value: '
10    printf("Enter the x value: ");
11
12    // READ x
13    scanf("%f", &x);
14
15    // PRINT 'Enter the y value: '
16    printf("\nEnter the y value: ");
17
18    // READ y
19    scanf("%f", &y);
20
21    // COMPUTE a AS x * y
22    float a = x * y;
23
24    // COMPUTE b AS x + y
25    float b = x + y;
26
27    // COMPUTE result AS b^2 + a * (b-x) * (a+y)
28    float result = powf(b, 2) + a * (b - x) * (a + y);
29
30    // PRINT 'Computed: ' and PRINT result
31    printf("\nComputed: %f", result);
32
33    return 0;

```

**Output/s**

Enter the x value: -3  
 Enter the y value: -89  
 Computed: -4221350.000000

Enter the x value: 80  
 Enter the y value: -12  
 Computed: -11192816.000000

Enter the x value: 0  
 Enter the y value: 4  
 Computed: 16.000000

 100 / 100 Tests Passed
**Lessons learned**

I learned how to use pseudocode design to implement a program. Putting the pseudocode as comments is a great way to guide me through each step of the implementation, however, it takes up a lot of lines in the source code if every step is commented!

**LAB 4: 4x02a – Broken Rectangle Calculator**

23/03/2022 – Wednesday , Week 4

**Program aim**

Find the bug in the given source code, and fix it

**Purpose of program**

Take user input of width and height of a rectangle and then calculate and output the rectangle's perimeter and area

**Implementation (comparison to given source code)****Given Source Code**

```

1 #include <stdio.h>
2
3 int main()
4 {
5     float width = 0.0f;
6     float height = 0.0f;
7
8     scanf("%f", &width);
9     scanf("%f", &height)
10
11    printf("Rectangle\'s Width: ");
12    /* printf("Rectangle\'s Height: ")
13
14    float area = width * height;*/
15    float perimeter = width + width + height + height;
16
17    printf("Area is %f\n", perimeter);
18    /* printf("Perimeter is %f\n", area); */
19
20    return 0;
21 }
```

**My Source Code**

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     float width = 0.0f;
6     float height = 0.0f;
7
8 // PRINT "Rectangle's width: "
9     printf("Rectangle\'s Width: ");
10 // SCAN width
11     scanf("%f", &width);
12
13 // PRINT "Rectangle's height: "
14     printf("Rectangle\'s Height: ");
15 // SCAN height
16     scanf("%f", &height);
17
18 // COMPUTE area AS width * height
19     float area = width * height;
20 // COMPUTE perimeter AS width*2 + height*2
21     float perimeter = width*2 + height*2;
22
23 // PRINT "Area is "
24     printf("Area is %f\n", area);
25 // PRINT "Perimeter is "
26     printf("Perimeter is %f\n", perimeter);
27
28     return 0;
29 }
```

**Output comparison****Given output****Error output log:**

```

warning C4255: 'main_': no function prototype given: converting '()' to '(void)'
: error C2146: syntax error: missing ';' before identifier 'printf__DO_NOT_EDIT_THIS_FILE'
5): error MSB6006: "CL.exe" exited with code 2.
```

**My output**

```

Rectangle's Width: 91.449997
Rectangle's Height: 39.570000
Area is 3618.676270
Perimeter is 262.039978
```

100 / 100 Tests Passed

**Bugs**

- Line 3: missing **void** in main
- Line 9: missing semicolon ;
- Line 11: excess indentation
- Uncomment lines 12-14 and 18-20 and fix their indentation
- Lines 17 & 18: swap the variables printed (**area** printed in line 17 and **perimeter** printed in line 18)
- Line 15: (optional) perimeter calculation can be simplified to (**width\*2**) + (**height\*2**)

## LAB 4: 4x03a – Enjoyment Question

23/03/2022 – Wednesday , Week 4

### Program aim

Write a program that asks the user if they are enjoying the COMP500/ENSE501 labs.

### Pseudocode (plan)

```
PRINT "Are you enjoying the COMP500/ENSE501 labs (y/n) ? "
READ answer
IF answer is 'y', PRINT "Great! Keep going this is only week 4!"
ELSE IF answer is 'n', PRINT "Oh no! Don't worry only eight short weeks to go!"
ELSE answer, PRINT "Your response makes no sense!"
```

### Implementation

#### Source Code (Input)

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      char answer = '\0';
6
7      printf("Are you enjoying the COMP500/ENSE501 labs (y/n)? ");
8      scanf(" %c", &answer);
9
10     if (answer == 'y')
11     {
12         printf("\nGreat! Keep going this is only week 4!");
13     }
14     else if (answer == 'n')
15     {
16         printf("\nOh no! Don't worry only eight short weeks to go!");
17     }
18     else
19     {
20         printf("\nYour response makes no sense!");
21     }
22
23     return 0;
24 }
```

#### Output/s

Are you enjoying the COMP500/ENSE501 labs (y/n)? !  
Your response makes no sense!

Are you enjoying the COMP500/ENSE501 labs (y/n)? y  
Great! Keep going this is only week 4!

Are you enjoying the COMP500/ENSE501 labs (y/n)? n  
Oh no! Don't worry only eight short weeks to go!

94 / 94 Tests Passed

### Lessons learned

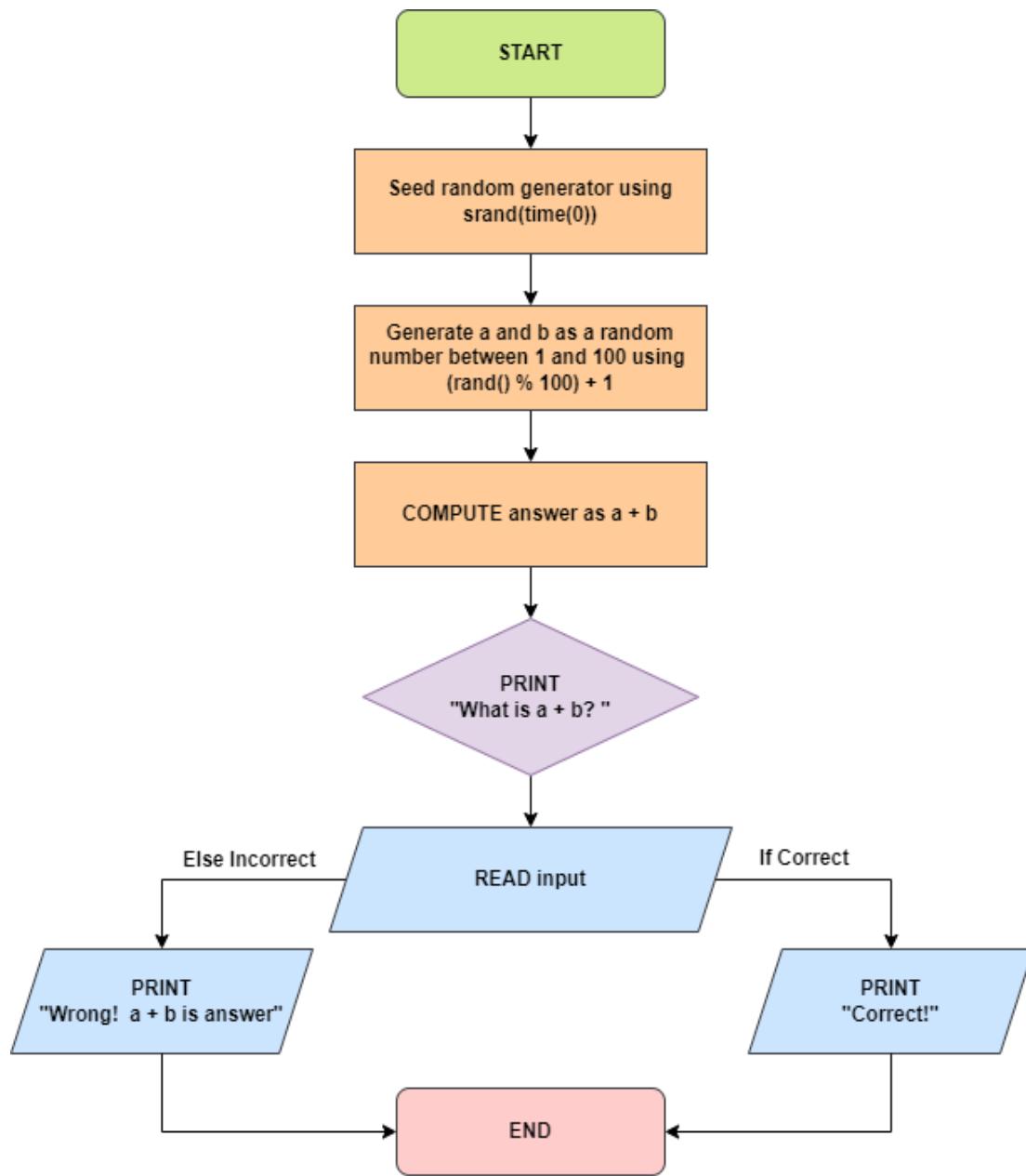
I learned how to use **if else** statements in c programming.

**LAB 4: 4x04a – Math Quiz**  
23/03/2022 – Wednesday , Week 4

**Program aim**

Generate two random whole numbers between 1 and 100 that the user needs to solve the sum of. The program needs to check if the user's answer is correct/incorrect to either congratulate the user for the correct answer or provide the correct answer if the user got it wrong.

**Flowchart plan (Created with draw.io)**



**LAB 4: 4x04a – Math Quiz (Continued)**

23/03/2022 – Wednesday , Week 4

Implementation	
Source Code (Input)	Output/s
<pre> 2  #include &lt;stdlib.h&gt; 3  #include &lt;time.h&gt; 4 5  int main(void) 6  { 7      // seed random generator 8      srand(time(0)); 9 10     // generate a and b as a number between 1 and 100 11     int a = (rand() % 100) + 1; 12     int b = (rand() % 100) + 1; 13     int answer = a + b; 14 15     int input; 16 17     printf("What is %d + %d? ", a, b); 18     scanf("%d", &amp;input); 19 20     if (input == answer) 21     { 22         printf("\nCorrect!"); 23     } 24     else 25     { 26         printf("\nWrong! %d + %d is %d", a, b, answer); 27     } </pre>	<p>What is 27 + 50? 60</p> <p>Wrong! 27 + 50 is 77</p> <p>What is 80 + 86? 195</p> <p>Wrong! 80 + 86 is 166</p> <p>What is 33 + 99? 132</p> <p>Correct!</p>  <p>100 / 100 Tests Passed</p>
<b>Lessons learned</b>	
I learnt how to implement flowcharts for the first time. Flowcharts are especially helpful for showing the different results of if statements.	

**LAB 4: 4x05a – Discount Percentage**

23/03/2022 – Wednesday , Week 4

**Program aim**

Ask the user to input a whole number representing the total purchase price at a retail store in dollars.

Based upon the store's discount rewards program, the program must calculate and output the appropriate discount, and then present the resulting final payable total.

**Plan**

Purchase Amount	Discount Percentage
$x < 2500$	0.00%
$2500 \leq x < 6500$	5.00%
$6500 \leq x \leq 10000$	10.00%
$x > 10000$	12.50%

**Testing/errors/debugging**

```

11    if (purchase >= 2500)
12    {
13        if (purchase < 6500)
14        {
15            discount_percentage = 5.0f;
16        }
17        if (purchase >= 6500 && purchase <= 10000)
18        {
19            discount_percentage = 10.0f;
20        }
21        if (purchase > 10000)
22        {
23            discount_percentage = 12.5f;
24        }
25    }

```

Obviously, the source code to the left would not provide the correct output.

I wan't thinking when I coded these nested ifs!

The purchase discount is restricted to  $2500 \leq \text{purchase} < 6500$  in line 13- so the nested ifs beyond line 17 wouldn't work as they require purchase to be greater than 6500

**Implementation****Source Code (Input)**

```

3 int main(void)
4 {
5     float purchase = 0.0f;
6     float discount_percentage = 0.0f;
7
8     printf("Input the total purchase price: ");
9     scanf("%f", &purchase);
10
11    if (purchase >= 2500 && purchase < 6500)
12    {
13        discount_percentage = 0.05f;
14    }
15    else if (purchase >= 6500 && purchase <= 10000)
16    {
17        discount_percentage = 0.1f;
18    }
19    else if (purchase > 10000)
20    {
21        discount_percentage = 0.125f;
22    }
23
24    float discount = 0.0f;
25    discount = purchase * discount_percentage;
26
27    float total = 0.0f;
28    total = purchase - discount;
29
30    printf("Discount is: %.2f \n", discount);
31    printf("Payable Total is: %.2f", total);

```

**Output/s**

0.0% discount:

```

Input the total purchase price: 259.279999
Discount is: 0.00
Payable Total is: 259.28

```

5.0% discount:

```

Input the total purchase price: 4748.990234
Discount is: 237.45
Payable Total is: 4511.54

```

10.0% discount:

```

Input the total purchase price: 6803.839844
Discount is: 680.38
Payable Total is: 6123.46

```

12.5% discount:

```

Input the total purchase price: 16289.929688
Discount is: 2036.24
Payable Total is: 14253.69

```



100 / 100 Tests Passed

**Lessons learned**

Make use of boolean operators (!, &&, ||) instead of many nested ifs!

**Note from lab (unrelated to exercise, but a reminder of formatting placeholders)**

Formatting placeholder	Use
%x	Prints base-16 value of the ASCII character
%d	Prints decimal (base-10) value of the ASCII character
%c	Prints ASCII character

## LAB 4: 4x06a – Study Hours

24/03/2022 – Thursday (lab), Week 4

**Program aim**

Ask user to state how many hours they spent studying in preparation for the Practical Test. User must respond with a whole number to indicate the number of hours.

**Pseudocode (plan)**

```

START
PRINT How many hours did you spend studying for the test?
IF study_hours is between 26 and 39 hours inclusive THEN
    PRINT Good, but was it enough...
ELSE IF study_hours is between 0 and 25 hours inclusive THEN
    PRINT study_hours hours is probably not enough!
ELSE IF study_hours is 40+ hours THEN
    PRINT Excellent, you should be well prepared for the test!
ELSE IF study_hours is negative THEN
    PRINT Invalid, you cannot have studied a negative number of hours!
END

```

**Implementation**

Source Code (Input)	Output/s
<pre> 1 #include &lt;stdio.h&gt; 2 3 int main(void) 4 { 5     int study_hours = 0; 6 7     printf("How many hours did you spend studying for the test? "); 8     scanf("%d", &amp;study_hours); 9 10    if (study_hours &gt;= 26 &amp;&amp; study_hours &lt;= 39) 11    { 12        printf("\nGood, but was it enough..."); 13    } 14    else if (study_hours &gt;= 0 &amp;&amp; study_hours &lt;= 25) 15    { 16        printf("\n%d hours is probably not enough!", study_hours); 17    } 18    else if (study_hours &gt;= 40) 19    { 20        printf("\nExcellent, you should be well prepared for the test!"); 21    } 22    else if (study_hours &lt; 0) 23    { 24        printf("\nInvalid, you cannot have studied a negative number of hours!"); 25    } 26 27 28 }</pre>	<pre> How many hours did you spend studying for the test? -50 Invalid, you cannot have studied a negative number of hours!  How many hours did you spend studying for the test? 5 5 hours is probably not enough!  How many hours did you spend studying for the test? 27 Good, but was it enough...  How many hours did you spend studying for the test? 40 Excellent, you should be well prepared for the test!  101 / 101 Tests Passed </pre>

**Lessons learned**

I am getting better at utilising the **&&** boolean operator for if statements!

## **LAB 4: 4x07a – Simple Guessing Game**

**24/03/2022 – Thursday (lab), Week 4**

### **Program aim**

Generate a secret number between 1 and 10. Give the user three chances to guess the secret number.

### **Pseudocode (plan)**

START

Use RANDOM to GENERATE secret\_number as integer between 1 and 10

PRINT I'm thinking of a number between 1 and 10...

PRINT What is your first guess?

IF guess is equal to secret\_number THEN

PRINT CORRECT!

PRINT Well done, guess was my number! You took one guess!

ELSE guess is not equal to secret\_number THEN

PRINT Incorrect!

PRINT What is your second guess?

IF guess is equal to secret\_number THEN

PRINT Correct!

PRINT Well done, guess was my number! You took two guesses!

ELSE guess is not equal to secret\_number THEN

PRINT Incorrect!

PRINT What is your third guess?

IF guess is equal to secret\_number THEN

PRINT Correct!

PRINT Well done, guess was my number! You took three guesses!

ELSE guess is not equal to secret\_number THEN

PRINT Incorrect!

PRINT You could not guess my number! It was secret\_number.

END

**LAB 4: 4x07a – Simple Guessing Game (Continued)**

24/03/2022 – Thursday (lab), Week 4

Implementation	
Source Code (Input)	Output/s
<pre> 1 #include &lt;stdio.h&gt; 2 #include &lt;stdlib.h&gt; 3 #include &lt;time.h&gt; 4 5 int main(void) 6 { 7     srand(time(0)); 8 9     int guess = 0; 10    int secret_number = (rand() % 10) + 1; 11 12    printf("I'm thinking of a number between 1 and 10...\n"); 13    printf("You have three chances to guess the number...\n\n"); 14 15    printf("What is your first guess? "); 16    scanf("%d", &amp;guess); 17 18    if (guess == secret_number) 19    { 20        printf("Correct! \n\n"); 21        printf("Well done, %d was my number! You took one guess!", guess); 22    } 23    else 24    { 25        printf("Incorrect! \n\n"); 26        printf("What is your second guess? "); 27        scanf("%d", &amp;guess); 28 29        if (guess == secret_number) 30        { 31            printf("Correct! \n\n"); 32            printf("Well done, %d was my number! You took two guesses!", guess); 33        } 34        else 35        { 36            printf("Incorrect! \n\n"); 37            printf("What is your third guess? "); 38            scanf("%d", &amp;guess); 39 40            if (guess == secret_number) 41            { 42                printf("Correct! \n\n"); 43                printf("Well done, %d was my number! You took three guesses!", guess); 44            } 45            else 46            { 47                printf("Incorrect! \n\n"); 48                printf("You could not guess my number! It was %d.", secret_number); 49            } 50        } 51    } 52 53    return 0; 54 }</pre>	<p>One guess:</p> <p>I'm thinking of a number between 1 and 10... You have three chances to guess the number...</p> <p>What is your first guess? 2 Correct!</p> <p>Well done, 2 was my number! You took one guess!</p> <p>Two guesses:</p> <p>I'm thinking of a number between 1 and 10... You have three chances to guess the number...</p> <p>What is your first guess? 4 Incorrect!</p> <p>What is your second guess? 2 Correct!</p> <p>Well done, 2 was my number! You took two guesses!</p> <p>Three guesses:</p> <p>I'm thinking of a number between 1 and 10... You have three chances to guess the number...</p> <p>What is your first guess? 5 Incorrect!</p> <p>What is your second guess? 5 Incorrect!</p> <p>What is your third guess? 9 Correct!</p> <p>Well done, 9 was my number! You took three guesses!</p> <p>Three guesses and incorrect:</p> <p>I'm thinking of a number between 1 and 10... You have three chances to guess the number...</p> <p>What is your first guess? -3 Incorrect!</p> <p>What is your second guess? 2 Incorrect!</p> <p>What is your third guess? 0 Incorrect!</p> <p>You could not guess my number! It was 10.</p>

**Lessons learned**

I learnt how to implement nested ifs properly! I learned that proper planning is important for program designs that require many different outcomes. Perhaps I could have used a flowchart instead of pseudocode?

**Week 4 lecture 012 Notes- Repetition, while, Input Validation, do while, break, continue**

Notes Retrieved from Steffan Hooper, W04 Lec 012, 2022

**25/03/2022 - Friday****while:** used for creating while loops (like in python). Pseudocode example:

WHILE LOOP EXAMPLE	
Pseudocode	Source code
<pre> DECLARE count_down SET count_down TO 10  WHILE count_down &gt; 0     PRINT count_down     PRINT "...\\n"     SUBTRACT 1 FROM count_down ENDWHILE  PRINT "The timer has expired!"</pre>	<pre> ... int count_down = 10;  while (count_down &gt; 0) {     printf("%d...\\n", count_down);     --count_down; } printf("Timer has expired!\\n"); ...</pre>

**Validation of User Input (a form of error check):**

USER INPUT VALIDATION EXAMPLE	
Source code	
<pre> ... while (0 == scan_result) {     scan_result = scanf("%d", &amp;number);     if (0 == scan_result)     {         printf("Please enter a number! Try again: ");         scanf("%*[^\n]");     } } ... scanf("%*[^\n]"); → This clears the input buffer. Good to use when the user needs to re-input a value</pre>	

**do...while** is another way to make loops: it will run the do part at least once and it will repeat statements if the condition evaluates to be true

USER INPUT VALIDATION EXAMPLE	
Source code	
<pre> ... do → The do block will be executed once {     printf("%d...\\n", counter);     ++counter; } while (counter &lt; 5); → if while condition is true, then the do block will execute again ...</pre>	

## Week 4 lecture 012 Notes- Repetition, while, Input Validation, do while, break, continue (continued)

Notes Retrieved from Steffan Hooper, W04 Lec 012, 2022

**25/03/2022 – Friday**

**break** Keyword: will stop loop from iterating. (ex. Evaluate If statement, if it is true, break the loop)  
**continue** Keyword: cause loop to stop at current iteration and continue to next iteration

### **Coding standard:**

- **do while** loop is good for when one iteration needs to be executed
- **while** loop is good for iterating over some instructions

**LAB 4: 4x08a – Human Resources Tool**

25/03/2022 – Friday , Week 4

**Program aim**

Design, implement and test a C program which helps a recruiter decide whether to interview a candidate for an IT job. The recruiter only considers two criteria. Firstly, the candidate's number of years of programming experience, and secondly, whether or not the candidate knows how to program in C. If at least one of these criteria are met, they can be interviewed.

**Pseudocode (plan)**

```

START
PRINT How many years' experience?
READ to int variable years_experience
PRINT Does the candidate know C (y/n)?
READ to char variable know_c
IF years_experience is equal or more than 3 OR know_c is equal to y THEN
    PRINT This candidate must be interviewed.
ELSE
    PRINT Do not interview this candidate.
END

```

**Implementation**

Source Code (Input)	Output/s
<pre> 1 #include &lt;stdio.h&gt; 2 3 int main(void) 4 { 5     int years_experience = 0; 6     char know_c = '\0'; 7 8     printf("How many years' experience? "); 9     scanf("%d", &amp;years_experience); 10 11    printf("Does the candidate know C (y/n)? "); 12    scanf(" %c", &amp;know_c); 13 14    if (years_experience &gt;= 3    know_c == 'y') 15    { 16        printf("\nThis candidate must be interviewed."); 17    } 18    else 19        printf("\nDo not interview this candidate."); 20 21    return 0; 22 }</pre>	<pre> How many years' experience? 0 Does the candidate know C (y/n)? n  Do not interview this candidate.  How many years' experience? 8 Does the candidate know C (y/n)? y  This candidate must be interviewed.  How many years' experience? 8 Does the candidate know C (y/n)? n  This candidate must be interviewed.  </pre>
	
	10 / 10 Tests Passed

**Lessons learned**

I learned how to implement the || OR boolean operator for the first time!

**LAB 4: 4x09a – Alphabetic**

25/03/2022 – Friday , Week 4

**Program aim**

Ask the user to input a character and check if the user's input is an alphabetic letter or not, and then provide an output either to state the input was an alphabetic letter, or not.

**Pseudocode (plan)**

```

START
PRINT Input a character:
READ to char variable user_input

IF user_input is between a-z inclusive OR between A-Z inclusive THEN
    PRINT The input was alphabetic
ELSE
    PRINT The input was not alphabetic
END

```

**Implementation**

Source Code (Input)	Output/s
	Not alphabetic <b>Input a character: %</b> <b>The input was not alphabetic.</b>
	Is alphabetic (lowercase) <b>Input a character: C</b> <b>The input was alphabetic.</b>
	Is alphabetic (uppercase) <b>Input a character: M</b> <b>The input was alphabetic.</b>
<pre> 1 #include &lt;stdio.h&gt; 2 3 int main(void) 4 { 5     char user_input = '\0'; 6 7     printf("Input a character: "); 8     scanf("%c", &amp;user_input); 9 10    if (user_input &gt;= 'a' &amp;&amp; user_input &lt;= 'z'    user_input &gt;= 'A' &amp;&amp; user_input &lt;= 'Z') 11    { 12        printf("\nThe input was alphabetic."); 13    } 14    else 15        printf("\nThe input was not alphabetic."); 16 17    return 0; 18 }</pre>	 94 / 94 Tests Passed

**Lessons learned**

I learned how to implement multiple boolean operators (`&&` and `||`) in a if statement!  
I also learned that chars can be compared in a range like integers/floats.

**LAB 4: 4x10a – Text Input Birthday Season**

25/03/2022 – Friday , Week 4

**Program aim**

Using sequence, selection and `strcmp`, ask user to enter their birth month as text, and then output the name of the month and the New Zealand season in they were born.

Season	Month Range
Summer	December, January, February
Spring	September, October, November
Winter	June, July, August
Autumn	March, April, May

**Pseudocode (plan)**

```

START
PRINT What month were you born in?
READ to c-string array birthday_month
IF birthday_month is December OR January OR February THEN
    PRINT birthday_month in New Zealand is in Summer.
ELSE IF birthday_month is September OR October OR November THEN
    PRINT birthday_month in New Zealand is in Autumn.
ELSE IF birthday_month is June OR July OR August THEN
    PRINT birthday_month in New Zealand is in Winter.
ELSE IF birthday_month is March OR April OR May THEN
    PRINT birthday_month in New Zealand is in Spring.
ELSE birthday_month is not a month THEN
    PRINT birthday_month is not a month!
END

```

**Implementation****Source Code (lines 1-34)**

```

1  #include <stdio.h>
2  [#include <string.h>]
3
4  int main(void)
5  {
6      char birthday_month[10] = "";
7
8      printf("What month were you born in? ");
9      scanf("%s", &birthday_month);
10
11     // Summer is December, January, February
12     if (strcmp(birthday_month, "December") == 0 || strcmp(birthday_month, "January") == 0 || strcmp(birthday_month, "February") == 0)
13     {
14         printf("\n%s in New Zealand is in Summer.", birthday_month);
15     }
16     // Spring is September, October, November
17     else if (strcmp(birthday_month, "September") == 0 || strcmp(birthday_month, "October") == 0 || strcmp(birthday_month, "November") == 0)
18     {
19         printf("\n%s in New Zealand is in Spring.", birthday_month);
20     }
21     // Winter is June, July, August
22     else if (strcmp(birthday_month, "June") == 0 || strcmp(birthday_month, "July") == 0 || strcmp(birthday_month, "August") == 0)
23     {
24         printf("\n%s in New Zealand is in Winter.", birthday_month);
25     }
26     // Autumn is March, April, May
27     else if (strcmp(birthday_month, "March") == 0 || strcmp(birthday_month, "April") == 0 || strcmp(birthday_month, "May") == 0)
28     {
29         printf("\n%s in New Zealand is in Autumn.", birthday_month);
30     }
31     else
32     {
33         printf("\n%s is not a month!", birthday_month);
34     }

```

## LAB 4: 4x10a – Text Input Birthday Season (Continued)

25/03/2022 – Friday , Week 4

Output/s	
What month were you born in? January January in New Zealand is in Summer.	Summer NZ birthday
What month were you born in? March March in New Zealand is in Autumn.	Autumn NZ birthday
What month were you born in? June June in New Zealand is in Winter.	Winter NZ Birthday
What month were you born in? September September in New Zealand is in Spring.	Spring NZ Birthday
What month were you born in? Banana Banana is not a month!	Not a month



**15 / 15 Tests Passed**

### **Lessons learned**

I learned how to implement `strcmp` – which is used to compare two strings/chars, and if they are the same, it outputs 0.

# WEEK 5

## Week 5 lecture 013 Notes- for, Arrays and for Loops

Notes Retrieved from Steffan Hooper, W05 Lec 013, 2022

28/03/2022 – Monday

for loops control whether a sequence of instructions is repeated, example of syntax:

**This for loop will be executed 10 times**

```
for (int count = 0; count < 10; ++count)
{
    // Body...
}

int count variable declared and assigned to 0, condition for loop body to execute again, update to count
(increment)
```

counters don't have to start from 0. counters can be incremented up, down, and by values > 0, ex:  
counter -= 10

Any variable type can be used to control for loop: chars can be used, for example:

```
for (char k = 'A'; k <= 'Z'; ++k)
{
    printf("%c [%d]\n", k, k); → loop going from A-Z will print the alphabet
}
printf("\n");
printf("A to Z has been printed!\n");
```

**Arrays can be used in for loops**

```
int data[] = { 1, 2, 4, 8, 16, 32, 64, 128, 256 };
for (int index = 0; index < 9; ++index) → elements of array are iterated
{
    printf("data[%d] is %d\n", index, data[index]);
}
```

Prints contents of the index, prints the index number

Example of iterating a mathematical operation (division)

This will take each element in the array, divide it by two, and update the element

```
for (int index = 0; index < 6; ++index)
{
    numbers[index] = numbers[index] / 2.0f;
```

Index of array, going through each element, dividing by two, do 6 times, array saved into

REMEMBER: char arrays need to be terminated using null char '\0'

**LAB 5: 5x01a – Number to Word switch**

31/03/2022 – Thursday (lab), Week 5

**Program aim**

Using a switch, ask user to input a whole number between 0 and 9. Output the number as a word in text. Detect when the user's input is not within the required range.

**Pseudocode (Plan)**

```

PRINT Input a number between 0 and 9
READ number
SWITCH for the number variable
CASE for input of '0'
PRINT "zero"
BREAK
Repeat CASES, PRINTS and BREAKS for values 0-9
Using DEFAULT, PRINT "Invalid input!"
```

**Implementation**

Source Code (lines 5-44)	Output/s
5      int number; 6 7      printf("Input a number between 0 and 9: "); 8      scanf("%d", &number); 9 10     switch (number) 11     { 12       case 0: 13         printf("\nzero"); 14         break; 15       case 1: 16         printf("\none"); 17         break; 18       case 2: 19         printf("\ntwo"); 20         break; 21       case 3: 22         printf("\nthree"); 23         break; 24       case 4: 25         printf("\nfour"); 26         break; 27       case 5: 28         printf("\nfive"); 29         break; 30       case 6: 31         printf("\nsix"); 32         break; 33       case 7: 34         printf("\nseven"); 35         break; 36       case 8: 37         printf("\neight"); 38         break; 39       case 9: 40         printf("\nnine"); 41         break; 42       default: 43         printf("\nInvalid input!"); 44 }	<p>Input a number between 0 and 9: 0</p> <p>zero</p> <p>(checking negative number- out of range)</p> <p>Input a number between 0 and 9: -5</p> <p>Invalid input!</p> <p>Input a number between 0 and 9: 9</p> <p>nine</p> <p>Input a number between 0 and 9: 19</p> <p>Invalid input!</p>  <p>20 / 20 Tests Passed</p>

**Lessons learned**

I learned how to implement **switch**, **cases**, and **default** for the first time!

**Week 5 lecture 014 Notes- for Loop bugs, Nested Selection**

Notes Retrieved from Steffan Hooper, W05 Lec 014, 2022

**30/03/2022 - Wednesday****while vs for loops:**

- **for** loop is good when I know how many iterations I need (the starting point and the ending point are known)
- The **while** loop is good when I don't know how many iterations I need (the program needs to iterate over some instructions, and during an iteration it will become known when to stop)

REMEMBER FOR LOOP FORMAT: **for (int n = 0; n < 10; n++)**When I accidentally create an infinite for loop (**for (;;);**), press ctrl-c to halt program.**PRINTING ARRAY ELEMENTS FORWARDS/BACKWARDS**

Source code (print forwards)	Source code (print backwards)
<pre>int data[] = { 1, 2, 4, 8, 16, 32, 64, 128, 256 }; for (int index = 0; index &lt; 9; ++index) {     printf("data[%d] is %d\n", index, data[index]); }</pre>	<pre>int data[] = { 1, 2, 4, 8, 16, 32, 64, 128, 256 }; for (int index = 8; index &gt;= 0; --index) {     printf("data[%d] is %d\n", index, data[index]); }</pre>
Index starts from 0, iterates 8 times by + 1 until it reaches last [8] element	Index starts from 8, iterates 8 times by - 1 until it reaches 1st [0] element

Arrays can be copied to another array by iterating over the elements of one list and equating it to another list of the same size or over.

**HOW TO SEARCH FOR AN ELEMENT IN AN ARRAY:**

```
int unordered_data[] = { 31, 42, 24, 16, 30, 45, 10, 20 };
int found_at = -1;
for (int i = 0; i < 8; ++i)
{
    if (unordered_data[i] == 30) ← iterates until an element is equal to 30
    {
        found_at = i;
    }
}
printf("30 is at index: %d\n", found_at); ← index would be [4]
```

## Week 5 lecture 014 Notes- for Loop bugs, Nested Selection (Continued)

Notes Retrieved from Steffan Hooper, W05 Lec 014, 2022

30/03/2022 - Wednesday

### USING WHILE LOOP TO FIND CHAR LENGTH OF C-STRING

```
char message[] = "Hello World";
int count = 0;
while (message[count] != '\0') ← while loop is used because # of iterations are unknown
{
    ++count; ← count iterates +1 until all the chars in message are iterated through
}
printf("The string is %d characters long.\n", count);
```

### USING SWITCH-CASE TO FIND VOWELS AND CONSONANTS IN ALPHABET

```
char block_of_text[] = "abcdefghijklmnopqrstuvwxyz";
int current = 0;
while (block_of_text[current] != '\0')
{
    printf("%c is a ", block_of_text[current]);
    switch (block_of_text[current])
    {
        case 'a': // Fall through...
        case 'e': // Fall through...
        case 'i': // Fall through...
        case 'o': // Fall through...
        case 'u':
            printf("vowel.\n"); ← if block_of_text[current] = a, e, i, o, u- it is detected as vowel
            break;
        default: ← like ELSE
            printf("consonant.\n");
    }
    ++current;
}
```

Notes from the lab:

**31/03/2022– Thursday (lab), Week 4**

According to Steffan:

- “You can avoid creating difficult bugs/problems by choosing the correct literal for the type of data you are operating on”
- “Getting in the habit of using '\0' when working with chars helps create robust source code.”
- “Picking the right type for the job can avoid hidden conversions”

\0 is the ASCII character for null character zero (it's basically zero for chars)

0.0 for double

0.0f for float

**LAB 5: 5x02a – Conditional Max of Two**

31/03/2022 – Thursday (lab), Week 5

**Program aim**

Use conditional operator to find largest of 2 user inputted integer numbers.

**Pseudocode (Plan)**

```

PRINT First number?
READ first_number
PRINT Second number?
READ second_number

COMPUTE largest_number as CONDITIONAL OPERATOR:
first_number > second_number ? first_number : second_number
What ? True : False

PRINT "The larger of first number and second number is largest number"

```

**Testing/errors/debugging**

Putting the conditional operator in the print line (line 14) also works

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     int first_number = 0;
6     int second_number = 0;
7
8     printf("First number? ");
9     scanf("%d", &first_number);
10
11    printf("Second number? ");
12    scanf("%d", &second_number);
13
14    printf("\n\nThe larger of %d and %d is %d.", first_number, second_number, first_number > second_number ? first_number : second_number);
15
16    return 0;
17 }

```

**Implementation****Source Code****Output/s**

<pre> 1 #include &lt;stdio.h&gt; 2 3 int main(void) 4 { 5     int first_number = 0; 6     int second_number = 0; 7 8     printf("First number? "); 9     scanf("%d", &amp;first_number); 10 11    printf("Second number? "); 12    scanf("%d", &amp;second_number); 13 14    int largest_number; 15 16    largest_number = first_number &gt; second_number ? first_number : second_number; 17 18    printf("\n\nThe larger of %d and %d is %d.", first_number, second_number, largest_number); 19 20    return 0; 21 } </pre>	<pre> First number? 761 Second number? 846  The larger of 761 and 846 is 846.  First number? 669 Second number? 762  The larger of 669 and 762 is 762.  Testing negative input First number? -100 Second number? -50  The larger of -100 and -50 is -50. </pre>
---	---

**Lessons learned**

I learned how to implement the conditional operator for the first time!

## LAB 5: 5x03a – Count Up Loop

31/03/2022 – Thursday (lab), Week 5

### Program aim

Write a program using a loop that will output a count from 40 to 400 in steps of ten.

### Pseudocode (Plan)

```
FOR count = 40, add 10 to count UNTIL count is less than or equal to 400:  
    PRINT count
```

### Testing/errors/debugging

At one point, I forgot to make `count <= 400`, and instead I made it `count < 400`. This resulted in the for loop output stopping at count = 390.

### Implementation

#### Source Code

```
1 #include <stdio.h>  
2  
3 int main(void)  
4 {  
5     // count = 40, add 10 until count = 400  
6     for (int count = 40; count <= 400; count += 10)  
7     {  
8         printf("%d\n", count);  
9     }  
10    return 0;  
11 }  
12 }
```

#### Output/s

```
Microsoft Visual Studio Debug Console  
40  
50  
60  
70  
80  
90  
100  
110  
120  
130  
140  
150  
160  
170  
180  
190  
200  
210  
220  
230  
240  
250  
260  
270  
280  
290  
300  
310  
320  
330  
340  
350  
360  
370  
380  
390  
400
```

### Lessons learned

I learnt how to implement a for loop for the first time! It was strange not having to declare the `count` variable before the for loop line as it was declared within the for loop line.

## LAB 5: 5x04a – Simple Loop

31/03/2022 – Thursday (lab), Week 5

### Program aim

Write a program using a loop that will output the numbers 1 to n in square brackets, where n is based upon user input. Each number must be separated by a single space.

### Pseudocode (Plan)

```

START
    PRINT n?
    READ integer n
    FOR count = n, add 1 to count UNTIL count is less than or equal to n:
        PRINT [count]
END

```

### Implementation

Source Code	Output/s
<pre> 1 #include &lt;stdio.h&gt; 2 3 int main(void) 4 { 5     int n = 0; 6     printf("n? "); 7     scanf("%d", &amp;n); 8     printf("\n"); 9 10    // count = input, add 1 until count = input 11    for (int count = 1; count &lt;= n; count++) 12    { 13        printf("[%d] ", count); 14    } 15 16    return 0; 17 }</pre>	<p>n? 1 [1]</p> <p>n? 3 [1] [2] [3]</p> <p>n? 13 [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13]</p> <p>n? 21 [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21]</p> <p style="text-align: right;">100 / 100 Tests Passed</p>

### Lessons learned

I learned how to increment (ex: `count++`) in a for loop and use user input in a for loop for the first time!

## **Week 5 lecture 015 Notes- How to Program**

Notes Retrieved from Steffan Hooper, W05 Lec 015, 2022

**01/04/2022 – Friday**

### **22 key points to remember when attempting to program anything**

1. Understand the problem
2. Note the parts you know how to do
3. Note the parts you don't know how to do
4. Look for a shortcut
5. BUT only take it if you know that you can do it the hard way
6. Assemble your 'toolkit' (What do I know that can help me solve this problem?)
7. Have a plan for what the program is going to do
8. Have a plan for how you are to write the program. Types of planning:
  - Start from the beginning (top-down) if you know how to start
  - Start from what you know (bottom-up) if you can't do step 1
  - Take down the biggest target- unsure of any steps? Break down the biggest step.

Don't write the program in one go, and don't multitask on different parts of the program

Follow the programming cycle:

EDIT > COMPILE > RUN > TEST

9. Program with purpose (think of what you are coding and why)
10. Use comments to remind you of your plan
11. Read your own code
12. Compile Frequently
13. Test Early and Test Often
14. Test with a Purpose (What part did I just add, change, or delete? Expectations?)
15. Debugging (use the Visual Studio Debugger when you have messy confusing code)  
Comment out /\* \*/ code you are not sure about
16. Accept that it may take some time- have patience
17. Guess once, then look it up
18. FOCUS
19. Be aware of your own mood
20. You are programming an unthinking, unfeeling machine- it only does what you tell it to do!
21. Never make assumptions- read the question carefully
22. Celebrate your success! Reflect, note progress and lessons learnt, learn to enjoy  
programming 😊

**LAB 5: 5x05a – Multiplication Table**

02/03/2022 – Saturday, Week 5

**Program aim**

Using a loop, ask user to input a whole number. Then output the multiplication table, from 0 to 14, for the whole number input.

**Pseudocode (Plan)**

```

PRINT Enter a whole number:
READ as integer variable number_input
FOR count = 0, add 1 to count UNTIL count is less than or equal to 14:
    PRINT The 7 Times Table:
    PRINT -----
    COMPUTE multiplication as count * number_input
    PRINT count x number_input = multiplication

```

**Implementation****Source Code**

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     int number_input = 0;
6     int multiplication = 0;
7
8     printf("Enter a whole number: ");
9     scanf("%d", &number_input);
10
11    printf("\nThe %d Times Table: \n", number_input);
12    printf("----- \n\n");
13
14    // count = 0, add 1 until count = 14
15    for (int count = 0; count <= 14; count++)
16    {
17        multiplication = count * number_input;
18        printf("%d x %d = %d \n", count, number_input, multiplication);
19    }
20
21    return 0;
22 }

```

**Output/s**

Enter a whole number: 3	Enter a whole number: 44
The 3 Times Table:	The 44 Times Table:
-----	-----
0 x 1 = 0	0 x 44 = 0
1 x 3 = 3	1 x 44 = 44
2 x 3 = 6	2 x 44 = 88
3 x 3 = 9	3 x 44 = 132
4 x 3 = 12	4 x 44 = 176
5 x 3 = 15	5 x 44 = 220
6 x 3 = 18	6 x 44 = 264
7 x 3 = 21	7 x 44 = 308
8 x 3 = 24	8 x 44 = 352
9 x 3 = 27	9 x 44 = 396
10 x 3 = 30	10 x 44 = 440
11 x 3 = 33	11 x 44 = 484
12 x 3 = 36	12 x 44 = 528
13 x 3 = 39	13 x 44 = 572
14 x 3 = 42	14 x 44 = 616

100 / 100 Tests Passed

**Lessons learned**

I learned how to implement math operations within a for loop, for the first time!

## LAB 5: 5x06a – Power using a Loop

02/03/2022 – Saturday, Week 5

### Program aim

Using loops, write a program which computes the power of any whole number base, raised to any positive whole number power. DON'T USE `pow()` OR `powf()` FROM `math.h`

### Pseudocode (Plan)

```
RESULT is equal to 1
PRINT Enter the base:
READ base

PRINT Enter the power:
READ power

PRINT base ^ power is the same as...
PRINT "base" ...

FOR loop : count until less than or equal to power
    PRINT ... * base"
    COMPUTE result as result * base

PRINT ... "which is result"
```

### Testing/errors/debugging

EXAMPLE: say `base = 2` and `power = 3`, which is same as  $2^3$  and results in the answer 8.

While coding, I made the calculation in line 21: `base = base * power`, which would multiply:

$2 * 3 = 6$ , then  $6 * 3 = 18$ , then  $18 * 3 = 54$ . This is the **wrong answer!**

Then I made the calculation in line 21: `base = base * base`, which would multiply:

$2 * 2 = 4$ , then  $4 * 4 = 16$ , then  $16 * 16 = 256$ . This is also the **wrong answer!**

Then finally, I actually thought about how exponents worked. I made a new variable called `result` and assigned it to the value 1. I made the calculation go: `result = result * base`, which would multiply:

$1 * 2 = 2$ , then  $2 * 2 = 4$ , then  $4 * 2 = 8$ . This is the **correct answer!**

**LAB 5: 5x06a – Power using a Loop (continued)**

02/03/2022 – Saturday, Week 5

Implementation	
Source Code	Output/s
<pre> 4  { 5      int base = 0; 6      int power = 0; 7      int result = 1; 8 9      printf("Enter the base: "); 10     scanf("%d", &amp;base); 11     printf("Enter the power: "); 12     scanf("%d", &amp;power); 13 14     printf("\n%d ^ %d is the same as...\n", base, power); 15     printf("\n%d", base); 16 17     // count = 1, add 1 until count is less than or equal to power 18     for (int count = 1; count &lt;= power; count++) 19     { 20         printf(" * %d", base); 21         result = result * base; 22     } 23 24     printf(" which is %d", result); 25 26     return 0; 27 }</pre>	<pre> Enter the base: 2 Enter the power: 2 2 ^ 2 is the same as... 2 * 2 * 2 which is 4</pre>
	<pre> Enter the base: 22 Enter the power: 2 22 ^ 2 is the same as... 22 * 22 * 22 which is 484</pre>
	<pre> Enter the base: 9 Enter the power: 6 9 ^ 6 is the same as... 9 * 9 * 9 * 9 * 9 * 9 which is 531441</pre>

**Lessons learned**

STOP AND THINK about how the program is supposed to function before coding it!

I learned how to combine prints outside and within loops, and I learned how to implement exponents using a for loop only and without the use of `pow()` or `powf()` from `math.h`.

## LAB 5: 5x07a – ASCII Alphabet Printer

02/03/2022 – Saturday, Week 5

### Program aim

Print out all the lowercase letters of the alphabet, followed by all upper-case letters of the alphabet. Each letter must be separated by a single space. The lower and uppercase alphabet must be separated by a newline.

### Pseudocode (Plan)

```
START
    DECLARE capital as char
    DECLARE lowercase as char
    FOR lowercase, iterate from a to z, increment lowercase by 1
        PRINT "lowercase "
    FOR capital, iterate from A to Z, increment capital by 1
        PRINT "capital "
END
```

### Implementation

#### Source Code

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     char capital = '\0';
6     char lowercase = '\0';
7
8     // char range between a to z incremented by 1
9     for (lowercase = 'a'; lowercase <= 'z'; ++lowercase)
10    {
11        printf("%c ", lowercase);
12    }
13    printf("\n");
14
15    // char range between A to Z incremented by 1
16    for (capital = 'A'; capital <= 'Z'; ++capital)
17    {
18        printf("%c ", capital);
19    }
20
21    return 0;
22 }
```

#### Output

```
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
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
```

### Lessons learned

I learned how to use **chars** in for loops!

## LAB 5: 5x08a – Fix this Loop

02/03/2022 – Saturday, Week 5

### Program aim

Fix the loop in the given source code

### Purpose of program

Print a count down from 10 to 1 using a loop.

### Implementation (comparison to given source code)

#### Given Source Code

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int count_down = 10;
6
7      while (count_down > 0)
8      {
9          printf("%d... ", count_down);
10     }
11
12     printf("\n\nTimer has expired!\n");
13
14     return 0;
15 }
```

#### My Source Code

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int count_down = 10;
6
7      while (count_down > 0)
8      {
9          printf("%d... ", count_down);
10         --count_down;
11     }
12
13     printf("\n\nTimer has expired!\n");
14
15     return 0;
16 }
```

### Output comparison

#### Given output

Infinitely outputs “10...”!

```
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10... 10...
```

#### My output

```
Microsoft Visual Studio Debug Console
10... 9... 8... 7... 6... 5... 4... 3... 2... 1...
Timer has expired!
```

### Bug

Count\_down holds the value 10, and the while loop iterates while count\_down is over 0, therefore, the loop will go on forever due to the value of count\_down staying 10 forever!

To fix the bug and print the count\_down properly, the count\_down variable must be subtracted by 1 in the iteration loop.

**LAB 5: 5x09a – Input and Display**

02/03/2022 – Saturday, Week 5

**Program aim**

Write a program that can read user input values into an array, and then print out the contents of the array.

**Step by step plan**

1. Read user input values into an array. A maximum of 500 elements can be entered
2. Once the user has decided on the number of elements, allow the user to input each element. Store each input in an array element.
3. Print out the elements of the array.

**Testing/errors/debugging**

```
Enter the required number of elements (Max 500): 3
Now enter the 10 elements of the array...
Set [1] to: 4
Set [2] to: 3
Set [3] to: 5

The elements of the array are:
{ -858993460, 4, 3, 5 }
```

This is the output when **i** (in line 28) is set to 0 instead of 1.

The first array element is outputted incorrectly because nothing is actually entered for element [0] due to count's iteration in the while loop happening before the scan.

**Implementation****Source Code**

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int input_element = 0;
6     int input_array[500];
7
8     printf("Enter the required number of elements (Max 500): ");
9     scanf("%d", &input_element);
10
11    printf("\nNow enter the 10 elements of the array... \n\n");
12
13    int count = 0;
14
15    while (count < input_element)
16    {
17        printf("Set [%d] to: ", ++count); // count iterated
18        scanf("%d", &input_array[count]);
19    }
20
21    printf("\nThe elements of the array are: \n\n");
22
23    printf("{ ");
24    for (int i = 1; i <= input_element; i++)
25    {
26        printf("%d", input_array[i]);
27        if (i < input_element)
28        {
29            printf(", "); // print comma until last element
30        }
31    }
32    printf(" }");
33
34    return 0;
35 }
```

**Output/s**

```
Enter the required number of elements (Max 500): 8
Now enter the 10 elements of the array...
Set [1] to: 1
Set [2] to: 2
Set [3] to: 3
Set [4] to: 4
Set [5] to: 5
Set [6] to: 6
Set [7] to: 7
Set [8] to: 8

The elements of the array are:
{ 1, 2, 3, 4, 5, 6, 7, 8 }
```

```
Enter the required number of elements (Max 500): 3
Now enter the 10 elements of the array...
Set [1] to: 3
Set [2] to: 5
Set [3] to: 7

The elements of the array are:
{ 3, 5, 7 }
```

```
Enter the required number of elements (Max 500): 13
Now enter the 10 elements of the array...
Set [1] to: 3
Set [2] to: 24
Set [3] to: 533
Set [4] to: 65
Set [5] to: 54
Set [6] to: 654
Set [7] to: 2
Set [8] to: 34
Set [9] to: 6
Set [10] to: 53
Set [11] to: 65
Set [12] to: 34
Set [13] to: 12

The elements of the array are:
{ 3, 24, 533, 65, 54, 654, 2, 34, 6, 53, 65, 34, 12 }
```

**Lessons learned**

I learnt how to use 2 different loops in the same program to achieve a single outcome, how to print and input arrays using loops, and iterate a variable within a print statement

## LAB 5: 5x10a – Your String Length

02/03/2022 – Saturday, Week 5

### Program aim

Print the string length of a char array using loops and without using `string.h's strlen`.

### Step by step plan

1. Declare a char array of dimension 128
2. prompt the user to input a word and store it in the char array.
3. After the word has been input, count how many characters are in the word, and print the result.

### Testing/errors/debugging

13    `while (word[c] != "/0")` I tried to make line 15 loop it's count until there are no empty characters left, however, this did not work because the logical operator != (doesn't equal) doesn't work with char arrays...

```
warning C4130: '!=': logical operation on address of string constant
warning C4047: '!=': 'int' differs in levels of indirection from 'char [1]'
```

So, I changed the while loop to iterate over the alphabet... but this is just overcomplicated!

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     char word[128] = "";
6     int c = 0;
7
8     printf("Enter a word: ");
9     scanf("%s", &word);
10
11    // iterate counting letter until an element in the word list isn't an alphabet letter
12    while ((word[c] >= 'a' && word[c] <= 'z') || (word[c] >= 'A' && word[c] <= 'Z'))
13    {
14        c++;
15    }
16
17    printf("The word is %d characters long.", c);
18
19    return 0;
20 }
```

THEN, I realised that the first method wasn't working because I was printing the null character wrong, it is printed: '\0' with backslash and single quotes! So I changed the while loop to: `while word[c] != '\0'` and it ended up working as intended.

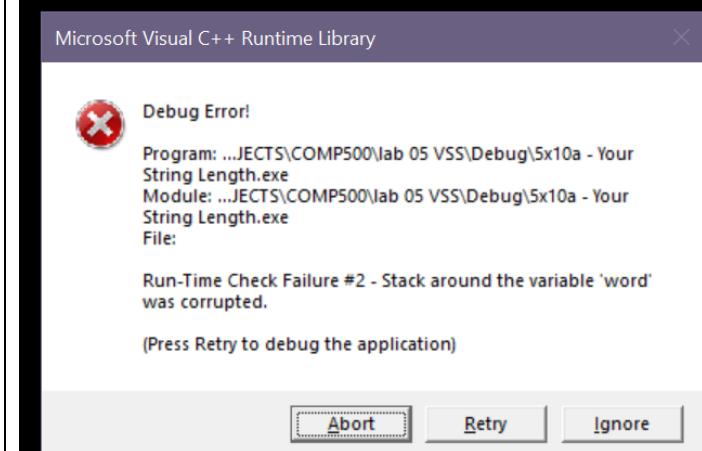
I also realised that multi-letter inputs wouldn't be recognised (only the fist word would be counted), so I fixed this by changing line 9: `scanf("%s", &word)` to `scanf("%[^n]", &word)` which will scan any character until a newline is entered, instead of scanning and stopping after a space.

## **LAB 5: 5x10a – Your String Length (Continued)**

**02/03/2022 – Saturday, Week 5**

Implementation	
Source Code	Output/s
<pre>1 #include &lt;stdio.h&gt; 2 3 int main(void) 4 { 5     char word[128] = "\0"; 6     int c = 0; 7 8     printf("Enter a word: "); 9     scanf(" %[^\n]", &amp;word); 10 11    // iterate counting elements until an element isn't null 12 13    while (word[c] != '\0') 14    { 15        c++; 16    } 17 18    printf("The word is %d characters long.", c); 19 20 21 }</pre>	<pre>Enter a word: Francisca The word is 9 characters long.</pre>
	<pre>Enter a word: mozzarella The word is 10 characters long.</pre>
	<pre>Enter a word: Supercalifragilisticexpialidocious The word is 34 characters long.</pre>
	<pre>Enter a word: Hi The word is 2 characters long.</pre>
	<pre>Enter a word: Four words counting spaces The word is 26 characters long.</pre>

**Output when the input is over 128 characters:**



## Lessons learned

I learnt how to count elements in a c-string array without using `string.h`'s `strlen` by implementing the logic for the problem without using the pre-made functionality.

# WEEK 6

## Canvas Engagements 4/04/2022

There were some helpful students who helped me learn how customised I can make the portfolio.

### Canvas question:



Francisca Nel  
3 Apr 17:42 Last reply 4 Apr 0:11

Hi Steffan,

Can we make the portfolio pretty (ex. make the tables colourful)?

Thanks

[Reply](#) | 2 replies

### Responses:



Haruki Uda  
3 Apr 20:05

⋮

Hi Francisca,

I'm not Steffan but I need my A on my portfolio so I'm just gonna answer this.

Yes, you can make the portfolio pretty as long as it meets the basic formatting requirements state in the portfolio part a outline. As seen in the image attached.

Good luck on your portfolio!

---

FORMATTING AND STYLE REQUIREMENTS:

Each page of the *Portfolio* must have:

- Your Student ID in the header, aligned top left.
- The page-number out of total-number-of-pages in the footer, aligned bottom-right.

Your *Portfolio*'s formatting must feature:

- Text in an easily readable font, with single line spacing.
- Source code in bold **Courier New**, with single line spacing, and adhering to good whitespace practices.
- Appropriately styled headings or sections.

Images can be included, such as:

- Scans/photos of handwritten notes, program designs, or traces.
- Screenshots of source code implementations, compiler errors, testing via program execution, debugging practices and program runtime errors.

Ensure all images are readable and do not obstruct text within the document. All formatting must support the reader to easily understand the document.



Tony Yee  
4 Apr 0:11

⋮

Haruki Uda

3 Apr 20:05

Hi Francisca, I'm not Steffan but I need my A on my portfolio so I'm just gonna answer this. Yes, you can make the portfolio pretty as long as it meets the basic for ...

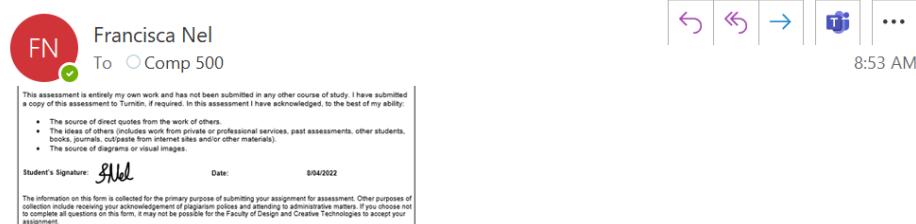
[Read more](#)

As Haruki said above, yes, as long as it meets the formatting requirement. Also from page 1 of the brief "Each student's Portfolio will be creative, unique, and personal – there is no prescribed, pre-made template to follow – no two portfolios will be the same."

4/04/2022

*Note: the comp500 inbox email is mine:*

Question regarding what to fill out in the cover sheet of the portfolio



I also don't know how to put a tick in the boxes, as Steffan requires that it be ticked instead of crossed,  
But it comes out as crossed when the default value in properties is set to 'checked'

1. I understand it is my responsibility to keep a copy of my assignment.  Yes  No

### Canvas question and response:

 Steffan Hooper AUTHOR TEACHER  
4 Apr 12:08 Last reply 4 Apr 12:10

From the COMP500/ENSE501 inbox:

"I don't know how to put a tick in the boxes, as Steffan requires that it be ticked instead of crossed,

But it comes out as crossed when the default value in properties is set to 'checked'"

[Reply](#) | 1 reply

---

 Steffan Hooper AUTHOR TEACHER  
4 Apr 12:10

Hello

Please note this is not my requirement, the form is required by the Faculty of Design and Creative Technologies.

I recommend using the Adobe PDF software to fill in the form and digitally add a Tick.

Thanks  
Steffan

Though I did not use adobe PDF software to digitally add a tick,

In response to this feedback, I used paint.net to make the provided tick image into a transparent image (to the left). It can overlay the tick boxes via the settings:  
picture format > warp text > in front of text.



Result:



**Week 6 lecture 015 Notes- Nested Loops, goto**

Notes Retrieved from Steffan Hooper, W06 Lec 016, 2022

**04/04/2022 - Monday**

For loops and while loops can be nested.

Nested Loops	
Pseudocode	Source Code
<b>REPEAT 5 times</b> ← outer loop <b>PRINT +</b>  <b>REPEAT 5 times</b> ← Nested inner loop <b>PRINT -</b> <b>ENDREPEAT</b>  <b>PRINT +</b> <b>PRINT newline</b> <b>ENDREPEAT</b>	<pre>. . . for (int i = 1; i &lt;= 5; ++i) {     printf("+");     for (int k = 1; k &lt;= 5; ++k)     {         printf("-");     }     printf("+\n"); } . . .</pre>
OUTPUT	
<pre>+----+ +----+ +----+ +----+ +----+</pre>	

ASCII art can be directed by the user- ex, input can determine an ASCII rectangle's width or length by iterating the input through a loop.

REMEMBER to validate user input (to prevent errors)

The **goto** keyword can be used to alter the normal sequence of a program:

goto example
Source Code
<pre>... {     int count_down = 10;  tick: ← goto needs a label (label is an identifier)     printf("%d...\\n", count_down);      --count_down;      if (count_down &gt; 0)     {         goto tick; ← the goto will cause the program to jump to the label (tick)     }     printf("Timer has expired!\\n"); ... }</pre>

AVOID using goto (it's considered bad programming style) as it can make the program's logical flow complex to follow.

## LAB 6: 6x01a – Pseudo Code Loop

07/04/2022 – Thursday (lab), Week 6

### Program aim

Given the following pseudo code, write the corresponding source code

### Pseudocode (given by exercise)

```
FOR (integer n is initially equal to 0  
      keep looping as long as n is less than 10  
      add 1 to n at the end of the loop)  
Each time around the loop do this:
```

```
IF (n is greater than 4) THEN  
    PRINT n followed by a newline.  
ELSE  
    PRINT the result of 9 - n followed by a newline.  
ENDIF
```

```
ENDFOR
```

### Purpose of program

The program needs to count from 9 down to 5, then count up from 5 to 9 using a loop

### Implementation

#### Source Code

#### Output/s

```
1 #include <stdio.h>  
2  
3 int main(void)  
4 {  
5     for (int n=0; n<10; n++)  
6         if (n > 4)  
7             {  
8                 printf("%d \n", n);  
9             }  
10        else  
11            {  
12                printf("%d \n", 9 - n);  
13            }  
14  
15     return 0;  
16 }
```

```
9  
8  
7  
6  
5  
5  
6  
7  
8  
9
```

### Lessons learned

I learned how to follow pseudocode of a **for** loop

## LAB 6: 6x02a – Fizz Buzz

07/04/2022 – Thursday (lab), Week 6

### Program aim

Ask the user for a whole number. Then print out from 1 to the whole number entered by the user, with each number on a newline. If the number is a multiple of 3, print **Fizz** instead of the number. If the number is a multiple of 5, print **Buzz** instead of the number. If the number is both a multiple of 3 and 5, print **FizzBuzz**.

### Pseudocode (Plan)

```

FOR (integer n is initially equal to 0
      keep looping as long as n is less than number_input
      add 1 to n at the end of the loop)
  Each time around the loop do this:

    IF (n is a multiple of 3) THEN <- (multiple: use modulus (%) to check if remainder is 0)
      PRINT Fizz followed by a newline.
    ELSE IF (n is a multiple of 5) THEN
      PRINT Buzz followed by a newline.
    ELSE IF (n is a multiple of 3 AND n is a multiple of 5) THEN
      PRINT FizzBuzz followed by a newline.
    ELSE
      PRINT n followed by a newline.
    ENDIF
ENDFOR

```

### Implementation

#### Source Code

```

1  #include <stdio.h>
2
3  int main(void)
4  {
5      int number_input = 0;
6      printf("Enter a whole number: ");
7      scanf("%d", &number_input);
8
9      for (int n = 1; n <= number_input; n++)
10     {
11         if (n % 3 == 0) // does n/3 have remainder 0?
12         {
13             printf("Fizz \n");
14         }
15         else if (n % 5 == 0) // does n/5 have remainder 0?
16         {
17             printf("Buzz \n");
18         }
19         else if (n % 3 == 0 || n % 5 == 0) // does n/3 AND n/5 have remainder 0?
20         {
21             printf("FizzBuzz");
22         }
23         else // otherwise, print the number normally
24         {
25             printf("%d \n", n);
26         }
27     }
28
29     return 0;
30 }

```

#### Output/s

```

Enter a whole number: 25
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
Fizz
16
17
Fizz
19
Buzz
Fizz
22
23
Fizz
Buzz

```

### Lessons learned

I learned how to use **if** and **else** in a **for** loop with regards to printing and using modulus.

**LAB 6: 6x03a – Highest Number**

07/04/2022 – Thursday (lab), Week 6

**Program aim**

Ask the user to input a series of numbers. When the user enters zero, the program must stop accepting numbers. The program must then report to the user the highest number entered.

**Pseudocode (Plan)**

```

READ number_input
IF number_input is 0
    PRINT No numbers entered
ELSE
    WHILE number_input is not equal to zero:
        PRINT Enter a number (0 to quit):
        READ number_input
        IF number_input is larger than largest number
            Save number_input to largest_number
        PRINT The highest number was largest_number
    ENDWHILE
ENDIF

```

**Implementation**

Source Code	Output/s
<pre> 1 #include &lt;stdio.h&gt; 2 3 int main(void) 4 { 5     int number_input = 0; 6 7     printf("Enter a number (0 to quit): "); 8     scanf("%d", &amp;number_input); 9     int largest_number = number_input; 10 11    if (number_input == 0) 12    { 13        printf("\nNo numbers entered!"); 14    } 15    else 16    { 17        while (number_input != 0) 18        { 19            printf("Enter a number (0 to quit): "); 20            scanf("%d", &amp;number_input); 21            if (number_input &gt; largest_number) 22            { 23                largest_number = number_input; 24            } 25        } 26        printf("\nThe highest number was %d", largest_number); 27    } 28 29    return 0; 30 }</pre>	<pre> Enter a number (0 to quit): 1 Enter a number (0 to quit): 100 Enter a number (0 to quit): 3 Enter a number (0 to quit): 80 Enter a number (0 to quit): 0 The highest number was 100  Enter a number (0 to quit): 1 Enter a number (0 to quit): 2 Enter a number (0 to quit): 3 Enter a number (0 to quit): 3 Enter a number (0 to quit): 2 Enter a number (0 to quit): 1 Enter a number (0 to quit): 0 The highest number was 3  Enter a number (0 to quit): 0 No numbers entered! </pre>

**Lessons learned**

I learned how to implement `while` loops within `if else` statements!

## **LAB 6: 6x04a – Guess My Number**

**07/04/2022 – Thursday (lab), Week 6**

### **Program aim**

Write a program which gives the user seven turns to guess a secret number between 1 and 100.

### **Pseudocode (Plan)**

```
GENERATE RANDOM NUMBER BETWEEN 1-100
DECLARE guesses_left as 7
DECLARE guesses_total as 0
DECLARE guess as 0

PRINT I'm thinking of a number between 1 and 100

WHILE guess does not equal answer
    PRINT What is your guess?
    READ guess
    IF guess is less than answer
        PRINT Your guess of (guess) is too low!
    ELSE IF guess is greater than answer
        PRINT Your guess of (guess) is too high!
    PRINT you have --guesses_left turns left!
    ++guesses_total

    IF guess equals answer
        PRINT You guessed it in guesses_total turns
        PRINT Well done!
    IF guess doesn't equal answer
        PRINT You ran out of turns! The answer was answer
```

**LAB 6: 6x04a – Guess My Number (Continued)**

07/04/2022 – Thursday (lab), Week 6

**Implementation****Source Code**

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  int main(void)
6  {
7      // seed random generator
8      srand(time(0));
9
10     // generate secret number
11     int answer = (rand() % 100) + 1;
12
13     int guesses_left = 7;
14     int guesses_total = 0;
15     int guess = 0;
16
17     printf("I'm thinking of a number between 1 and 100 \n\n");
18     printf("You have %d turns left!", guesses_left);
19
20     while (guess != answer && guesses_left != 0)
21     {
22         printf("\n\nWhat is your guess? ");
23         scanf("%d", &guess);
24         if (guess < answer)
25         {
26             printf("Your guess of %d is too low!", guess);
27         }
28         else if (guess > answer)
29         {
30             printf("Your guess of %d is too high!", guess);
31         }
32
33         printf("\nYou have %d turns left!", --guesses_left);
34         ++guesses_total;
35     }
36
37     if (guess == answer)
38     {
39         printf("\nYou guessed it in %d turns. \n\n Well done!", guesses_total);
40     }
41     else if (guess != answer)
42     {
43         printf("\n\nYou ran out of turns! The number was %d", answer);
44     }
45
46
47     return 0;
48 }
```

## LAB 6: 6x04a – Guess My Number (Continued)

07/04/2022 – Thursday (lab), Week 6

### Output/s

#### Using 7 turns

```
I'm thinking of a number between 1 and 100
You have 7 turns left!
What is your guess? 50
Your guess of 50 is too low!
You have 6 turns left!
What is your guess? 75
Your guess of 75 is too low!
You have 5 turns left!
What is your guess? 80
Your guess of 80 is too low!
You have 4 turns left!
What is your guess? 90
Your guess of 90 is too low!
You have 3 turns left!
What is your guess? 95
Your guess of 95 is too high!
You have 2 turns left!
What is your guess? 93
Your guess of 93 is too low!
You have 1 turns left!
What is your guess? 94
You have 0 turns left!
You guessed it in 7 turns.
Well done!
```

#### Using 6 turns

```
I'm thinking of a number between 1 and 100
You have 7 turns left!
What is your guess? 50
Your guess of 50 is too high!
You have 6 turns left!
What is your guess? 25
Your guess of 25 is too high!
You have 5 turns left!
What is your guess? 15
Your guess of 15 is too high!
You have 4 turns left!
What is your guess? 5
Your guess of 5 is too high!
You have 3 turns left!
What is your guess? 3
Your guess of 3 is too low!
You have 2 turns left!
What is your guess? 4
You have 1 turns left!
You guessed it in 6 turns.
Well done!
```

#### Running out of turns

```
I'm thinking of a number between 1 and 100
You have 7 turns left!
What is your guess? 50
Your guess of 50 is too low!
You have 6 turns left!
What is your guess? 5
Your guess of 5 is too low!
You have 5 turns left!
What is your guess? 80
Your guess of 80 is too high!
You have 4 turns left!
What is your guess? 03
Your guess of 3 is too low!
You have 3 turns left!
What is your guess? 84
Your guess of 84 is too high!
You have 2 turns left!
What is your guess? 3
Your guess of 3 is too low!
You have 1 turns left!
What is your guess? 4
Your guess of 4 is too low!
You have 0 turns left!
You ran out of turns! The number was 61
```

#### Note

The program description did not show what to output if the user ran out of turns, so I added an if statement for if the user runs out of turns that prints:  
“You ran out of turns! The number was (answer)”

## LAB 6: 6x05a – Vowel Replace

07/04/2022 – Thursday, Week 6

### Program aim

User can input text up to 80 characters in length. The program must iterate through the string and replace the following vowels:

Vowel	Replacement Character
A	\$
E	#
I	@
O	*
U	=

### Pseudocode (Plan)

```
INCLUDE string.h for access to strlen (array size)

DECLARE string_array of size 80

PRINT >

SCAN input as string_array c-string
DECLARE array_size as strlen (array size) of string_array

WHILE count is less than array_size
    IF string_array element is 'a' OR 'A'
        SAVE string_array element as '$'

    ELSE IF string_array element is 'e' OR 'E'
        SAVE string_array element as '#'

    ELSE IF string_array element is 'i' OR 'I'
        SAVE string_array element as '@'

    ELSE IF string_array element is 'o' OR 'O'
        SAVE string_array element as '*'

    ELSE IF string_array element is 'u' OR 'U'
        SAVE string_array element as '='

    Count++ (increment)
ENDWHILE

PRINT the new string_array
```

**LAB 6: 6x05a – Vowel Replace (Continued)**

07/04/2022 – Thursday, Week 6

**Implementation****Source Code**

```

1  #include <stdio.h>
2
3  int main(void)
4  {
5      char string_array[80] = "\0";
6      int c = 0;
7
8      printf("> ");
9      scanf("%[^\\n]", &string_array);
10
11     while (string_array[c] != '\0') // loop while count is less than size of array
12     {
13         if (string_array[c] == 'a' || string_array[c] == 'A') // if a or A, save element as $
14         {
15             string_array[c] = '$';
16         }
17         else if (string_array[c] == 'e' || string_array[c] == 'E') // if e or E, save element as #
18         {
19             string_array[c] = '#';
20         }
21         else if (string_array[c] == 'i' || string_array[c] == 'I') // if i or I, save element as @
22         {
23             string_array[c] = '@';
24         }
25         else if (string_array[c] == 'o' || string_array[c] == 'O') // if o or O, save element as *
26         {
27             string_array[c] = '*';
28         }
29         else if (string_array[c] == 'u' || string_array[c] == 'U') // if u or U, save element as =
30         {
31             string_array[c] = '=';
32         }
33         c++;
34     }
35     printf("%s", string_array);
36
37     return 0;
38 }
```

**Output/s**

```
> Francisca |> Hello world! |> Three different words |> a e i o u
Fr$nc@sc$ |H#ll* w*rld! |fThr## d@ff#r#nt w*rds |$ # @ * =
```

**Lessons learned**

I learnt how to replace the elements of characters of a c-string using a `while` loop and `if` `else` statements!

## LAB 6: 6x06a – Nested for Loop Square

07/04/2022 – Thursday, Week 6

### Program aim

Write a program, using nested for loops, which prints the following 5x5 square:

```
#####
#####
```

USE THE FOLLOWING PRINTF STATEMENTS ONLY: `printf("#")`; and `printf("\n")`;

### Pseudocode (Plan)

```
FOR (integer height equal to 0,
      loop until height is less than 5,
      add 1 to height each iteration)

FOR (integer width equal to 0,
      loop until width is less than 5,
      add 1 to width each iteration)
    PRINT #
ENDFOR

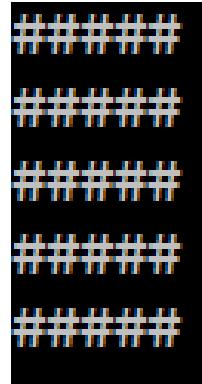
PRINT newline
ENDFOR
```

### Implementation

#### Source Code

#### Output/s

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     for (int height = 0; height < 5; height++) // outer loop: determines height
6     {
7         for (int width = 0; width < 5; width++) // inner loop: determines width
8         {
9             printf("#");
10        }
11        printf("\n"); // creates new rows everytime the inner loop finishes
12    }
13    return 0;
14 }
```



### Lessons learned

I learned how to use nested for loops to print an ASCII square!

**LAB 6: 6x07a – User Defined Table Size**

07/04/2022 – Thursday, Week 6

**Program aim**

Using nested for loops, print a user-defined multiplication table.

When user inputs a width of 15 and a height of 3, It should look like this:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
3	6	9	12	15	18	21	24	27	30	33	36	39	42	45

**Pseudocode (Plan)**

```

PRINT Width?
READ width_input

PRINT Height?
READ height_input

FOR (integer height equal to 0,
      loop until height is less than or equal to height_input,
      add 1 to height each iteration)

FOR (integer width equal to 0,
      loop until width is less than or equal to width_input,
      add 1 to width each iteration)

    PRINT width*height

ENDFOR

PRINT newline
ENDFOR

```

**Implementation****Source Code**

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     int width_input = 0;
6     int height_input = 0;
7
8     printf("Width? ");
9     scanf("%d", &width_input);
10
11    printf("Height? ");
12    scanf("%d", &height_input);
13
14    printf("\n");
15
16    for (int height = 1; height <= height_input; height++) // outer loop
17    {
18        for (int width = 1; width <= width_input; width++) // inner loop
19        {
20            printf("%4d", width*height); // REMINDER the 4 in %4d is width option
21        }
22        printf("\n"); // creates new rows everytime the inner loop finishes
23    }
24
25    return 0;
26 }

```

**Output/s**

Width? 15 Height? 5	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75	Width? 5 Height? 6	Width? 2 Height? 8
	1 2 3 4 5 2 4 6 8 10 3 6 9 12 15 4 8 12 16 20 5 10 15 20 25 6 12 18 24 30	1 2 3 6 4 8 5 10 6 12 7 14 8 16	

**Lessons learned**

I used the `printf` width option for the first time, and I learned I can iterate the printing of math operations into shapes using `for` loops

## LAB 6: 6x08a – Rectangle Drawing

07/04/2022 – Thursday, Week 6

### Program aim

Write a program which asks the user for the width and height of a rectangle to draw. The program must then draw the shape using ASCII art.

### Pseudocode (Plan)

```
READ width_input
READ height_input

// FIRST ROW OF THE RECTANGLE (1 is subtracted as + adds columns)
PRINT +
FOR (w equals 1, loop while w is less than width_input minus 1, +1 to w
each loop)
    PRINT -
ENDFOR
PRINT + followed by newline

// BODY OF RECTANGLE
FOR (h equals 1, loop while h is less than height_input minus 1, +1 to h
each loop)
    PRINT |
        FOR (w equals 1, loop while w is less than width_input minus 1, +1 to w
each loop)
            PRINT " " (space)
        ENDFOR
    PRINT | followed by newline

// LAST ROW OF THE RECTANGLE (basically a copy of the first row of the
rectangle)
PRINT +
FOR (w equals 1, loop while w is less than width_input minus 1, +1 to w
each loop)
    PRINT -
ENDFOR
PRINT + followed by newline
```

**LAB 6: 6x08a – Rectangle Drawing (Continued)**

07/04/2022 – Thursday, Week 6

Implementation	
Source Code	Output/s
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      int width_input = 0; 6      int height_input = 0; 7 8      printf("Rectangle width? "); 9      scanf("%d", &amp;width_input); 10 11     printf("Rectangle height? "); 12     scanf("%d", &amp;height_input); 13 14     // FIRST ROW OF THE RECTANGLE 15     printf("+"); 16     for (int w = 1; w &lt; width_input - 1; ++w) 17     { 18         printf("-"); 19     } 20     printf("\n"); 21 22     // BODY OF RECTANGLE 23     for (int h = 1; h &lt; height_input - 1; ++h) 24     { 25         printf(" "); 26         for (int w = 1; w &lt; width_input - 1; ++w) 27         { 28             printf(" "); 29         } 30         printf("\n"); 31     } 32 33     // LAST ROW OF THE RECTANGLE 34     printf("+"); 35     for (int w = 1; w &lt; width_input - 1; ++w) 36     { 37         printf("-"); 38     } 39     printf("\n"); 40 41     return 0; 42 }</pre>	<pre> Rectangle width? 2 Rectangle height? 3 ++     ++</pre> <pre> Rectangle width? 5 Rectangle height? 5 +++++             +++++</pre> <pre> Rectangle width? 10 Rectangle height? 3 -----+     -----+</pre> <pre> Rectangle width? 2 Rectangle height? 2 ++ ++</pre> <pre> Rectangle width? 15 Rectangle height? 2 -----+ -----+</pre>

**Lessons learned**

I mixed and matched printing lines with and without for loops due to the corners of the rectangle having to be different than the dash ‘-’ symbol. Perhaps in the future when I’m more skilled with loops, I may be able to create the same rectangle using loop only without repetitive print statements

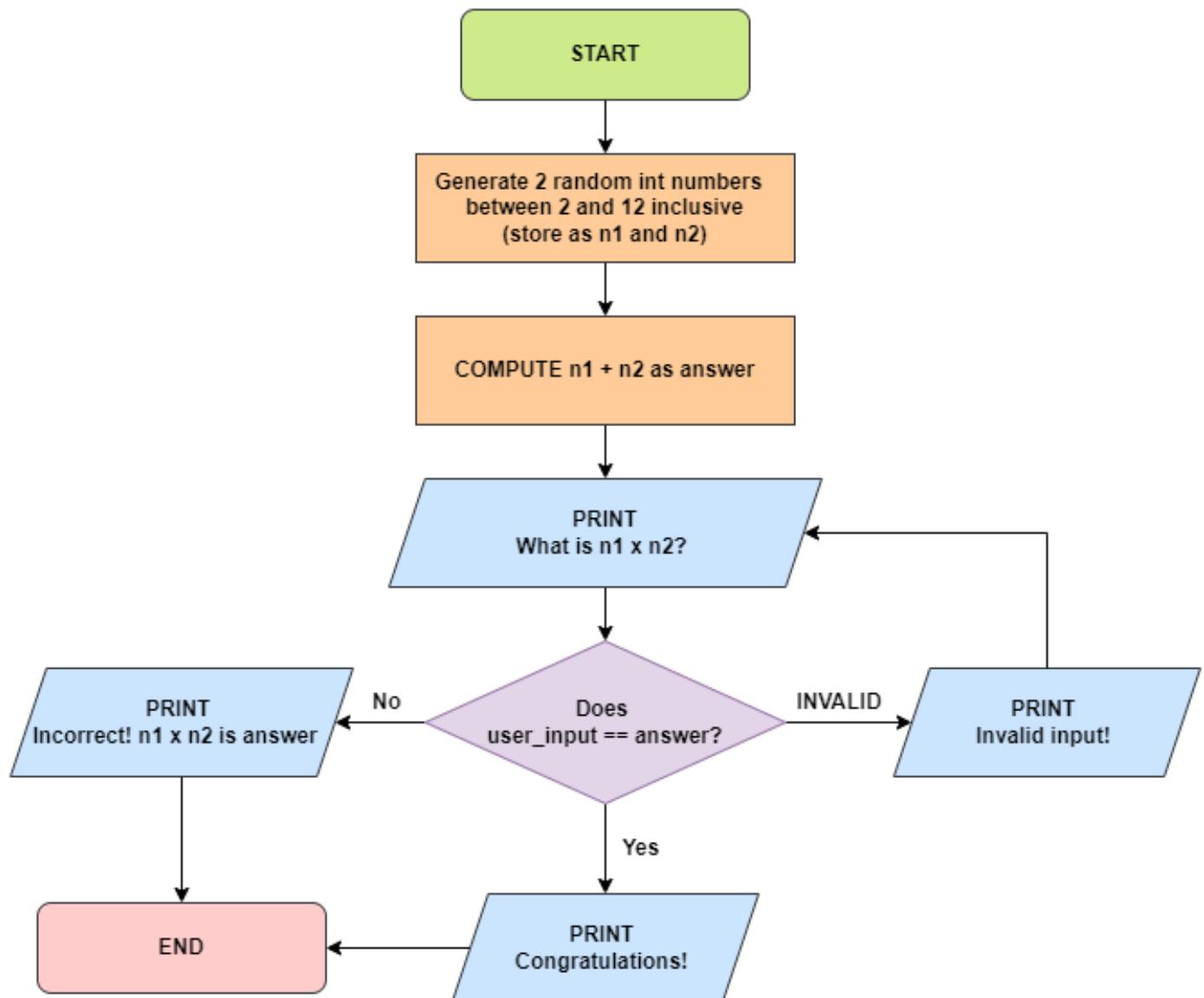
## LAB 6: 6x09a – Math Quiz Validator

07/04/2022 – Thursday, Week 6

### Program aim

Generate two random whole numbers between 2 and 12, inclusive, and ask the user to solve the product of the two numbers. Check if the user's arithmetic is correct, and then provide an output either congratulating the user for their correct answer or correcting them by providing the correct answer.

### Flowchart plan (Created with draw.io)



## NOTES FROM W04 Example Code 012g- Validating Input

I'm going back to this exercise example to understand how validating input works!

```
int number = 0;

printf("Input a number: ");
int scan_result = 0;

while (0 == scan_result)
{
    scan_result = scanf("%d", &number); → Creating scan result from scanf

    if (0 == scan_result) → If scan result is invalid:
    {
        printf("Please enter a number! Try again: "); → re-input

        char bad_input = 0;
        while (bad_input != '\n')
        {
            scanf("%c", &bad_input); → scan bad input to prevent error
        }
    }

    printf("The number: %d was entered.\n", number); → print if input is valid
```

### STEPS:

If I want to validate the input of ex. An integer to a char:

- Create a scan result which detects whether scan is invalid. (`scan_result = scanf(...)`)
- If scan result equates to 0, then it is invalid.
- From here, ask user to re-input and clear the input buffer/scan bad input

**LAB 6: 6x09a – Math Quiz Validator (Continued)**

07/04/2022 – Thursday, Week 6

Implementation	
Source Code	Output/s
<pre> 1 #include &lt;stdio.h&gt; 2 #include &lt;stdlib.h&gt; 3 #include &lt;time.h&gt; 4 5 int main(void) 6 { 7     srand(time(0)); 8 9     int n1 = (rand() % 11) + 2; 10    int n2 = (rand() % 11) + 2; 11    int answer = n1 * n2; 12 13    int user_input = 0; 14    int validate = 0; 15 16    while (validate == 0) 17    { 18        printf("What is %d x %d? ", n1, n2); 19        validate = scanf("%d", &amp;user_input); 20 21        if (validate == 0) 22        { 23            printf("Invalid input!"); 24            scanf("%*[^\n]"); 25            printf("\n"); 26        } 27    } 28    if (user_input == answer) 29    { 30        printf("\nCongratulations!"); 31    } 32    else 33    { 34        printf("\nIncorrect! %d x %d is %d", n1, n2, answer); 35    } 36 37 38 }</pre>	<p>What is 4 x 8? twenty four Invalid input! What is 4 x 8? TWENTY FOUR Invalid input! What is 4 x 8? 24  Incorrect! 4 x 8 is 32</p> <p>What is 4 x 11? 44</p> <p>Congratulations!</p> <p>What is 11 x 11? #\$&amp;*@(*\$ Invalid input! What is 11 x 11? ????? Invalid input! What is 11 x 11? HELLO Invalid input! What is 11 x 11? 1111  Incorrect! 11 x 11 is 121</p>
Lessons learned	
<p>From the Drop-in Lecture Q and A Session 017 recording, I learnt:</p> <ul style="list-style-type: none"> <li>– Bad input can cause infinite loops. To fix this, clean the buffer by scanning in everything in the buffer until newline is entered (use this: <code>scanf("%*[^\n]")</code>)</li> <li>– To validate input, create a new variable that saves the <code>scanf</code> statement. If that variable == 0, then it is invalid.</li> </ul>	

**LAB 6: 6x10a – Game Wander**

07/04/2022 – Thursday, Week 6

**Program aim**

Create a program that tracks a game player on a two-dimensional grid.

**Instructions (plan)**

1. The player starts at (0, 0).  
POSITIVE x is east and NEGATIVE x is west  
POSITIVE y is north and NEGATIVE y is south
2. Allow the player to choose to move north, south, east, or west.
3. Update the player's position based upon the user's choice.
4. If the user chooses to quit, print the final position of the player.

**Implementation****Source Code**

```

1  #include <stdio.h>
2
3  int main(void)
4  {
5      int x_coord = 0; // --West, ++East
6      int y_coord = 0; // --South, ++North
7
8      char player_input = '\0';
9
10     printf("The player starts at (0, 0)... \n");
11
12     while (player_input != 'q')
13     {
14         printf("Player at (%d, %d) - move N,S,E,W (q to quit)? ", x_coord, y_coord);
15         scanf(" %c", &player_input);
16
17         if (player_input == 'N' || player_input == 'n')
18         {
19             ++y_coord;
20         }
21         else if (player_input == 'S' || player_input == 's')
22         {
23             --y_coord;
24         }
25         else if (player_input == 'E' || player_input == 'e')
26         {
27             ++x_coord;
28         }
29         else if (player_input == 'W' || player_input == 'w')
30         {
31             --x_coord;
32         }
33     }
34
35     return 0;
36 }
```

**Output/s**

The player starts at (0, 0)...  
 Player at (0, 0) - move N,S,E,W (q to quit)? N  
 Player at (0, 1) - move N,S,E,W (q to quit)? n  
 Player at (0, 2) - move N,S,E,W (q to quit)? e  
 Player at (1, 2) - move N,S,E,W (q to quit)? E  
 Player at (2, 2) - move N,S,E,W (q to quit)? w  
 Player at (1, 2) - move N,S,E,W (q to quit)? q  
 The player ends at (1, 2)

The player starts at (0, 0)...  
 Player at (0, 0) - move N,S,E,W (q to quit)? s  
 Player at (0, -1) - move N,S,E,W (q to quit)? s  
 Player at (0, -2) - move N,S,E,W (q to quit)? s  
 Player at (0, -3) - move N,S,E,W (q to quit)? s  
 Player at (0, -4) - move N,S,E,W (q to quit)? s  
 Player at (0, -5) - move N,S,E,W (q to quit)? s  
 Player at (0, -6) - move N,S,E,W (q to quit)? w  
 Player at (-1, -6) - move N,S,E,W (q to quit)? q  
 The player ends at (-1, -6)

The player starts at (0, 0)...  
 Player at (0, 0) - move N,S,E,W (q to quit)? q  
 The player ends at (0, 0)

**Lessons learned**I learned how to update integer variables through **if** **else** statements within a **while** loop

**APA 7<sup>th</sup> Referencing**  
**08/04/2022 – Friday, Week 6**

Hooper, S. (2022). *Week 1 - Lecture 1* [Video]. Panopto.

<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=a55f74da-c3ca-4b72-88c1-ade20169c61f>

Hooper, S. (2022). *Week 1 - Lecture 2* [Video]. Panopto.

<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=bf999f48-279f-4bd2-ac65-ade20169c6bf>

Hooper, S. (2022). *Week 1 - Lecture 3* [Video]. Panopto.

<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=0e074bd7-679a-444f-9626-ade20169c76b>

Hooper, S. (2022). *Week 1 - Lecture 4* [Video]. Panopto.

<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=6f6d274e-ad96-4e31-a7c2-ade20169c806>

Hooper, S. (2022). *Week 2 - Lecture 5* [Video]. Panopto.

<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=421ac0b0-e913-42e3-a6f6-ade20169c8b0>

Hooper, S. (2022). *Week 2 - Lecture 6* [Video]. Panopto.

<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=8698b2d0-e014-4eb1-a663-ade20169c93f>

Hooper, S. (2022). *Week 3 - Lecture 7* [Video]. Panopto.

<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=a02f5329-dd74-45af-8873-ade20169c9f3>

Hooper, S. (2022). *Week 3 - Lecture 8* [Video]. Panopto.

<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=850e3d02-0e46-4ef7-82be-ade20169caab>

Hooper, S. (2022). *Week 3 - Lecture 9* [Video]. Panopto.

<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=a09bad2d-3861-47a4-9964-ade20169cbcc>

Hooper, S. (2022). *Week 4 - Lecture 10* [Video]. Panopto.

<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=64afa5ff-8399-4e92-afa5-ade20169cc78>

**APA 7<sup>th</sup> Referencing**  
**08/04/2022 – Friday, Week 6**

Hooper, S. (2022). *Week 4 - Lecture 11* [Video]. Panopto.  
<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=5ef39960-40ac-4c44-9785-ade20169cd86>

Hooper, S. (2022). *Week 4 - Lecture 12* [Video]. Panopto.  
<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=910f2597-53a1-44bb-93e4-ade20169ce23>

Hooper, S. (2022). *Week 5 - Lecture 13* [Video]. Panopto.  
<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=bc7832ff-1f48-43e8-b50c-ade20169cf2d>

Hooper, S. (2022). *Week 5 - Lecture 14* [Video]. Panopto.  
<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=c83adc05-915a-4418-ac41-ade20169d015>

Hooper, S. (2022). *Week 5 - Lecture 15* [Video]. Panopto.  
<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=ba6c7a8f-affb-429c-9f79-ade20169d128>

Hooper, S. (2022). *Week 6 - Lecture 16* [Video]. Panopto.  
<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=1d924172-e101-45be-9042-ade20169d1b0>

Hooper, S. (2022). *Week 6 - Lecture 17* [Video]. Panopto.  
<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=6631564d-7567-4028-b646-ade20169d254>

Hooper, S. (2022). *Week 6 - Lecture 18* [Video]. Panopto.  
<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=bce21bc6-3c7b-4fc7-b09c-ade20169d2eb>

Hooper, S (2022). *Drop-in Lecture Q and A Session 017* [Video]. Panopto.  
<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=b4997cac-56c2-4ec3-8b96-ae6f00284973>

<b>First Name</b>	Francisca	<b>Family Name</b>	Nel	<b>Student ID No</b>	21145382
<b>Course Name</b>	Programming Concepts and Techniques	<b>Course Code</b>	COMP500	<b>Assignment Due Date</b>	7/06/2022
<b>Lecturer</b>	Steffan Hooper	<b>Tutorial Day</b>	Thursday	<b>Date Submitted</b>	7/06/2022
<b>Tutor</b>	Xinyu Hu	<b>Tutorial Time</b>	12:10pm	<b>No.Words/Pages</b>	<b>185 pages</b>

In order to ensure fair and honest assessment results for all students, it is a requirement that the work that you hand in for assessment is your own work. If you are uncertain about any of these matters, then please discuss them with your lecturer.

Plagiarism and Dishonesty are methods of cheating for the purposes of General Academic Regulations (GAR)  
<http://www.aut.ac.nz/calendar>

**Assignments will not be accepted if this section is not completed and signed.**

Please read the following and **tick** ✓ to indicate your understanding:

1. I understand it is my responsibility to keep a copy of my assignment.



No

2. I have signed and read the **Student's Statement below**.



No

3. I understand that a software programme (Turnitin) that detects plagiarism and copying may be used on my assignment.



No

#### Student's Statement:

This assessment is entirely my own work and has not been submitted in any other course of study. I have submitted a copy of this assessment to Turnitin, if required. In this assessment I have acknowledged, to the best of my ability:

- The source of direct quotes from the work of others.
- The ideas of others (includes work from private or professional services, past assessments, other students, books, journals, cut/paste from internet sites and/or other materials).
- The source of diagrams or visual images.

Student's Signature:

Date: 7/06/2022

The information on this form is collected for the primary purpose of submitting your assignment for assessment. Other purposes of collection include receiving your acknowledgement of plagiarism policies and attending to administrative matters. If you choose not to complete all questions on this form, it may not be possible for the Faculty of Design and Creative Technologies to accept your assignment.

**Timesheet**

Entry ID	Week	Date	Start Time	Duration	Session Type	Location
34	7	11/04/2022	1:35pm	2.5 hr	Lectures (pre-rec)	Online/Home
35	7	15/04/2022	12:00pm	1 hr	Lecture (pre-rec)	Online/Home
36	Break	26/04/2022	8:00pm	1.5 hr	Self-directed	Home
37	Break	27/04/2022	6:00pm	3 hr	Self-directed	Home
38	Break	28/04/2022	7:00pm	6 hr	Self-directed	Home
39	Break	29/04/2022	5:00pm	2 hr	Self-directed	Home
40	Break	30/04/2022	6:00pm	3 hr	Self-directed	Home
41	Break	1/05/2022	4:00pm	2.5 hr	Lectures (pre-rec)	Online/Home
42	8	2/05/2022	8:00am	1 hr	Lecture	WA220
43	8	3/05/2022	5:30pm	1 hr	Self-directed	Home
44	8	4/05/2022	11:00am	1 hr	Lecture	WG403
45	8	5/05/2022	12:10pm	3 hr	Lab	WZ514
46	8	6/05/2022	12:00pm	1 hr	Lecture (pre-rec)	Online/Home
47	8	8/05/2022	6:00pm	2 hr	Self-directed	Home
48	9	9/05/2022	8:00am	1 hr	Lecture	WA220
49	9	9/05/2022	4:00pm	2 hr	Self-directed	Home
50	9	10/05/2022	5:30pm	3 hr	Self-directed	Home
51	9	11/05/2022	11:00am	1 hr	Lecture	WG403
52	9	11/05/2022	5:00pm	2 hr	Self-directed	Home
53	9	12/05/2022	12:10pm	3 hr	Lab	WZ514
54	9	13/05/2022	9:00am	1 hr	Lecture	WG403
55	10	16/05/2022	8:00am	1 hr	Lecture	WA220
56	10	18/05/2022	11:00am	1 hr	Lecture	WG403
57	10	19/05/2022	12:10pm	3 hr	Lab	WZ514
58	10	20/05/2022	9:00am	1 hr	Lecture	WG403
59	11	23/05/2022	8:00am	1 hr	Lecture	WA220
60	11	23/05/2022	10:00pm	1 hr	Self-directed	Home
61	11	25/05/2022	11:00am	1 hr	Lecture	WG403
62	11	26/05/2022	11:00am	1 hr	Self-directed	Campus
63	11	26/05/2022	12:00pm	3 hr	Lab	WZ514
64	11	27/05/2022	9:00am	1 hr	Lecture	WG403
65	11	27/05/2022	5:00pm	2.5 hr	Self-directed	Home
66	11	28/05/2022	12:00am	4 hr	Self-directed	Home
67	12	30/05/2022	8:00am	1 hr	Lecture	WA220
68	12	30/05/2022	9:00pm	2 hr	Self-directed	Home
69	12	31/05/2022	3:00pm	5 hr	Self-directed	Home
70	12	1/06/2022	11:00am	1 hr	Lecture	WG403
71	12	1/06/2022	8:00pm	1 hr	Self-directed	Home
72	12	2/06/2022	12:00pm	3 hr	Lab	WZ514
73	12	3/06/2022	12:00pm	1 hr	Lecture	Online/Home
74	12	3/06/2022	2:00pm	3 hr	Self-directed	Online/Home
75	12	4/06/2022	3:00pm	30 min	Self-directed	Online/Home
76	12	5/06/2022	6:00pm	2 hr	Self-directed	Online/Home
77	13	6/06/2022	12:00pm	7 hr	Self-directed	Online/Home
			<b>TOTAL</b>	<b>90.5 hr</b>		

# **WEEK 7**

## Week 6 lecture 017 Notes- Scope, Variable Lifespan

WRITTEN 11/04/2022- Monday, Week 7

Notes Retrieved from Steffan Hooper (2022)

Variables inside **main** are **local** variables.

**Lifespan** depends on where a variable is declared within the sequence of a program. Lifespan is the measure of lines it can be accessed in (ex. 'width' is declared in line 5 whereas area is declared in line 14. Width has a longer lifespan).

Syntax error **c2374** "redefinition; multiple initialization" happens when a variable is initialized multiple times.

Syntax Error **c2065** "undeclared identifier" happens, for example, when the compiler doesn't know what value a variable holds when it is called to print.

When a variable is declared within a for loop, ex: **for (int k=0; k<15; ++k)** - it only exists within the for loop, thus it cannot be used outside the for loop. If it is accessed outside the loop, syntax error **c2065** will occur.

The value a variable holds depends on its position (ex. **k** is defined as -1 in main and **k** defined as 0 in a for loop...When we print **k** outside the loop, its value will be -1 again). When defined in different for loops, new variables of the same name (but different address) is created. DON'T USE THE SAME VARIABLE NAMES as it is bad programming practice.

REMEMBER: to print address of a variable, use **%p** format placeholder.

SORTING ALGORITHM: pseudocode to sort the values in an array into ascending order:

```
INITIALIZE data list as 7 unordered values
DECLARE temp variable
DECLARE j variable

FOR (i is initially equal to 1
      Keep looping UNTIL i is less than 7
      Add 1 to i at start of each loop)
      Each time around the loop do this:

      INITIALIZE temp as i as element of data list
      j equals i minus 1

      WHILE j is greater than or equal to zero AND temp is less than j as
            element of data list

            j plus 1 as element of data REPLACED by j as element of data
            j equals j minus 1
      ENDWHILE
      j plus 1 as element of data equals temp
ENDFOR
```

## Week 6 lecture 018 Notes- Functions, void, return, The Call Stack

WRITTEN 11/04/2022- Monday, Week 7

Notes Retrieved from Steffan Hooper (2022)

I learnt the concept of functions when learning python in high school!

**Source lines of code (SLOC):** software metric measuring a property or characteristic of a piece of source code.

**Functions** help create modular programs through reusable lines of code

Functions have: name, parameters (input data), and return type (output data)

Defining function: `int main(void)`

`data type returned by finished execution of function, function name, function parameters (void = none)`

`printf, scanf, rand, time...` are existing C library functions. Own functions are called the same way.

Calling/invoking function:

**Call:** when a function is invoked (called)

**Caller:** the function that invoked the function currently being executed

**Callee:** the function being called

...

```
int main(void) → main is caller for printf
{
    printf("Hello\n"); → printf is the callee that is called by main
```

```
return 0; → for main, the the caller is the operating system that executed the program
}
```

**return keyword:** `return` is used to stop the execution of a function to go back to the caller function. 0 means the program executed correctly with no errors (as it expects an integer)

**Function prototypes / function declaration:** declaring a function by telling the compiler the function name, input type, and type of data it returns without a body (function definitions have a body). Ex:

```
void print_welcome(void); → function declaration
```

**void Keyword:** represents a type. `void` stores no data. It can be used for a function that takes no arguments or a function that does not return any data to the caller (`void main(void)` instead of `int main(void)` and `return;` instead of `return 0;`).

**CODING STANDARDS: source file layout (top to bottom)**

`#include` directives → function prototypes → main function definition → function definitions

- Function prototypes and definitions are ordered alphabetically by function name
- Each function and section separated by 1 line break

**Black box:** system viewed only in terms of input and output with no knowledge of internal components- like function prototypes.

## Week 6 lecture 018 Notes Continued- Functions, void, return, The Call Stack

WRITTEN 11/04/2022- Monday, Week 7

Notes Retrieved from Steffan Hooper (2022)

**Error: LNK2019: unresolved external symbol...** is a linker error that occurs when a function definition is missing after function declaration (prototype).

**Call stack:** stores information about active functions in an executing program (input, returned data, declared variables...). Accessed through debugging.

**Stack Frame/Activation Records/Activation Frames:** created when function is called and active until function returns. The Stack is a data structure in computer science:  
top of stack is active function → as function is called, new stack frame is pushed onto stack → as current function returns, top of stack frame is popped off stack

**Control Abstraction/black boxing:** used to manage complexity of programs by abstracting it.  
Functions are created to have a single job and are implemented in specific parts of the program.

**The call stack window** in the Visual Studio Debugger shows the current stack frames pushed onto the Call Stack. REMEMBER press F11 to step into function call. **Stepping out** (debug menu > step out) allows the function to keep executing until it returns to caller.

**Don't repeat, use modularity**

Programming process and practice:

- Split large programming tasks into smaller steps
- Create clear flow-control and interfaces between components
- Create modular code to reduce complexity
- It is good programming practice to ensure loops and functions have single exit points (avoid multiple **return** statements and loop **breaks**)
- Modularisation is further used in programs with multiple .c source code files

## Week 7 lecture 019 Notes Continued- Functions, void, return, The Call Stack

WRITTEN 15/04/2022- Friday, Week 7

Notes Retrieved from Steffan Hooper (2022)

**Functions** can also be called subprogram, subroutine, procedure, or callable unit. They can have their own flowchart also, ex: START print\_newline → PRINT newline → END

**Function parameters:** functions can be declared to take input, and can have any number of parameters.

```
void print_middle(int number_of_lines) → function takes number_of_lines input (1 parameter)
{
    for (int k = 0; k < number_of_lines; ++k) → function controls the loop
    {
        printf(" | ");
        printf("\n");
    }
}
```

**Calling with arguments:** when invoking call, provide the arguments. Ex:

```
int main(void) → main function executes print_middle
{
    print_middle(3); → print_middle call takes 3 as argument. 3 is copied into number_of_lines
                      parameter
}
```

Example of multiple parameters:

```
void print_middle(int number_of_lines, int number_of_spaces, char bar, char space)
```

Example of multiple arguments:

```
print_middle(3, 7, '|', ' ') → number of lines, number of spaces, char bar, char space
```

**Call by value:** copying a value from a function into the parameter of a different function.

The argument data passed from caller to callee is copied, and the callee function doesn't get access to the original values stored in the caller (as it is out of scope).

### Functions keywords

- Parameters: allow data to be input into a function by the caller
- Arguments: data the caller inputs into function parameters
- Call by Value: copy caller's argument into a function's parameter
- Local Variables: variables declared in function that are only accessible (local) within their function
- Scope: parameters local to their function

Try avoiding global scope!

### Variable scope coding standards:

- Keep variable lifespan as short as possible. Limit the scope of variables.
- Restrict variable access.
- Avoid Global variables as they create source code that is hard to debug. Avoiding global variables allow variable name reuse (ex. Don't make i, which is used in loops, global!)
- Use local variables
- Use parameters
- Don't use global variables as input, pass input into functions via arguments and parameters instead!
- Name variables after what they store

**LAB 7: 7x01a – Rectangle Computation**

26/04/2022 – Tuesday, mid-semester break

**Program aim**

Ask user for width and height of a rectangle, print an ASCII art diagram, compute area and perimeter, and display results to 3 decimal places. Utilise functions and function declarations. Avoid global variables and `goto`.

**Step-by-step plan**

1. Declare functions: `print_diagram`, `compute_area`, and `compute_perimeter` all with parameters of width and height variables.
2. `print_diagram` function will print the rectangle shown in exercise
3. `compute_area` function will return width x height as `float` (area)
4. `compute_perimeter` function will return width + width + height + height as `float` (perimeter)
5. `main` function will take width and height input from user and store as floats.
6. Print rectangle using `print_diagram` function with width and height as parameters
7. Assign returned value of `compute_area` to `area` variable
8. Assign returned value of `compute_perimeter` to `perimeter` variable
9. Print `area` and `perimeter` values to user

**Implementation****Source Code (lines 3-43)****Output/s / Testing**

```

3 void print_diagram(float width, float height);
4 float compute_area(float width, float height);
5 float compute_perimeter(float width, float height);
6
7 int main(void)
8 {
9     float width = 0.0f;
10    float height = 0.0f;
11
12    printf("Rectangle width? ");
13    scanf("%f", &width);
14    printf("Rectangle height? ");
15    scanf("%f", &height);
16
17    print_diagram(width, height);
18
19    float area = compute_area(width, height);
20    float perimeter = compute_perimeter(width, height);
21
22    printf("The area is: %.3f \n", area);
23    printf("The perimeter is: %.3f \n", perimeter);
24    return 0;
25 }
26
27 void print_diagram(float width, float height)
28 {
29     printf("\n %.2f \n", width);
30     printf("+----+ \n");
31     printf(" | | %.2f \n", height);
32     printf("+----+ \n\n");
33 }
34
35 float compute_area(float width, float height)
36 {
37     return width * height; // return area
38 }
39
40 float compute_perimeter(float width, float height)
41 {
42     return width + width + height + height; // return perimeter
43 }
```

```

Rectangle width? 5
Rectangle height? 4

5.00
+----+
| | 4.00
+----+

The area is: 20.000
The perimeter is: 18.000

Rectangle width? 43
Rectangle height? 55.53

43.00
+----+
| | 55.53
+----+

The area is: 2387.790
The perimeter is: 197.060

Rectangle width? 5.523425
Rectangle height? 90.2342342

5.52
+----+
| | 90.23
+----+

The area is: 498.402
The perimeter is: 191.515

```

**Lessons learned**

I learned how to utilise functions, function declarations, and retrieving return values from functions to assign to variables for the first time in C- it follows the same logic as python functions.

## LAB 7: 7x02a – AI Drink Selection

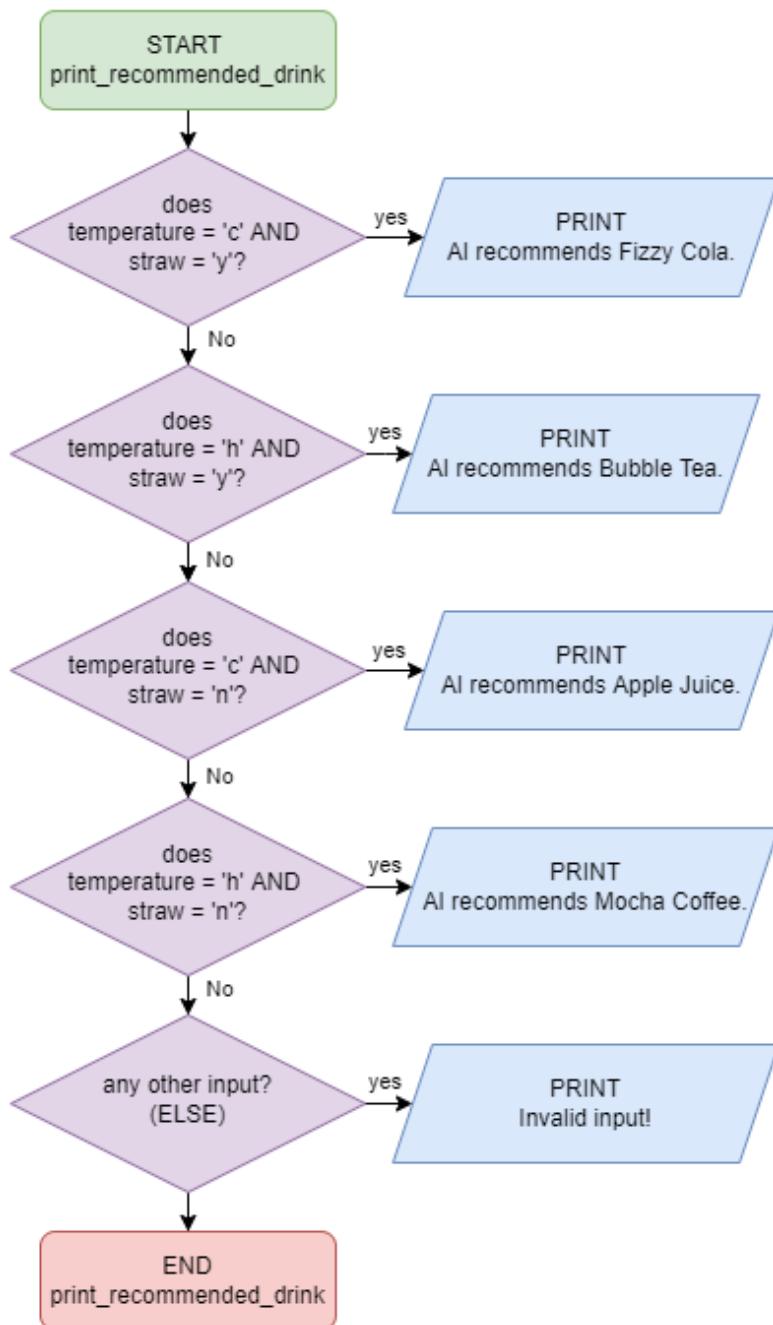
27/04/2022 –Wednesday, mid-semester break

### Program aim

Use selection to choose a user's drink option based on this table:

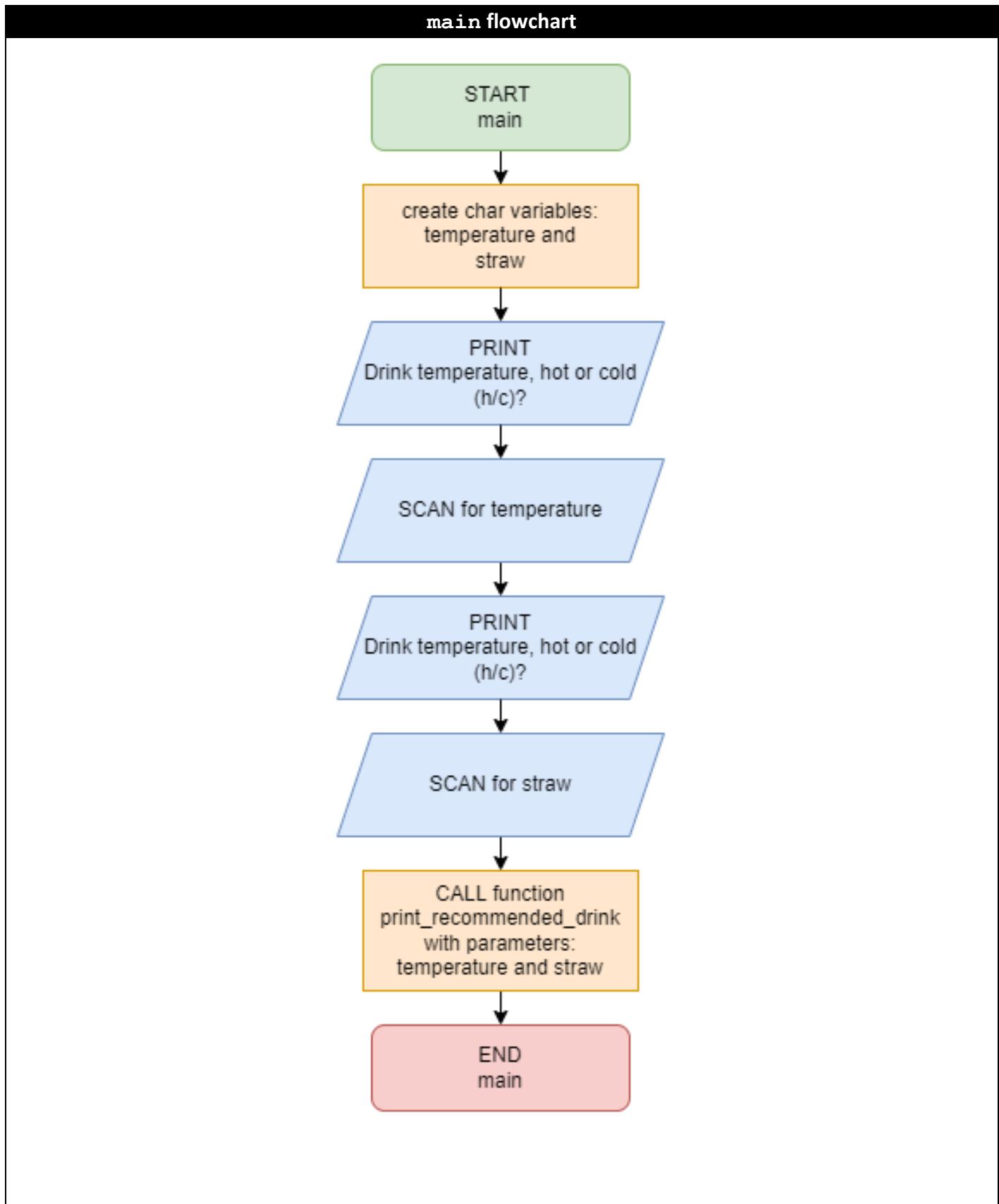
Temperature?	Straw?	AI Recommendation:
Cold	Yes	Fizzy cola
Hot	Yes	Bubble tea
Cold	No	Apple juice
Hot	No	Mocha coffee

### print\_recommended\_drink flowchart



**LAB 7: 7x02a – AI Drink Selection (continued)**

27/04/2022 –Wednesday, mid-semester break



**LAB 7: 7x02a – AI Drink Selection (continued)**

27/04/2022 –Wednesday, mid-semester break

**Implementation****Source Code (lines 3-43)**

```

3 void print_recommended_drink(char temperature, char straw);
4
5 int main(void)
6 {
7     char temperature = '\0';
8     char straw = '\0';
9
10    printf("Drink temperature, hot or cold (h/c)? ");
11    scanf(" %c", &temperature);
12
13    printf("Must use a straw, yes or no (y/n)? ");
14    scanf(" %c", &straw);
15
16    print_recommended_drink(temperature, straw);
17
18    return 0;
19}
20
21 void print_recommended_drink(char temperature, char straw)
22 {
23     if (temperature == 'c' && straw == 'y')
24     {
25         printf("AI recommends Fizzy Cola.");
26     }
27     else if (temperature == 'h' && straw == 'y')
28     {
29         printf("AI recommends Bubble Tea.");
30     }
31     else if (temperature == 'c' && straw == 'n')
32     {
33         printf("AI recommends Apple Juice.");
34     }
35     else if (temperature == 'h' && straw == 'n')
36     {
37         printf("AI recommends Mocha Coffee.");
38     }
39     else
40     {
41         printf("Invalid input!");
42     }
43 }
```

**Output/s / Testing**

Drink temperature, hot or cold (h/c)? c  
 Must use a straw, yes or no (y/n)? y  
 AI recommends Fizzy Cola.

Drink temperature, hot or cold (h/c)? h  
 Must use a straw, yes or no (y/n)? y  
 AI recommends Bubble Tea.

Drink temperature, hot or cold (h/c)? c  
 Must use a straw, yes or no (y/n)? n  
 AI recommends Apple Juice.

Drink temperature, hot or cold (h/c)? h  
 Must use a straw, yes or no (y/n)? n  
 AI recommends Mocha Coffee.

Drink temperature, hot or cold (h/c)? h  
 Must use a straw, yes or no (y/n)? i  
 Invalid input!

Drink temperature, hot or cold (h/c)? j  
 Must use a straw, yes or no (y/n)? a  
 Invalid input!

**Lessons learned**

I learnt how to utilise separate flowcharts for each function to plan a program.

## LAB 7: 7x03a – User Assessment

27/04/2022 –Wednesday, mid-semester break

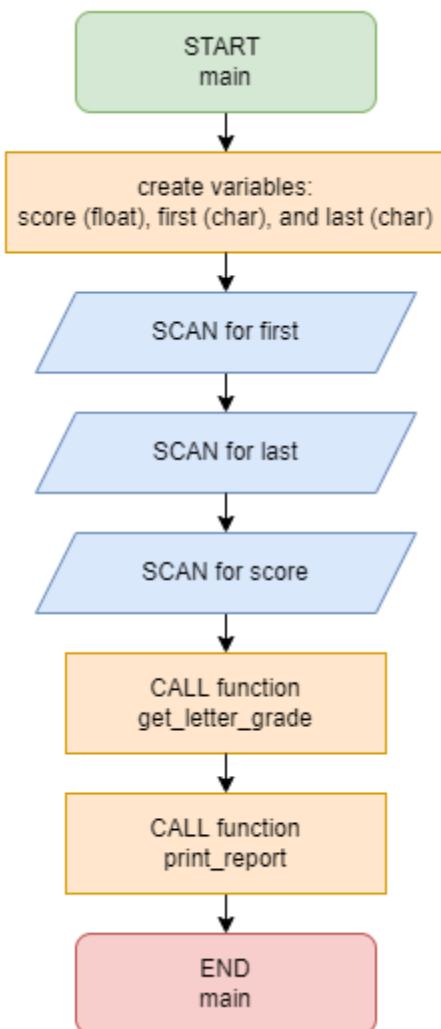
### Program aim

Ask user for first and last initials and assessment score. Compute and display their grade according to the table below:

Assessment score range:	Equivalent letter grade:
$80 \leq \text{score} \leq 100$	A
$65 \leq \text{score} < 80$	B
$50 \leq \text{score} < 65$	C
$0 \leq \text{score} < 50$	D
Any other score	N

### Implementation

#### main Flowchart



#### Source Code (lines 1-27)

```

1  #include <stdio.h>
2
3  char get_letter_grade(float score);
4  void print_report(char first, char last, float score);
5
6  int main(void)
7  {
8      float score = 0.0f;
9      char first = '\0';
10     char last = '\0';
11
12     printf("First name initial? ");
13     scanf(" %c", &first);
14
15     printf("Last name initial? ");
16     scanf(" %c", &last);
17
18     printf("Assessment score? ");
19     scanf("%f", &score);
20
21     get_letter_grade(score);
22
23     print_report(first, last, score);
24
25
26 }
27
  
```

**LAB 7: 7x03a – User Assessment (continued)**

27/04/2022 –Wednesday, mid-semester break

Implementation	
get_letter_grade Flowchart	Source Code (lines 28-51)
<pre> graph TD     Start([START get_letter_grade]) --&gt; Cond1{is score over 80?}     Cond1 -- Yes --&gt; ReturnA[RETURN 'A']     Cond1 -- No --&gt; Cond2{is score between 65 and 79?}     Cond2 -- Yes --&gt; ReturnB[RETURN 'B']     Cond2 -- No --&gt; Cond3{is score between 50 and 64?}     Cond3 -- Yes --&gt; ReturnC[RETURN 'C']     Cond3 -- No --&gt; Cond4{is score between 0 and 49?}     Cond4 -- Yes --&gt; ReturnD[RETURN 'D']     Cond4 -- No --&gt; Cond5{is score anything else?}     Cond5 -- Yes --&gt; ReturnN[RETURN 'N']     End([END get_letter_grade])   </pre>	<pre> 28   char get_letter_grade(float score) 29   { 30     if (score &gt;= 80) 31     { 32       return 'A'; 33     } 34     else if (score &gt;= 65 &amp;&amp; score &lt; 80) 35     { 36       return 'B'; 37     } 38     else if (score &gt;= 50 &amp;&amp; score &lt; 65) 39     { 40       return 'C'; 41     } 42     else if (score &gt;= 0 &amp;&amp; score &lt; 50) 43     { 44       return 'D'; 45     } 46     else 47     { 48       return 'N'; 49     } 50   51     </pre>

## LAB 7: 7x03a – User Assessment (continued)

27/04/2022 –Wednesday, mid-semester break

### Implementation

#### `print_report` step-by-step plan

1. Define `print_report` with parameters first (char), last (char), and score (float)
2. PRINT “(first initial)(last initial)’s assessment score of (score) is a (grade) grade.”  
The `grade` will be the return value of the `get_letter_grade` function

#### Source Code (lines 52-55)

```
52 | void print_report(char first, char last, float score)
53 | {
54 |     printf("%c%c's assessment score of %f is a %c grade. ", first, last, score, get_letter_grade(score));
55 | }
```

### Output/s / Testing

#### A grade output

```
First name initial? A
Last name initial? L
Assessment score? 98.76
AL's assessment score of 98.760002 is a A grade.
```

#### B grade output

```
First name initial? F
Last name initial? N
Assessment score? 60
FN's assessment score of 60.000000 is a B grade.
```

#### C grade output

```
First name initial? K
Last name initial? L
Assessment score? 55.55
KL's assessment score of 55.549999 is a C grade.
```

#### D grade output

```
First name initial? F
Last name initial? N
Assessment score? 44.53
FN's assessment score of 44.529999 is a D grade.
```

#### N grade output

```
First name initial? J
Last name initial? V
Assessment score? -74.3
JV's assessment score of -74.300003 is a N grade.
```

### Lessons learned

I learnt how to get the return of a function and output it from inside a `printf` statement using the line:  
`get_letter_grade(score)`

## LAB 7: 7x04a – Print Triangle

28/04/2022 –Thursday, mid-semester break

### Program aim

Ask user to input integer, which becomes height of an ASCII art triangle. Use the following prototype: **void print\_triangle(int total\_rows);** and define the function for the printing of the triangle only with no **scanf**.

### main Pseudocode

```
PRINT "Height?"  
READ total_rows  
CALL function print_triangle with parameter total_rows
```

### print\_triangle Pseudocode

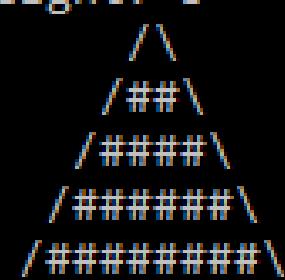
```
FOR i equals 0, loop until i is less than or equal to total_rows, post-increment i by 1 each loop (triangle height)  
    FOR j equals 0, loop until j is less than or equal to total_rows, post-increment j by 1 each loop (left spaces)  
        PRINT " " (space for left side of triangle)  
    PRINT "/" (for left side of triangle before the body)  
    FOR k equals 0, loop until k is less than or equal to (2 x k - 1), post-increment k by 1 each loop (triangle body)  
        PRINT "#" (symbol for body of triangle)  
    PRINT "\\\" (for right side of triangle after body)  
    PRINT "\n" (new line at end of each loop for new row)
```

### Implementation

#### Source Code

```
1  #include <stdio.h>  
2  
3  void print_triangle(int total_rows);  
4  
5  int main(void)  
6  {  
7      int total_rows = 0;  
8  
9      printf("Height? ");  
10     scanf("%d", &total_rows);  
11  
12     print_triangle(total_rows);  
13  
14     return 0;  
15 }  
16  
17 void print_triangle(int total_rows)  
18 {  
19     for (int i = 0; i < total_rows; i++) // total triangle height  
20     {  
21         for (int s = 0; s <= total_rows - i; s++) // left spaces  
22         {  
23             printf(" ");  
24         }  
25  
26         printf("/"); // "/" symbol to left of triangle shape  
27  
28         for (int b = 0; b <= (2 * i - 1); b++) // triangle body  
29         {  
30             printf("#");  
31         }  
32  
33         printf("\\");  
34         printf("\n");  
35     }  
36 }
```

#### Output/s / Testing

Height? 5  


Height? 1  


Height? 13  


## LAB 7: 7x05a – Vector Length

28/04/2022 –Thursday, mid-semester break

### Program aim

Ask user for the x and y coordinates for a 2-tuple vector. Output the vector in the vector-notation-style (call the **print\_vector** function) and compute the magnitude of the vector (call the **compute\_magnitude** function) using the formula:  $|v| = \sqrt{v_x^2 + v_y^2}$ , and print the magnitude (call the **print\_magnitude** function)

### Step-by-step plan

<b>main</b> function:	<b>print_vector</b> function:	<b>compute_magnitude</b> function:	<b>print_magnitude</b> function
1. create float variables: vx and vy 2. PRINT "x? " and READ vx 3. PRINT "y? " and READ vy 4. CALL <b>print_vector(vx, vy)</b> 5. CALL <b>compute_magnitude(vx, vy)</b> 6. CALL <b>print_magnitude(vx, vy)</b>	1. DEFINE <b>print_vector</b> with <b>vx</b> (float) and <b>vy</b> (float) as parameters 2. PRINT vector in vector-notation-style (from exercise example)	1. DEFINE <b>compute_magnitude</b> with <b>vx</b> (float) and <b>vy</b> (float) as parameters 2. From <b>math.h</b> , COMPUTE the return value using <b>sqrt()</b> and <b>powf()</b>	1. DEFINE <b>print_magnitude</b> with <b>vx</b> (float) and <b>vy</b> (float) as parameters 2. PRINT the magnitude in the format shown in the exercise

### Implementation

#### Source Code (lines 2-39)

```

2  |#include <math.h>
3
4  void print_vector(float vx, float vy);
5  float compute_magnitude(float vx, float vy);
6  void print_magnitude(float vx, float vy);
7
8  int main(void)
9  {
10    float vx = 0.0f;
11    float vy = 0.0f;
12
13    printf("x? ");
14    scanf("%f", &vx);
15
16    printf("y? ");
17    scanf("%f", &vy);
18
19    print_vector(vx, vy);
20    compute_magnitude(vx, vy);
21    print_magnitude(vx, vy);
22
23    return 0;
24  }
25
26 void print_vector(float vx, float vy)
27 {
28   printf("\nVector = < %.3f, %.3f >", vx, vy);
29 }
30
31 float compute_magnitude(float vx, float vy)
32 {
33   // COMPUTE |v| = sqrt(vx^2 + vy^2)
34   return sqrtf(powf(vx, 2) + powf(vy, 2));
35 }
36
37 void print_magnitude(float vx, float vy)
38 {
39   printf("\n\n< %.3f, %.3f >| = %.3f", vx, vy, compute_magnitude(vx, vy));
40 }
```

#### Output/s / Testing

```
x? 5
y? 4

Vector = < 5.000, 4.000 >

|< 5.000, 4.000 >| = 6.403
```

```
x? 10
y? 17

Vector = < 10.000, 17.000 >

|< 10.000, 17.000 >| = 19.723
```

```
x? 1
y? 2

Vector = < 1.000, 2.000 >

|< 1.000, 2.000 >| = 2.236
```

```
x? 100
y? 59

Vector = < 100.000, 59.000 >

|< 100.000, 59.000 >| = 116.108
```

## LAB 7: 7x06a - Cube Function

28/04/2022 –Thursday, mid-semester break

Program aim		
Input-process-output plan		
main function		
Input	Process	Output
Integer number	Set <code>base_value</code> to inputted integer number	<code>base_value</code> variable
calculate_cube function		
Input	Process	Output
base_value	COMPUTE <code>base_value * base_value * base_value</code>	return <code>base_value squared</code>
Implementation		
Source Code		Output/s / Testing
<pre> 1  #include &lt;stdio.h&gt; 2 3  int calculate_cube(int base_value) 4  { 5      return base_value * base_value * base_value; 6  } 7 8  int main(void) 9  { 10     int base_value = 0; 11 12     printf("What number do you want cubed? "); 13     scanf("%d", &amp;base_value); 14 15     printf("%d cubed is %d", base_value, calculate_cube(base_value)); 16 17     return 0; 18 }</pre>		<p>What number do you want cubed? 3 3 cubed is 27</p> <p>What number do you want cubed? 5 5 cubed is 125</p> <p>What number do you want cubed? -2 -2 cubed is -8</p> <p>What number do you want cubed? 77 77 cubed is 456533</p> <p>What number do you want cubed? 6 6 cubed is 216</p> <p>What number do you want cubed? 400 400 cubed is 64000000</p>
Lessons learned		
<p>I learnt how to call a function in <code>main</code> without the use of function prototypes/declaration.</p> <p>REMEMBER: I don't have to use prototypes if functions are defined before the <code>main</code> function. It's better to not use function prototypes when there are very few functions to call in <code>main</code></p> <p>However, it is better to use prototypes as it makes my programs more modular</p>		

LAB 7: 7x07a - Dinner Bill

28/04/2022 –Thursday, mid-semester break

## Program aim

Given the following source code, write a dinner bill calculator program.

## Given Source Code (from exercise)

```
#include <stdio.h>

// TODO: Declare compute_bill

// TODO: Declare print_bill

int main(void)
{
    float starter_price = 0.0f;
    float main_price = 0.0f;
    float dessert_price = 0.0f;

    printf("Starter price? ");
    scanf("%f", &starter_price);

    printf("Main price? ");
    scanf("%f", &main_price);

    printf("Dessert price? ");
    scanf("%f", &dessert_price);

    printf("\n");

    // TODO: Call print_bill

    return 0;
}

// TODO: Define compute_bill

// TODO: Define print_bill
```

## New Source code

```
1  #include <stdio.h>
2
3  // TODO: Declare compute_bill
4  float compute_bill(float starter_price, float main_price, float dessert_price);
5
6  // TODO: Declare print_bill
7  void print_bill(float starter_price, float main_price, float dessert_price);
8
9  int main(void)
10 {
11     float starter_price = 0.0f;
12     float main_price = 0.0f;
13     float dessert_price = 0.0f;
14
15     printf("Starter price? ");
16     scanf("%f", &starter_price);
17
18     printf("Main price? ");
19     scanf("%f", &main_price);
20
21     printf("Dessert price? ");
22     scanf("%f", &dessert_price);
23
24     printf("\n");
25
26     // TODO: Call print_bill
27     print_bill(starter_price, main_price, dessert_price);
28
29     return 0;
30 }

31
32 // TODO: Define compute_bill
33 float compute_bill(float starter_price, float main_price, float dessert_price)
34 {
35     return starter_price + main_price + dessert_price;
36 }
37
38 // TODO: Define print_bill
39 void print_bill(float starter_price, float main_price, float dessert_price)
40 {
41     printf("Starter: $%.2f \n", starter_price);
42     printf("Main: $%.2f \n", main_price);
43     printf("Dessert: $%.2f \n", dessert_price);
44     printf("----- \n");
45     printf("Total: $%.2f \n", compute_bill(starter_price, main_price, dessert_price));
46 }
```

## Output/s / Testing

Starter price? 5  
Main price? 7  
Dessert price? 3  
  
Starter: \$5.00  
Main: \$7.00  
Dessert: \$3.00  
-----  
Total: \$15.00

Starter price? 10.35  
Main price? 45.80  
Dessert price? 9.99  
  
Starter: \$10.35  
Main: \$45.80  
Dessert: \$9.99  
-----  
Total: \$66.14

Starter price? 66.60  
Main price? 80.88  
Dessert price? 55.20  
  
Starter: \$66.60  
Main: \$80.88  
Dessert: \$55.20  
-----  
Total: \$202.68

Starter price? 0.08  
Main price? 0.5  
Dessert price? 0.02  
  
Starter: \$0.08  
Main: \$0.50  
Dessert: \$0.02  
-----  
Total: \$0.60

## Lessons Learned

To increase modularity, I need to avoid putting '\n' newlines in `printf` statements in functions outside `main`. (like in the given source code) Instead, I need to create a separate `printf` statement dedicated to a newline in `main`, or a separate function for creating newlines outside `main`.

LAB 7: 7x08a - Is Vowel

28/04/2022 –Thursday, mid-semester break

## Program aim

Detect if the user has or hasn't entered a vowel using a function named **is\_vowel** that accepts **char** input to either return a value of 1 if the input is a vowel, or a value of 0 if the input isn't a vowel.

## Step-by-step plan

is_vowel function	main function
<ol style="list-style-type: none"> <li>1. Define <b>is_vowel</b> as function that accepts char input.</li> <li>2. Create an <b>if</b> statement with a series of OR logical operators to check for lowercase and uppercase vowel input</li> <li>3. IF <b>get_char</b> is a vowel, <b>return 1</b></li> <li>4. ELSE <b>get_char</b> is not a vowel, <b>return 0</b></li> </ol>	<ol style="list-style-type: none"> <li>1. Create char variable <b>get_char</b></li> <li>2. PRINT "&gt;" and READ for <b>get_char</b></li> <li>3. PRINT the return value of the <b>is_vowel</b> function</li> </ol>

## Implementation

Source Code	Output/s / Testing																												
<pre> 1 #include &lt;stdio.h&gt; 2 3 char is_vowel(char get_char) 4 { 5     if (get_char == 'a'    get_char == 'e'    get_char == 'i'    get_char == 'o'    get_char == 'u'    6         get_char == 'A'    get_char == 'E'    get_char == 'I'    get_char == 'O'    get_char == 'U') 7         return 1; 8     else 9         return 0; 10 } 11 12 int main(void) 13 { 14     char get_char = '\0'; 15 16     printf("&gt; "); 17     scanf(" %c", &amp;get_char); 18 19     printf("%d", is_vowel(get_char)); 20 21     return 0; 22 }</pre>	<p>Uppercase vowel input testing</p> <table border="1"> <tr> <td>&gt; A</td> <td>&gt; E</td> <td>&gt; I</td> <td>&gt; O</td> <td>&gt; U</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>1</td> <td>1</td> </tr> </table> <p>Lowercase vowel input testing</p> <table border="1"> <tr> <td>&gt; a</td> <td>&gt; e</td> <td>&gt; i</td> <td>&gt; o</td> <td>&gt; u</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>1</td> <td>1</td> </tr> </table> <p>Testing non-vowels</p> <table border="1"> <tr> <td>&gt; 9</td> <td>&gt; \$</td> <td>&gt; string</td> <td>&gt; c</td> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> </table>	> A	> E	> I	> O	> U	1	1	1	1	1	> a	> e	> i	> o	> u	1	1	1	1	1	> 9	> \$	> string	> c	0	0	0	0
> A	> E	> I	> O	> U																									
1	1	1	1	1																									
> a	> e	> i	> o	> u																									
1	1	1	1	1																									
> 9	> \$	> string	> c																										
0	0	0	0																										

## LAB 7: 7x09a - Draw ASCII Box

28/04/2022 –Thursday, mid-semester break

### Program aim

Write a function named **draw\_ascii\_box** that outputs an ASCII art box. It must take 5 parameters: **char** for horizontal sides; **char** for vertical sides; **char** for the box corners; **int** for width; and **int** for height. Define 2 helper functions to help structure the algorithm.

### Implementation

main function plan	Source Code (lines 1-34)
<ol style="list-style-type: none"> <li>1. Create function prototypes: <b>draw_ascii_box</b> for printing the ASCII box and 2 other helper functions: printing the top and bottom of the ASCII box and another helper function for printing the body of the ASCII box.</li> <li>2. Define 3 char variables (for box characters): <b>horizontal_sides</b>, <b>vertical_sides</b>, and <b>corners</b>. Then define 2 int functions: <b>width</b> and <b>height</b>.</li> <li>3. Ask and read user input for box width</li> <li>4. Ask and read user input for box height</li> <li>5. Ask and read user input for horizontal side character</li> <li>6. Ask and read user input for vertical side character</li> <li>7. CALL <b>draw_ascii_box</b> function with <b>horizontal_sides</b>, <b>vertical_sides</b>, <b>corners</b>, <b>width</b>, <b>height</b> as parameters.</li> </ol>	<pre style="font-family: monospace; font-size: 0.9em; margin: 0;"> 1  #include &lt;stdio.h&gt; 2 3  void draw_ascii_box(char horizontal_sides, char vertical_sides, char corners, int width, int height); 4  void print_top_bottom(char horizontal_sides, char corners, char width); 5  void print_body(char vertical_sides, char width); 6 7  int main(void) 8  { 9      char horizontal_sides = '\0'; 10     char vertical_sides = '\0'; 11     char corners = '\0'; 12     int width = 0; 13     int height = 0; 14 15     printf("ASCII box width? "); 16     scanf(" %d", &amp;width); 17 18     printf("ASCII box height? "); 19     scanf(" %d", &amp;height); 20 21     printf("Character for corners? "); 22     scanf(" %c", &amp;corners); 23 24     printf("Character for horizontal sides? "); 25     scanf(" %c", &amp;horizontal_sides); 26 27     printf("Character for vertical sides? "); 28     scanf(" %c", &amp;vertical_sides); 29 30     draw_ascii_box(horizontal_sides, vertical_sides, corners, width, height); 31 32     return 0; 33 } 34 </pre>

**LAB 7: 7x09a - Draw ASCII Box (continued)**

28/04/2022 –Thursday, mid-semester break

print_top_bottom plan	Source Code (lines 35-45)
<ol style="list-style-type: none"> <li>1. Define <code>print_top_bottom</code> function with <code>horizontal_sides</code>, <code>corners</code>, and <code>width</code> as parameters.</li> <li>2. Print left corner character</li> <li>3. Create for loop, set <code>i</code> to 2 to subtract from corner characters, post-increment <code>i</code>, loop printing <code>horizontal_sides</code> character until it is at <code>width</code> length. ENDFOR</li> <li>4. Print right corner character</li> <li>5. Print newline</li> </ol>	<pre> 35 void print_top_bottom(char horizontal_sides, char corners, char width) 36 { 37     printf("%c", corners); 38     for (int i = 2; i &lt; width; i++) 39     { 40         printf("%c", horizontal_sides); 41     } 42     printf("%c", corners); 43     printf("\n"); 44 } 45 </pre>
print_body plan	Source Code (lines 46-56)
<ol style="list-style-type: none"> <li>1. Define <code>print_body</code> function with <code>vertical_sides</code> and <code>width</code> as parameters.</li> <li>2. Print left <code>vertical_sides</code> character</li> <li>3. Create for loop, set <code>i</code> to 2 to subtract from <code>vertical_sides</code> characters, post-increment <code>i</code>, loop printing space character until it is at <code>width</code> length. ENDFOR</li> <li>4. Print right <code>vertical_sides</code> character</li> <li>5. Print newline</li> </ol>	<pre> 46 void print_body(char vertical_sides, char width) 47 { 48     printf("%c", vertical_sides); 49     for (int i = 2; i &lt; width; i++) 50     { 51         printf(" "); 52     } 53     printf("%c", vertical_sides); 54     printf("\n"); 55 } 56 </pre>
draw_ascii_box plan	Source Code (lines 57-65)
<ol style="list-style-type: none"> <li>1. Define <code>draw_ascii_box</code> function with <code>horizontal_sides</code>, <code>vertical_sides</code>, <code>corners</code>, <code>width</code>, and <code>height</code> as parameters.</li> <li>2. CALL <code>print_top_bottom</code> function</li> <li>3. Create for loop, set <code>i</code> to 2 to subtract from <code>print_top_bottom</code> output, post-increment <code>i</code>, loop printing <code>print_body</code> function until it is at <code>height</code> length. ENDFOR</li> <li>4. CALL <code>print_top_bottom</code> function</li> </ol>	<pre> 57 void draw_ascii_box(char horizontal_sides, char vertical_sides, char corners, int width, int height) 58 { 59     print_top_bottom(horizontal_sides, corners, width); 60     for (int i = 2; i &lt; height; i++) 61     { 62         print_body(vertical_sides, width); 63     } 64     print_top_bottom(horizontal_sides, corners, width); 65 } </pre>

**LAB 7: 7x09a - Draw ASCII Box (continued)**

28/04/2022 –Thursday, mid-semester break

**Output/s / Testing**

ASCII box width? 5 ASCII box height? 5 Character for corners? X Character for horizontal sides? v Character for vertical sides? i XvvvX i i i i i i XvvvX	ASCII box width? 10 ASCII box height? 6 Character for corners? O Character for horizontal sides? = Character for vertical sides?    O=====O   O=====O	ASCII box width? 4 ASCII box height? 8 Character for corners? C Character for horizontal sides? H Character for vertical sides? V CHHC V V V V V V V V V V V V CHHC
ASCII box width? 4 ASCII box height? 3 Character for corners? @ Character for horizontal sides? * Character for vertical sides? ^ @**@ ^ @**@	ASCII box width? 7 ASCII box height? 4 Character for corners? X Character for horizontal sides? x Character for vertical sides? x XxxxxxX x x x x XxxxxxX	ASCII box width? 5 ASCII box height? 4 Character for corners? . Character for horizontal sides? . Character for vertical sides? . ..... . . . . . .

**Lessons Learned**

I learnt how to create an ASCII shape modularly through the use of functions that serve specific purposes.

Helper functions are useful for simplifying other functions, like the `print_top_bottom` and `print_body` function for simplifying the loop of the `draw_ascii_box` function.

**LAB 7: 7x10a - Maths Addition Quiz**

29/04/2022 –Friday, mid-semester break

Program aim	
Build a simple maths addition quiz program from the given source code.	
Implementation	
Given Source Code (from exercise)	New Source code (lines 14-57)
<pre>// COMP500/ENSE501 Modular Math Quiz // Student ID: #include &lt;stdio.h&gt; #include &lt;time.h&gt; #include &lt;stdlib.h&gt;  int check_answer(int left_op, int right_op, int user_answer); int get_random_number(void); int get_user_input(void); void print_prompt(void); void print_question(int left_op, int right_op); void run_quiz(int num_questions); void seed_prng(void);  void seed_prng(void) {     srand(time(0)); }  int main(void) {     // Part a:      // Part g:     return 0; }  // Part b:  // Part c:  // Part d:</pre>	<pre>14   void clear_scanf_buffer(void); 15   16   void seed_prng(void) 17   { 18       srand(time(0)); 19   } 20   21   int main(void) 22   { 23       // Part a: 24       seed_prng(); 25   26       // Part g: 27       int rounds = 0; 28   29       printf("Welcome to the Maths Addition Quiz! \n"); 30       printf("How many rounds? \n"); 31   32       print_prompt(); 33       scanf("%d", &amp;rounds); 34   35       run_quiz(rounds); 36   37       return 0; 38   }  39    40   // Part b: 41   void print_prompt(void) 42   { 43       printf("&gt; "); 44   } 45   46   // Part c: 47   int get_random_number(void) 48   { 49       return rand() % 10 + 1; 50   } 51   52   // Part d: 53   void print_question(int left_op, int right_op) 54   { 55       printf("%d + %d = ? \n", left_op, right_op); 56   } 57  </pre>

**LAB 7: 7x10a - Maths Addition Quiz (Continued)****29/04/2022 –Friday, mid-semester break****Implementation****Given Source Code (from exercise)****// Part e:****// Part f:****// Part h:****New Source code (lines 58-131)**

```

58 // Part e:
59 int check_answer(int left_op, int right_op, int user_answer)
60 {
61     if (left_op + right_op == user_answer)
62     {
63         printf("Correct! \n");
64         return 1;
65     }
66     else
67     {
68         printf("Incorrect! \n");
69         return 0;
70     }
71 }
72
73 // additional function for clearing scanf buffer
74 void clear_scanf_buffer(void)
75 {
76     char ignore = 0;
77
78     while (ignore != '\n')
79     {
80         scanf("%c", &ignore);
81     }
82 }
83
84 // Part f:
85 int get_user_input(void)
86 {
87     int user_input = 0;
88     int is_int = 0;
89
90     while (is_int == 0)
91     {
92         print_prompt();
93         is_int = scanf("%d", &user_input);
94
95         if (is_int == 0)
96         {
97             printf("Try again, must be a whole number! \n");
98             clear_scanf_buffer();
99         }
100    }
101    clear_scanf_buffer();
102
103    return user_input;
104 }
105
106 }
107
108 // Part h:
109 void run_quiz(int num_questions)
110 {
111     printf("The %d question quiz begins! \n", num_questions);
112     int total_correct = 0;
113
114     for (int q = 1; q <= num_questions; q++)
115     {
116         int number1 = get_random_number();
117         int number2 = get_random_number();
118
119         print_question(number1, number2);
120         int user_answer = get_user_input();
121
122         if (check_answer(number1, number2, user_answer) == 1)
123         {
124             total_correct++;
125         }
126     }
127
128     printf("Result: %d ", total_correct);
129     printf("out of %d correct!", num_questions);
130     printf("\n");
131 }
```

**LAB 7: 7x10a - Maths Addition Quiz (Continued)****29/04/2022 –Friday, mid-semester break****Output/s / Testing****Half-correct**

```
Welcome to the Maths Addition Quiz!
How many rounds?
> 5
The 5 question quiz begins!
9 + 5 = ?
> 9
Incorrect!
9 + 4 = ?
> 13
Correct!
5 + 3 = ?
> 8
Correct!
5 + 8 = ?
> 13
Correct!
2 + 7 = ?
> 5
Incorrect!
Result: 3 out of 5 correct!
```

**All correct & invalid input**

```
Welcome to the Maths Addition Quiz!
How many rounds?
> 3
The 3 question quiz begins!
2 + 6 = ?
> eight
Try again, must be a whole number!
> 8
Correct!
6 + 1 = ?
> ???
Try again, must be a whole number!
> 7
Correct!
9 + 4 = ?
> 13
Correct!
Result: 3 out of 3 correct!
```

**All incorrect**

```
Welcome to the Maths Addition Quiz!
How many rounds?
> 4
The 4 question quiz begins!
10 + 5 = ?
> 5
Incorrect!
7 + 5 = ?
> 5
Incorrect!
4 + 3 = ?
> 5
Incorrect!
9 + 10 = ?
> 5
Incorrect!
Result: 0 out of 4 correct!
```

**Lessons Learned**

I had to make an additional function that wasn't mentioned in the given source code to clear the `scanf` buffer because if I didn't, it would create an infinite loop that repeats: "Try again, must be a whole number!".

This taught me that for future loops that require error/input checking, I will need to clear the buffer / make a function for it to prevent an infinite error output loop.

**REMINDER:** Buffer-clearing functions read/iterates through each character from the input buffer until the end of the input (newline)- causing the buffer to clear.

**Lab 7 Midsem: Canvas Contribution**  
**30/04/2022 –Saturday, mid-semester break**

**Canvas Question**



Hayden Richard-Marsters  
30 Apr 16:46 Last reply 30 Apr 18:23

```
#include <stdio.h>

char get_letter_grade(float score);

void print_report(char first, char last, float score);

int main(void)
{
    char first_name_initial;
    char last_name_initial;
    float score;

    printf("first name initials? ");
    scanf("%c", &first_name_initial);

    printf("last name initials? ");
    scanf(" %c", &last_name_initial);

    printf("assessment score? ");
    scanf(" %f", &score);

    get_letter_grade(score);

    print_report(first_name_initial, last_name_initial, score);

    return 0;
}

char get_letter_grade(float score)
{
    if (80 <= score <= 100)
    {
        return 'A';
    }
    else if (65 <= score <= 80)
    {
        return 'B';
    }
    else if (50 <= score <= 65)
    {
        return 'C';
    }
    else if (0 <= score <= 50)
    {
        return 'D';
    }
    else
    {
        return 'N';
    }
}

void print_report(char first, char last, float score)
{
    printf("%c%c assessment score of %f is a %c grade.", first, last, score, get_letter_grade());
}

Hello, just abit stuck on how to include the char result of the get_letter_grade function in the print_report function.  
thank you
```

[Reply](#) | [2 replies](#)

**Lab 7 Midsem: Canvas Contribution**  
**30/04/2022 –Saturday, mid-semester break**

**My Answer**



Francisca Nel

30 Apr 18:13

:

Hi Hayden,

The char result of the get\_letter\_grade function doesn't output in the printf line for the print\_report function because the get\_letter\_grade needs the parameter 'score' to actually output the char grade. Ex: get\_letter\_grade(score). Hope this fixes the problem!

Some other things to note:

It's good practice to set float variables to 0.0f when defining them, like: float score = 0.0f, and char variables to null, like: first\_name\_initial = '\0'

the first name initial scan (line 73) needs a space before %c

Take a look at the relational operators in the get\_letter\_grade function, they should be formatted like: (80 <= score && score <= 100) instead of (80 <= score <= 100)- but this line still needs fixing

Check if the relational operators are correct because it seems your program only outputs the A grade no matter the input. Try setting the A grade's relational parameters to (score >= 80) as a start

I hope I helped! :)

After writing the feedback to the question, I realised that the relational operators I wrote in my own program (7x03a) of (score >= 80) for A was wrong as a grade with a score over 100 would be invalid, and should be an N grade output.

So I corrected myself in another reply:



Francisca Nel

30 Apr 18:23

:

Francisca Nel

30 Apr 18:13

Hi Hayden, The char result of the get\_letter\_grade function doesn't output in the printf line for the print\_report function because the get\_letter\_grade needs the pa ...

Read more

Oops, I meant try setting A to (score >= 80 && score <= 100), my bad! Ignore the part where I said set it to (score >= 80)

Quote

**Lab 7 Midsem: 7x11a - Treasure Finder****30/04/2022 –Saturday, mid-semester break****Program aim**

Design a program that allows the user to move in a 2-dimensional grid in the directions: N,E,S,W. The player can find hidden treasure at certain coordinates. The world must be limited to a range of 10 to -10 inclusive on each axis.

Input must support both lower and upper case: ex, 'N' and 'n'. I may pick my own 5<sup>th</sup> treasure!

Hidden Treasure	X location	Y location
Pirate's Chest	2	3
Golden Idol	-5	-3
Precious Gemstones	1	-2
Lost Artwork	5	2
Pokemon Card	-3	-3

**Pseudocode**

Create PROTOTYPE functions: `run_game`, `print_hidden_treasure`, `print_boundary`

main function	run_game function
<pre> START main Create variables: x_coord (int), y_coord (int), and coord_input(char)  PRINT Find the hidden treasure! PRINT ----- PRINT press 'q' to quit at any time...  CALL run_game with parameters x_coord, y_coord, coord_input  END main </pre>	<pre> START run_game with parameters: coord_input (char), x_coord (int), y_coord (int)  WHILE coord_input is NOT equal to 'q' (for quit)      CALL print_boundary      PRINT "You are at ((x_coord), (y_coord)).         Move(N/E/S/W)?""     READ coord_input      PRINT newline      IF coord_input is equal to 'n' OR 'N'         IF y_coord is NOT equal to 10             Add 1 to y_coord      ELSE IF coord_input is equal to 's' OR 'S'         IF y_coord is NOT equal to -10             subtract 1 from y_coord      ELSE IF coord_input is equal to 'e' OR 'E'         IF x_coord is NOT equal to 10             add 1 to x_coord      ELSE IF coord_input is equal to 'w' OR 'W'         IF x_coord is NOT equal to -10             subtract 1 from x_coord      CALL print_hidden_treasure  ENDWHILE END run_game </pre>

**Lab 7 Midsem: 7x11a - Treasure Finder (continued)**

30/04/2022 –Saturday, mid-semester break

**Pseudocode****print\_hidden\_treasure function**

```

START print_hidden_treasure with parameters x_coord
(int) and y_coord (int)

(all print in square formatting)

IF x_coord EQUALS 2 AND y_coord EQUALS 3
PRINT "You found a pirate's chest!"

ELSE IF x_coord EQUALS -5 AND y_coord EQUALS -3
PRINT "You found a golden idol!"

ELSE if x_coord EQUALS 1 AND y_coord EQUALS -2
PRINT "You found precious gemstones!"

ELSE if x_coord EQUALS 5 AND y_coord EQUALS 2
PRINT "You found a lost artwork!"

ELSE if x_coord EQUALS -3 AND y_coord EQUALS -3
PRINT "You found a Pokemon card!"

```

**print\_boundary function**

```

START print_boundary with parameters x_coord
(int) and y_coord (int)

IF x_coord EQUALS 10 OR x_coord EQUALS -10 OR
y_coord EQUALS 10 OR y_coord EQUALS -10

(all print in square formatting)

PRINT "You have reached the edge of the
world!"

```

**Implementation****Prototypes and main function (lines 1-23)**

```

1 #include <stdio.h>
2
3 void run_game(char coord_input, int x_coord, int y_coord);
4 void print_hidden_treasure(int x_coord, int y_coord);
5 void print_boundary(int x_coord, int y_coord);
6
7 int main(void)
8 {
9     int x_coord = 0;
10    int y_coord = 0;
11    char coord_input = '\0';
12
13    printf("Find the hidden treasure! \n");
14    printf("----- \n\n");
15
16    printf("Press 'q' to quit at anytime... \n\n");
17
18    run_game(coord_input, x_coord, y_coord);
19
20    printf("Goodbye!\n");
21    return 0;
22 }
23

```

**Run\_game function (lines 24-66)**

```

24 void run_game(char coord_input, int x_coord, int y_coord)
25 {
26     while (coord_input != 'q')
27     {
28         print_boundary(x_coord, y_coord);
29
30         printf("You are at (%d, %d). Move (N/E/S/W)? ", x_coord, y_coord);
31         scanf(" %c", &coord_input);
32
33         printf("\n");
34
35         if (coord_input == 'n' || coord_input == 'N')
36         {
37             if (y_coord != 10)
38             {
39                 y_coord++;
40             }
41         }
42         else if (coord_input == 's' || coord_input == 'S')
43         {
44             if (y_coord != -10)
45             {
46                 y_coord--;
47             }
48         }
49         else if (coord_input == 'e' || coord_input == 'E')
50         {
51             if (x_coord != 10)
52             {
53                 x_coord++;
54             }
55         }
56         else if (coord_input == 'w' || coord_input == 'W')
57         {
58             if (x_coord != -10)
59             {
60                 x_coord--;
61             }
62         }
63     }
64 }
65

```

**Lab 7 Midsem: 7x11a - Treasure Finder (continued)**

30/04/2022 –Saturday, mid-semester break

**Implementation****print\_hidden\_treasure  
function (lines 67-100)**

```

67 void print_hidden_treasure(int x_coord, int y_coord)
68 {
69     if (x_coord == 2 && y_coord == 3)
70     {
71         printf("*****\n");
72         printf("* You found a pirate's chest! *\n");
73         printf("*****\n");
74     }
75     else if (x_coord == -5 && y_coord == -3)
76     {
77         printf("*****\n");
78         printf("* You found a golden idol! *\n");
79         printf("*****\n");
80     }
81     else if (x_coord == 1 && y_coord == -2)
82     {
83         printf("*****\n");
84         printf("* You found precious gemstones! *\n");
85         printf("*****\n");
86     }
87     else if (x_coord == 5 && y_coord == 2)
88     {
89         printf("*****\n");
90         printf("* You found a lost artwork! *\n");
91         printf("*****\n");
92     }
93     else if (x_coord == -3 && y_coord == -3)
94     {
95         printf("*****\n");
96         printf("* You found a Pokemon card! *\n");
97         printf("*****\n");
98     }
99 }
100

```

**print\_boundary function (lines 101-110)**

```

101 void print_boundary(int x_coord, int y_coord)
102 {
103     if (x_coord == 10 || x_coord == -10 ||
104         y_coord == 10 || y_coord == -10)
105     {
106         printf("+-----+ \n");
107         printf("| You have reached the edge of the world! | \n");
108         printf("+-----+ \n");
109     }
110 }

```

**Output/s and testing****Finding Pirate's Chest**

```

Find the hidden treasure!
-----
Press 'q' to quit at anytime...

You are at (0, 0). Move (N/E/S/W)? e
You are at (1, 0). Move (N/E/S/W)? e
You are at (2, 0). Move (N/E/S/W)? n
You are at (2, 1). Move (N/E/S/W)? n
You are at (2, 2). Move (N/E/S/W)? n
*****
* You found a pirate's chest! *
*****
You are at (2, 3). Move (N/E/S/W)?

```

**Finding precious gemstones**

```

You are at (2, 3). Move (N/E/S/W)? s
You are at (2, 2). Move (N/E/S/W)? s
You are at (2, 1). Move (N/E/S/W)? s
You are at (2, 0). Move (N/E/S/W)? s
You are at (2, -1). Move (N/E/S/W)? s
You are at (2, -2). Move (N/E/S/W)? w
*****
* You found precious gemstones! *
*****
You are at (1, -2). Move (N/E/S/W)?

```

**Finding lost artwork**

```

You are at (1, -2). Move (N/E/S/W)? n
You are at (1, -1). Move (N/E/S/W)? n
You are at (1, 0). Move (N/E/S/W)? n
You are at (1, 1). Move (N/E/S/W)? n
You are at (1, 2). Move (N/E/S/W)? e
You are at (2, 2). Move (N/E/S/W)? e
You are at (3, 2). Move (N/E/S/W)? e
You are at (4, 2). Move (N/E/S/W)? e
*****
* You found a lost artwork! *
*****

```

## Lab 7 Midsem: 7x11a - Treasure Finder (continued)

30/04/2022 –Saturday, mid-semester break

### Output/s and testing

#### Finding Pokemon card

```
You are at (4, -3). Move (N/E/S/W)? w  
You are at (3, -3). Move (N/E/S/W)? w  
You are at (2, -3). Move (N/E/S/W)? w  
You are at (1, -3). Move (N/E/S/W)? w  
You are at (0, -3). Move (N/E/S/W)? w  
You are at (-1, -3). Move (N/E/S/W)? w  
You are at (-2, -3). Move (N/E/S/W)? w  
*****  
* You found a Pokemon card! *  
*****  
You are at (-3, -3). Move (N/E/S/W)?
```

#### Finding golden idol

```
You are at (-2, -3). Move (N/E/S/W)? w  
*****  
* You found a golden idol! *  
*****  
You are at (-4, -3). Move (N/E/S/W)? w  
*****  
* You found a golden idol! *  
*****  
You are at (-5, -3). Move (N/E/S/W)?
```

#### Reaching world edge and quitting

```
You are at (-9, -3). Move (N/E/S/W)? w  
+-----+  
| You have reached the edge of the world! |  
+-----+  
You are at (-10, -3). Move (N/E/S/W)? w  
+-----+  
| You have reached the edge of the world! |  
+-----+  
You are at (-10, -3). Move (N/E/S/W)? s  
+-----+  
| You have reached the edge of the world! |  
+-----+  
You are at (-10, -4). Move (N/E/S/W)? e  
You are at (-9, -4). Move (N/E/S/W)? q  
Goodbye!
```

### Lessons Learned

I struggled to find a way to implement the world edge restriction to the user entering a coordinate that would cause their position to exceed the 10 to -10 limit.

In the end, I learnt how to implement nested ifs to check whether to increment/decrement a coordinate value, thus stopping the coordinate values from increasing/decreasing past the limit and so that the `print_boundary` function continues to print that the user has reached the edge of the world

## Week 7 lecture 020 Notes- Function Bugs, Side Effects, Recursion, Enumeration

WRITTEN 1/05/2022- Sunday, Mid Semester Break

Notes Retrieved from Steffan Hooper (2022)

REMEMBER: declaring/prototypes: state function name/parameters/types without body.

Defining: state function name/parameters/types that match the prototype WITH body

Local variables are contained within the scope brackets {} of a function and are inaccessible outside it.

To save the return value (output) of a function in another function, assign a variable to the function with its parameters: `float answer = add(10.0f, 15.0f, 3.0f);`  
The output of `add` is saved as a local variable in `main` called `answer`

**10** is an argument value that is copied into the 'a' parameter of `float add(float a, float b, float c)`. 15 is copied to the 'b' parameter, and 3 is copied to the 'c' parameter. The arguments are passed into the function by value.

REMEMBER: caller- a function that calls another function, callee- a function that was called, callee- the currently executing function, but not a caller.

When creating functions, think:

What is it's job? (To print something out? To compute something? To get input from the user? A combination of activities?)

What does it need? (Parameters, types of parameters)

What will it output? (What type of data?)

REMEMBER: functions beginning with `void` do not return information, and functions with the parameter (`void`) do not receive information.

### COMMON BUGS

- Calling functions as variables (ex. `function`) instead of passing required arguments (`function(a, b, c)`) would output: **Warning C4047**
- Remember to return the correct type of information: if a function type is specified as `int function(int parameter)`, there must be an int return, ex: `return int_value`
- Remember to save the results returned by a function, saving output from function to `main` for example: `int result = sum(3, 8);` after, the `result` variable holding the `sum` function output can be used throughout `main`
- Remember to state the parameter types when declaring and defining functions!
- Remember to match functions declarations and definitions and their parameters/types, otherwise the error will output: **Warning: C2371**
- Function definitions do NOT have semicolons, only declarations do.

## Week 7 lecture 020 Notes- Function Bugs, Side Effects, Recursion, Enumeration (continued)

WRITTEN 1/05/2022- Sunday, Mid Semester Break

Notes Retrieved from Steffan Hooper (2022)

Program state: keywords

- **Memory** stores information
- As the program runs, values stored in memory are changed to store different information: **Algorithms** calculate information
- When information stored in memory is changed over time, the **state** of the program changes

A program's call stack grows and shrinks. **Stack frames**: Store local variables and parameters and facilitates the return values

Call stack:

- **Pushing**: a stack frame for function added to the top of the Call Stack
- **Popping**: a stack frame removed from the top of the Call Stack

**Side effect**: an extra action a function does (like printing). A pure function has no side effects. A function with side effects is an impure function.

**Recursion bug**: when a function calls itself, but undergoes an infinite loop

**Stack overflow**: when the computer runs out of memory from the call stack growing too large. This can happen during the recursion bug as it causes stack frames to continually be created and pushed onto the call stack. Error output: **Warning: C4717**

**Recursion is a valid programming technique** and is used when the solution needs smaller instances of the same problem- like calculating the factorial of a number:  $5! = 5 \times 4 \times 3 \times 2 \times 1$  (this can be solved using iteration as well)

Recursion:

- There needs to be a **base case**, which is where the function stops calling itself further. It needs to cause the function to return to the caller instead.
- Recursive algorithms can have more than one kind of **base case**
- If the base case is never reached, an **INFINITE LOOP** will happen!

```
int compute_factorial(int n)
{
    if (n >= 1)
    {
        return (n * compute_factorial(n - 1)); → recursion: function calls self
    }
    return 1; → base case: when if is false, return 1
}
```

**Week 7 lecture 020 Notes- Function Bugs, Side Effects, Recursion, Enumeration**  
**(continued)**

**WRITTEN 1/05/2022- Sunday, Mid Semester Break**

Notes Retrieved from Steffan Hooper (2022)

Enumeration: a way to define a new data type.

1. Call `enum` followed by enumerated type name.
2. In {} brackets, List comma-separated possible values the type can hold.
3. End new type declaration with a semicolon.
4. Declare a new variable using the enumerated type

CODE EXAMPLE:

Each constant in `enum` holds an integer value, like the index of an array.

```
enum Day
{
    SUNDAY, → Holds int value 0
    MONDAY, → Holds int value 1
    TUESDAY, → Holds int value 2
    WEDNESDAY, → Holds int value 3
    THURSDAY, → Holds int value 4
    FRIDAY, → Holds int value 5
    SATURDAY → Holds int value 6
};

enum Day today;
```

...

```
enum Day today = THURSDAY; → today assignment to THURSDAY causes today to hold
int value 4
printf("today holds: %d\n", today); → this will print 4
```

Enumerated constants can be compared using selection, can be used with arrays (must have a MAX value by adding an additional element to the enumeration), and can be used with loops (can loop starting from first number to the last number in the enumeration)

Enumerated constants can have a value set to them, for example:

```
...
SUNDAY = 10, → Holds int value 10
MONDAY, → Holds int value 11
TUESDAY = 20, → Holds int value 20
WEDNESDAY, → Holds int value 21
...
```

`enums` must be declared before prototypes and main if it's to be used across the program! If an enumerated type is declared inside a function, the type is only available in the function scope. To use in other functions, pass as parameter: `void print_day(enum Day to_print);` `enums` may also be returned as a value by functions.

## Week 7 lecture 021 Notes- Modularity, Extended ASCII, Random-Match-Three Game

WRITTEN 1/05/2022- Sunday, Mid Semester Break

Notes Retrieved from Steffan Hooper (2022)

REMINDER: Validating input. Modularised reading a whole number example:

```
int get_number_from_keyboard(void) → function takes no input from a caller
{
    int number = 0;
    int scan_result = 0;
    while (!is_input_valid(scan_result)) → loop while is_input_valid detects
                                                scan_result is invalid
    {
        scan_result = scanf("%d", &number); → get input
        check_validity_and_prompt(scan_result); → function that returns error
                                                message and clears input buffer
    }
    return number; → return the int input value
}
```

SPACING ASCII FOR TABLES: using modulus like this: `printf("[%3d] ", int_variable);` will output spaces before the integer value to create spacing until the limit (3) is reached, for example:  
an int value with 1 character will be spaced like [ 1],  
an int value with 2 characters will be spaced like [ 11],  
and an int value with 3 characters will be spaced like [111]

**Unsigned chars** are 1 byte (8 bits). ASCII chars use 7 of the 8 bits.

Extended ASCII use the 8<sup>th</sup> bit in the **unsigned char**. Extended ASCII chars range from 128-255. They are special characters that aren't on the keyboard, and can be used to create interesting ASCII art!

**Sleep()** from `#include <Windows.h>` is a Microsoft windows specific function that pauses the computer for a specified number of milliseconds: `Sleep(1000);` means pause for 1 second. Sleep can be used to delay print and other actions.

REMEMBER HOW TO SEED Pseudo Random Number Generator (PRNG): `srand(time(0));`

Random number generators can have a user-inputted range of random number selection:

```
int get_random(int low, int high) → high and low parameters can be inputted by user
{
    return low + (rand() % (high - low + 1)); → high and low specifies range
}
```

When creating a number to be used in a number that stays the same, remember to use **const** to avoid creating a "magic number" (for example, the fixed limit of a game).

**Code refactoring:** process of restructuring existing source code to improve design, structure, and implementation without changing behaviour and functionality. This includes reducing complexity, improving readability, and reducing unnecessary repetition.

# **WEEK 8**

## Week 8 lecture 022 Notes- Dot Operator, Passing Structures, Data Abstraction

2/05/2022- Monday, week 8

Notes Retrieved from Steffan Hooper (2022)

**Structures:** `struct` keyword- groups several types of information together to create complex data types- with different names and purposes.

**Composition:** combine simple data types into more complex types “has a” relationship- describes ownership. **Members:** what goes into the structure

**Formatting:**

```
Struct newtypename
{
    type member1;
    type member2;
}; → remember semicolon
```

**Cohesion:** how well elements fit together- members inside structs must fit together and make sense- this creates high cohesion: robustness, reliability...

Structs are like blueprints/templates/designs- what parts go into something? For example, to define a `rectangle struct`, it needs a `width` member and a `height` member. `Struct circle` needs `center_x`, `center_y`, and `radius` members.

**Members** can be `enum` types, but remember to declare `enum` first.

You can use a `struct` within a `struct`

`structs` tell the compiler new types of data the programmer wants to store.

**Dot operator:** used to access member of the structure variable, for example:

`fran.age = 18;`

variable instance of the struct named `fran`, `age` is a struct member, assign `age` member to 18

`sprintf`- to print into char array (string print)

`sprintf(example.name, "Fran");`

Call struct member in printf example: `printf("%s", example.name)`

`struct` is a data type in the declaration/definition of functions

structures can be passed by value in functions

**REMINDER:** `sizeof` is used to return the number of bytes in a data type.

The `order` members are declared in a `struct` can affect the overall structure size. member

arrays can be created from structures.

Revision (from lecture topics checklist)

2/05/2022 –Monday, week 8

Checklist			Revision Notes
Lab 1			
Lecture topics in preparation for Lab Tutorial 1:	Confident?	Familiar?	Unknown?
• The basic development cycle: ◦ Edit, Compile, Run, Test ◦ Syntax errors ◦ Logic errors	✓	✓	
• The main entry point	✓		
• <code>printf</code>	✓		
• String literals	✓		
• Escape character sequences	✓		
• Sequence	✓		
• Visual Studio 2017: ◦ Solution Explorer ◦ Editor ◦ Building ◦ Start without Debugger ◦ Compilation errors ◦ Output window ◦ Switching Startup Project	✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓		All good 😊
Lab 2			
Lecture topics in preparation for Lab Tutorial 2:	Confident?	Familiar?	Unknown?
• Algorithms: ◦ Good Practices ◦ The Input-Process-Output Pattern ◦ A Seven Step Approach to Problem Solving	✓ ✓ ✓		
• The C Tool Chain: ◦ Compiler ◦ Linker	✓ ✓	✓	
• Bugs	✓		
• Debugging		✓	
• Coding Standards: Whitespace	✓		
• Visual Studio: Cleaning		✓	
• Visual Studio Debugger: ◦ Breakpoints ◦ Stepping ◦ Watch Window	✓ ✓ ✓		
• Variables: ◦ Types ◦ Local variables ◦ Assignment ◦ Arithmetic: ▪ +, -, *, /, () ▪ Modulus ◦ Pre-increment ◦ Post-increment	✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓		C tool chain reminder: <ul style="list-style-type: none"><li>- <b>Source code editor:</b> the person programming</li><li>- <b>C compiler:</b> reader of instructions and converts into machine code</li><li>- <b>Linker:</b> merges code into a single executable program</li></ul> <b>Visual studio: Cleaning</b> is done through build > clean solution. This deletes IDE (integrated development environment) generated files such as <b>.obj</b> and <b>.exe</b>

Revision (from lecture topics checklist)

2/05/2022 –Monday, week 8

Checklist			Revision Notes			
			Lab 3			
Lecture topics in preparation for Lab Tutorial 3:	Confident?	Familiar?	Unknown?	Lecture topics in preparation for Lab Tutorial 3:	Confident?	
<ul style="list-style-type: none"> <li>• Characters:           <ul style="list-style-type: none"> <li>◦ Character literals, ASCII</li> <li>◦ Escape sequence codes</li> </ul> </li> <li>• printf outputting           <ul style="list-style-type: none"> <li>◦ Character arithmetic</li> </ul> </li> <li>• scanf:           <ul style="list-style-type: none"> <li>◦ Stream buffering</li> <li>◦ Format specifiers</li> <li>◦ Whitespace parsing</li> <li>◦ Reading multiple items</li> </ul> </li> <li>• printf result returned</li> <li>• scanf result returned</li> <li>• Arrays:           <ul style="list-style-type: none"> <li>◦ Declaring</li> <li>◦ Size</li> <li>◦ Assigning elements</li> <li>◦ Reading from elements</li> <li>◦ Initialising</li> </ul> </li> <li>• Visual Studio Debugger:           <ul style="list-style-type: none"> <li>◦ Arrays in the Watch Window</li> <li>◦ Memory Window</li> </ul> </li> <li>• Array bugs:           <ul style="list-style-type: none"> <li>◦ Off by one</li> </ul> </li> </ul>	✓	✓	✓	<ul style="list-style-type: none"> <li>◦ Index out of bounds</li> <li>• Using math.h:           <ul style="list-style-type: none"> <li>◦ powf, sqrtf, fabsf</li> <li>◦ floorf, ceilf, logf, log10f</li> <li>◦ sinf, cosf, tanf</li> </ul> </li> <li>• Random numbers:           <ul style="list-style-type: none"> <li>◦ Seeding: srand with time</li> <li>◦ Using rand()</li> <li>◦ Modulus and rand()</li> </ul> </li> <li>• Character arrays:           <ul style="list-style-type: none"> <li>◦ Declaring</li> <li>◦ Initialising</li> <li>◦ The null character</li> <li>◦ Null terminated C-Strings</li> <li>◦ Printing with %s</li> </ul> </li> <li>• Reading in C-String with scanf</li> <li>• C-String length vs array dimension</li> <li>• Scansets</li> <li>• Using string.h:           <ul style="list-style-type: none"> <li>◦ strlen, strcpy, strcat</li> </ul> </li> <li>• Conversions:           <ul style="list-style-type: none"> <li>◦ atoi, atof, sprintf</li> </ul> </li> </ul>	✓	✓

**Buffer:** data from keyboard that is stored temporarily before program accesses it

**TO READ SERIES OF CHARS (INCLUDING SPACES) ONLY STOPPING AT ENTER-** use scanf with %[^n]

Putting a space before “ %c” trims leading white space in the input buffer from pressing enter. No space = \n left in input buffer

Reading **multiple inputs** are useful for specific formatting of input, like date: `scanf("%d/%d/%d", &day, &month, &year);` INPUT MUST BE: “DD/MM/YY” (all at once, include ‘/’, no enter)

RANDOM NUMBERS REMINDER:

1. `#include <stdlib>` AND `<time.h>`
2. SEED `srand` WITH `time`: `srand(time(0));`
3. GENERATE RANDOM NUM: `(rand() % 6) + 1;` (6 is max)  
MODULUS RANGE IS 0-6, PLUS 1 CHANGES RANGE TO 1-6

HOW TO DO MINIMUM VALUES FOR RANDOM:  
`rand() % (max - min +1)) + min`

PROGRAM: PRACTICING USING math.h FUNCTIONSOURCE CODE

```

1  #include <stdio.h>
2  #include <math.h>
3
4  int main(void)
5  {
6      const float a = 16.0f;
7      const float b = 2.0f;
8      const float c = -16.0f;
9      const float d = 25.55f;
10     const float e = 1000.0f;
11
12     printf("-*- POWF FUNCTION -*-\n"); // powf function (base, power)
13     printf("%f^%f is: %f \n\n", a, b, powf(a, b));
14
15     printf("-*- SQRT FUNCTION -*-\n"); // sqrtf function
16     printf("square root of %f is: %f \n\n", a, sqrtf(a));
17
18     printf("-*- FABSF FUNCTION -*-\n"); // fabsf function (absolute value)
19     printf("|-%f| is: %f \n\n", c, fabsf(c));
20
21     printf("-*- FLOORF FUNCTION -*-\n"); // floorf function (round down)
22     printf("floor value of %.2f is: %f \n\n", d, floorf(d));
23
24     printf("-*- CEILF FUNCTION -*-\n"); // ceilf function (round up)
25     printf("ceiling value of %.2f is: %f \n\n", d, ceilf(d));
26
27     printf("-*- LOGF FUNCTION -*-\n"); // logf function
28     printf("ln(%f) is: %f \n\n", d, logf(d));
29
30     printf("-*- LOG10F FUNCTION -*-\n"); // log10f function
31     printf("log base 10 of (%f) is: %f \n\n", e, log10f(e));
32
33     printf("-*- SINF FUNCTION -*-\n"); // sinf function (sine)
34     printf("sine of %.2f is: %f \n\n", d, sinf(d));
35
36     printf("-*- COSF FUNCTION -*-\n"); // cosf function (cosine)
37     printf("cosine of %.2f is: %f \n\n", d, cosf(d));
38
39     printf("-*- TANF FUNCTION -*-\n"); // tanf function (tangent)
40     printf("tangent of %.2f is: %f \n\n", d, tanf(d));
41
42     return 0;
43 }
```

OUTPUT

```

-**- POWF FUNCTION -*-
16^2 is: 256.000000

-**- SQRT FUNCTION -*-
square root of 16 is: 4.000000

-**- FABSF FUNCTION -*-
|-16| is: 16.000000

-**- FLOORF FUNCTION -*-
floor value of 25.55 is: 25.000000

-**- CEILF FUNCTION -*-
ceiling value of 25.55 is: 26.000000

-**- LOGF FUNCTION -*-
ln(25.55) is: 3.240637

-**- LOG10F FUNCTION -*-
log base 10 of (1000) is: 3.000000

-**- SINF FUNCTION -*-
sine of 25.55 is: 0.405255

-**- COSF FUNCTION -*-
cosine of 25.55 is: 0.914204

-**- TANF FUNCTION -*-
tangent of 25.55 is: 0.443288

```

Revision (from lecture topics checklist)

2/05/2022 –Monday, week 8

**Lab 3 (continued)****Revision Notes**ARRAYS:Declaring: `int array_name[5];` → array size must be a constantInitialising: `int array_name[5] = {2, 4, 6, 8, 10};` → initialise if element data is knownDO `char array_name[] = "string";` if you don't know the char array length/no restriction, but want a c-stringNull-terminated c-string practice:**NOT null-terminated c-string**

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     char my_name[10];
6
7     my_name[0] = 'F';
8     my_name[1] = 'r';
9     my_name[2] = 'a';
10    my_name[3] = 'n';
11    //my_name[4] = '\0';
12
13    printf("%s", my_name);
14
15    return 0;
16 }
```

**NOT null-terminated c-string Output**
**Null-terminated c-string**

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     char my_name[10];
6
7     my_name[0] = 'F';
8     my_name[1] = 'r';
9     my_name[2] = 'a';
10    my_name[3] = 'n';
11    my_name[4] = '\0';
12
13    printf("%s", my_name);
14
15    return 0;
16 }
```

**Null-terminated c-string output**
**Note:** length of c-string is not the same as the size of the char arrayUsing string.h practice:**Source Code:**

```

1 #include <stdio.h>
2 #include <string.h>
3
4 int main(void)
5 {
6     char first_name[] = "Fran";
7     char last_name[] = "Nel";
8     char copy[64];
9     char full_name[100];
10
11     int string_length = strlen(first_name); // strlen function: length
12     strcpy(copy, first_name); // strcpy function: copy (copy, original)
13
14     // CREATING FULL NAME USING STRCAT (concatenate)
15     strcat(full_name, first_name); // copy first_name into full_name array
16     strcat(full_name, " "); // JOIN space character to the end
17     strcat(full_name, last_name); // JOIN last_name string to the end
18
19     printf("*- STRLEN FUNCTION -*\n");
20     printf("length of name_string is: %d \n\n", string_length);
21
22     printf("*- STRCPY FUNCTION -*\n");
23     printf("copied string is: %s \n\n", copy);
24
25     printf("*- STRCAT FUNCTION -*\n");
26     printf("concatenated full name: %s \n\n", full_name);
27
28     return 0;
29 }
```

**Output:**

```

-* - STRLEN FUNCTION -/
length of name_string is: 4

-* - STRCPY FUNCTION -/
copied string is: Fran

-* - STRCAT FUNCTION -/
concatenated full name: Fran Nel

```

Revision (from lecture topics checklist)

2/05/2022 –Monday, week 8

**Lab 3 (continued)****Revision Notes**Array Conversions using stdlib.h practice:**Source Code:**

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void)
5  {
6      char digit[] = "5.5";
7      char print_into[10];
8
9      double double_number = 0.0;
10     int int_number = 0;
11
12     double_number = atof(digit); // atof function: convert char array to double
13     int_number = atoi(digit); // atoi function: convert char array to int
14     sprintf(print_into, "%d", int_number); // sprintf function: print into array
15
16     printf("As ASCII: %s\n\n", digit);
17
18     printf("-*- ATOF FUNCTION -*-\n");
19     printf("Array as double: %f\n\n", double_number);
20
21     printf("-*- ATOI FUNCTION -*-\n");
22     printf("Array as integer: %d\n\n", int_number);
23
24     printf("-*- SPRINTF FUNCTION -*-\n");
25     printf("int printed into char array as char: %s\n\n", print_into);
26
27
28     return 0;
}

```

**Output:**

As ASCII: 5.5

-\* ATOF FUNCTION -\*-

Array as double: 5.500000

-\* ATOI FUNCTION -\*-

Array as integer: 5

-\* SPRINTF FUNCTION -\*-

int printed into char array as char: 5

NOTE: **printf** will take any type of value (int, float, char) and print it into the array as the array type. Like in my practice code: an **int** number is printed into a **char** array, so after, it needs to be printed using **%s** string instead of **%d** int

Revision (from lecture topics checklist)

3/05/2022 –Tuesday, week 8

**Lab 4****Checklist**

Lecture topics in preparation for Lab Tutorial 4:	Confident?	Familiar?	Unknown?
• Coding Standards: o Whitespace ,Layout, Naming	✓	✓	
• Commenting: o C Style o C++ Style		✓	
• Pseudo code: o Good practices o From design to implementation	✓	✓	
• // TODO : comments	✓		
• Debugging with comments	✓		
• Selection: o The if keyword o The else keyword	✓		
• Relational Operators: o ==, !=, >, >=, <, <=	✓	✓	
• Order of operations: o Precedence		✓	
• Comparing C-Strings: o strcmp	✓		
• Coding standards: o Layout	✓	✓	
• Nested selection:	✓		

Lecture topics in preparation for Lab Tutorial 4:	Confident?	Familiar?	Unknown?
o Scope o Inner block o Outer block		✓	
• else if Ladders	✓		
• Logical (Boolean) Operators: o Logical NOT o Logical OR o Logical AND	✓	✓	
• Bugs: o Edge cases o Boundary cases o Accidental assignment		✓	✓

**Practice / notes**

C style commenting: /\*comment\*/

C++ style commenting: //comment

Favour using // comments!

strcmp (compare c-strings) example

#include &lt;string.h&gt;

strcmp(a, "example")

outputs 0 if a and "example" match, so if statement can go: if (strcmp(a, "example") == 0)

Edge refers to &lt; or ≤? Inclusive or exclusive?

**Accidental assignment:** equality relation is "==" , NOT "="

Revision (from lecture topics checklist)

3/05/2022 –Tuesday, week 8

Lab 5

Checklist			Practice / notes		
Lecture topics in preparation for Lab Tutorial 5:	Confident?	Familiar?	Unknown?		
• Common Selection syntax errors and bugs	✓			Null statement: <code>for ( ; ; ) ;</code> → iterate null statement infinitely	
• The Null Statements		✓		Switch:	
• Selection:		✓		<b>Fall through</b> in switch is similar to the <b>OR</b> operator in if statements. You want to fall through multiple cases If you want to accept multiple inputs for one case:	
◦ <code>switch</code>	✓			<code>case 'y' : // Fall Through (REMEMBER COMMENT)</code>	
◦ Fall Though		✓		<code>case 'Y' :</code>	
◦ <code>case</code>		✓		{	
◦ <code>break</code>		✓		<code>printf("User said yes")</code>	
◦ <code>default</code>		✓		}	
• Nested switches		✓		<b>Break-</b> remember to use break; to end case/default	
• Ternary (Conditional) Operator		✓		<b>Default:</b> similar to else statement (any input except in case)	
• Visual Studio: Conditional Breakpoints			✓		
• Repetition:		✓			
◦ The <code>while</code> keyword	✓			<b>Ternary (conditional) operator</b> formatting:	
◦ Validation of User Input		✓		condition ? value_if_true : value_if_false	
◦ The <code>do</code> keyword	✓				
◦ The <code>break</code> keyword		✓		<b>Unrolled sequence vs rolled-up loop:</b>	
◦ The <code>continue</code> keyword	✓			Unrolled sequence refers to repetitive lines to achieve a solution. A rolled-up loop uses a single-line loop for the same solution.	
• Infinite loop	✓				
• Validation of User Input		✓			
• Coding Standards: Loops	✓				
Lecture topics in preparation for Lab Tutorial 5:	Confident?	Familiar?	Unknown?		
• Repetition:	✓			<b>Validation of user input:</b>	
◦ The <code>for</code> keyword	✓			Bad input:	
• <code>for</code> loops and arrays	✓			Integer var represents <b>number of items successfully scanned</b>	
• Unrolled sequence vs rolled-up loop			✓	<code>int scan_result = scanf("%d", &amp;number);</code>	
• <code>for</code> loops condition...		✓		IF <code>scan_result</code> is 0, invalid input. IF <code>scan_result</code> is 1, valid input.	
• Loop bugs		✓		<code>scanf("%*[^\n]");</code>	
• Forever loops	✓			scan everything in the input buffer and throw away the data (clears the input buffer)	

**REVISION - Lab 5: 5x01e – Switch Compute**

3/05/2022 –Tuesday, Week 8

**Program Aim**

Ask user to input an operator (+, -, \*, /, or %) and then two whole numbers. Using a switch, perform the operation specified by the user, and output the resulting computation. Detect when the user tries to divide by zero.

**Implementation****Source code**

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     float left = 0.0f;
6     float right = 0.0f;
7     char op = '\0';
8
9     printf("Operator? ");
10    scanf(" %c", &op);
11
12    printf("Left Operand? ");
13    scanf("%f", &left);
14
15    printf("Right Operand? ");
16    scanf("%f", &right);
17
18    printf("\n");
19
20    printf("Computing %.f %c %.f \n", left, op, right);
21
22    switch (op)
23    {
24        case '+':
25        {
26            printf("result: %.f", left + right);
27            break;
28        }
29        case '-':
30        {
31            printf("result: %.f", left - right);
32            break;
33        }
34        case 'x': // Fall Through
35        case '*':
36        {
37            printf("result: %.f", left * right);
38            break;
39        }
40        case '/':
41        {
42            switch ((int)right)
43            {
44                case 0:
45                {
46                    printf("Unable to compute divide by zero!");
47                }
48                break;
49                default:
50                {
51                    printf("result: %.2f", left / right);
52                }
53                break;
54            }
55        case '%':
56        {
57            printf("result: %d", (int)left % (int)right);
58            break;
59        }
59        default:
60            printf("Invalid operator!");
61            break;
62        }
63    }
64 }
```

**Output****+ operator output**

Operator? +
Left Operand? 5
Right Operand? 5

Computing 5 + 5
result: 10

**- operator output**

Operator? -
Left Operand? 10
Right Operand? 5

Computing 10 - 5
result: 5

**\* operator output**

Operator? \*
Left Operand? 5
Right Operand? 5

Computing 5 \* 5
result: 25

**/ operator output**

Operator? /
Left Operand? 5
Right Operand? 2

Computing 5 / 2
result: 2.50

**'x' fall through for \* operator output**

Operator? x
Left Operand? 5
Right Operand? 5

Computing 5 x 5
result: 25

**/ operator divide by zero output**

Operator? /
Left Operand? 10
Right Operand? 0

Computing 10 / 0
Unable to compute divide by zero!

**% operator output**

Operator? %
Left Operand? 10
Right Operand? 10

Computing 10 % 10
result: 0

**Invalid operator output**

Operator? a
Left Operand? 5
Right Operand? 3

Computing 5 a 3
Invalid operator!

**Lessons learned**

I revised **switch**, **case**, **default**, and **break** using this exercise. I added a fall through **case** to revise fall through.

**REVISION - Lab 5: 5x02d – Alpha or not**

3/05/2022 –Tuesday, Week 8

**Program Aim**

Ask the user to input an ASCII character. Use the conditional operator to compute whether the input is an alphabetic character or not.

**Implementation**

Source code	Output
<pre> 1 #include &lt;stdio.h&gt; 2 3 int main(void) 4 { 5     int scan_result = 0; 6     char input = '\0'; 7 8     printf("Please enter an ASCII character: "); 9     scan_result = scanf(" %c", &amp;input); 10 11    printf("%c", input); 12    if (input &gt;= 'a' &amp;&amp; input &lt;= 'z')    (input &gt;= 'A' &amp;&amp; input &lt;= 'Z') 13        ? printf(" is alphabetic") 14        : printf(" is not alphabetic"); 15 16    return 0; 17 }</pre>	Non-alphabetic output Please enter an ASCII character: 5 5 is not alphabetic
	Uppercase alphabetic output Please enter an ASCII character: Z Z is alphabetic
	Lowercase alphabetic output Please enter an ASCII character: g g is alphabetic

**Lessons learned**

I revised conditional operator using this exercise. I learnt that conditional operators do not need to be within a print, and the ‘true’ : ‘false’ outputs can be formatted in separate lines.

## Week 8 lecture 023 Notes- Pointers, Pass by Reference

4/05/2022- Wednesday, week 8

Notes Retrieved from Steffan Hooper (2022)

%x prints base-16, %X prints uppercase base-16

Addresses: REMEMBER: & gets address (which is why it is used for **scanf** scanning into)

Print address: **printf("%p", &data);**

What is int \*?

**Pointers:** a variable that stores the address of something ('points' to data)- record location of data by storing address in a pointer.

**int\* address = &input;** - stores address of an integer variable.

**scanf("%d", address);** - 'address' is pointing to the input variable, thus & isn't used

Declare a pointer: **int\* address;** Pointers can be declared for char, float, double...

Dereference address: **\*address** get data at location in memory

**Null pointer:** a pointer that holds value of zero (no data stored)

**Wild pointer:** pointer declared, but not initialised. When declaring pointer without knowing address to save it to, set the pointer to null. **int\* address = 0;**

**Dangling pointer:** a pointer once stored in valid address but has become invalid. Happens when calling pointers from functions out of scope (local variable).

**Pass by reference:** pass local data to other variables using parameters/arguments. Parameters like this: **good\_swap(int\* a, int\* b)** and operate with address data instead of variable data.

Pass by reference using address enables multiple returns of variables, without using **return**.

**REVISION - Lab 5: 5x02e – Digit or not**

4/05/2022 –Wednesday, Week 8

**Program Aim**

Ask user to input an ASCII character. Use the conditional operator to compute whether the input is a digit character or not.

**Implementation****Source code****Output**

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void)
5  {
6      char input = '\0';
7
8      printf("Please enter an ASCII character: ");
9      scanf(" %c", &input);
10
11     if (input >= '0' && input <= '9') // range of digits
12         printf("%c is a digit", input)
13     else
14         printf("%c is not a digit", input);
15
16     return 0;
17 }
```

Is digit output

```
Please enter an ASCII character: 5
5 is a digit
```

Not a digit output

```
Please enter an ASCII character: a
a is not a digit
```

Not a digit output (special character)

```
Please enter an ASCII character: &
& is not a digit
```

**Lessons learned**

I attempted to use validate `int` input, but it would validate and print `int` input, but if the input is a char it wouldn't print the character. I found that all I had to do was check for the range of digits (0-9) as chars- similar to `5x02d` for the alphabet

**REVISION - Lab 5: 5x03e – Loop random**

4/05/2022 –Wednesday, Week 8

**Program Aim**

Using a loop, generate five random numbers, between 3 and 7 inclusive. Each iteration of the loop must generate one random number.

**Implementation****Source code****Output**

```

1  #include <stdio.h>
2  #include <time.h>
3  #include <stdlib.h>
4
5  int main(void)
6  {
7      srand(time(0));
8      int rand_num = 0;
9      int i = 1;
10
11     while (i <= 5)
12     {
13         rand_num = rand() % (7 - 3 + 1) + 3;
14         printf("Iteration %d: Random number is %d \n", i, rand_num);
15         i++;
16     }
17     return 0;
18 }
```

```
Iteration 1: Random number is 3
Iteration 2: Random number is 5
Iteration 3: Random number is 6
Iteration 4: Random number is 7
Iteration 5: Random number is 6
```

```
Iteration 1: Random number is 6
Iteration 2: Random number is 5
Iteration 3: Random number is 4
Iteration 4: Random number is 4
Iteration 5: Random number is 7
```

```
Iteration 1: Random number is 4
Iteration 2: Random number is 7
Iteration 3: Random number is 5
Iteration 4: Random number is 5
Iteration 5: Random number is 3
```

**Lessons learned**

REMEMBER HOW TO DO MIN VALUES IN RANDOM: `rand() % (max - min +1) + min`

**REVISION - Lab 5: 5x04e – Get Even****4/05/2022 –Wednesday, Week 8****Program Aim**

Ask user to input an even number. Check that the number input by the user is even, when the input was not an even number, the program must ask the user to input an even number again.

**Implementation****Source code****Output**

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     int input = 0;
6
7     printf("Input an even number: ");
8     scanf("%d", &input);
9
10    while (input % 2 != 0)
11    {
12        printf("Input an even number: ");
13        scanf("%d", &input);
14
15        if (input % 2 == 0)
16            printf("Well done, that was an even number.");
17    }
18
19    return 0;
20 }
```

```

Input an even number: 33
Input an even number: 55
Input an even number: 35
Input an even number: 3
Input an even number: 6
Well done, that was an even number.

Input an even number: 3
Input an even number: 5
Input an even number: 7
Input an even number: 9
Input an even number: 11
Input an even number: 12
Well done, that was an even number.
```

**Lessons learned**

REMEMBER: use modulus % to check for remainders

**REVISION - Lab 5: 5x05e – Begin, end, step**

4/05/2022 –Wednesday, Week 8

**Program Aim**

Ask user for three number inputs – “begin at”, “end at”, and “step size”. Use a loop to print from the “begin at” number, to the “end at” number, increasing by the “step size” each iteration. The list of numbers printed must be separated by a comma and a single space.

**Implementation**

Source code	Output
<pre> 7   int step_size = 0; 8 9   printf("Begin at? "); 10  scanf("%d", &amp;begin); 11 12  printf("End at? "); 13  scanf("%d", &amp;end); 14 15  printf("Step size? "); 16  scanf("%d", &amp;step_size); 17 18  for (int i = begin; i &lt;= end; i = i + step_size ) 19  { 20      printf("%d", i); 21      if (i &lt;= (end - step_size)) 22      { 23          printf(", "); 24      } 25  } 26 27  return 0; 28 }</pre>	<pre> Begin at? 5 End at? 100 Step size? 10 5, 15, 25, 35, 45, 55, 65, 75, 85, 95</pre>
	<pre> Begin at? 1 End at? 10 Step size? 1 1, 2, 3, 4, 5, 6, 7, 8, 9, 10</pre>
	<pre> Begin at? 1 End at? 100 Step size? 6 1, 7, 13, 19, 25, 31, 37, 43, 49, 55, 61, 67, 73, 79, 85, 91, 97</pre>

**Lessons learned**

I used this exercise to revise for loops. REMEMBER: don't use a variable at the beginning of the for loop like:

**for (begin; begin <= end; begin = begin + step\_size)**, DO THIS INSTEAD:

**for (int i = begin; i <= end; i = i + step\_size)**

**REVISION - Lab 5: 5x05d – Times Tables**

4/05/2022 –Wednesday, Week 8

**Program Aim**

Prompt the user with the > symbol, and then wait for a whole number input. Using a loop, output the multiplication table, from 0 to 12, based upon the whole number input by the user.

**Implementation**

Source code	Output	
> 5	> 2	> 12
5 x 0 = 0	2 x 0 = 0	12 x 0 = 0
5 x 1 = 5	2 x 1 = 2	12 x 1 = 12
5 x 2 = 10	2 x 2 = 4	12 x 2 = 24
5 x 3 = 15	2 x 3 = 6	12 x 3 = 36
5 x 4 = 20	2 x 4 = 8	12 x 4 = 48
5 x 5 = 25	2 x 5 = 10	12 x 5 = 60
5 x 6 = 30	2 x 6 = 12	12 x 6 = 72
5 x 7 = 35	2 x 7 = 14	12 x 7 = 84
5 x 8 = 40	2 x 8 = 16	12 x 8 = 96
5 x 9 = 45	2 x 9 = 18	12 x 9 = 108
5 x 10 = 50	2 x 10 = 20	12 x 10 = 120
5 x 11 = 55	2 x 11 = 22	12 x 11 = 132
5 x 12 = 60	2 x 12 = 24	12 x 12 = 144

**REVISION - Lab 5: 5x05c – Reciprocals**

4/05/2022 –Wednesday, Week 8

**Program Aim**

Prompt the user with the ? symbol, and then wait for input. Using a loop, output “1 / (number)”, from 1 to the value input by the user.

**Implementation**

Source code	Output
? 10	? 20
1 / 1 is 1.000000	1 / 1 is 1.000000
1 / 2 is 0.500000	1 / 2 is 0.500000
1 / 3 is 0.333333	1 / 3 is 0.333333
1 / 4 is 0.250000	1 / 4 is 0.250000
1 / 5 is 0.200000	1 / 5 is 0.200000
1 / 6 is 0.166667	1 / 6 is 0.166667
1 / 7 is 0.142857	1 / 7 is 0.142857
1 / 8 is 0.125000	1 / 8 is 0.125000
1 / 9 is 0.111111	1 / 9 is 0.111111
1 / 10 is 0.100000	1 / 10 is 0.100000
1 / 11 is 0.090909	1 / 11 is 0.090909
1 / 12 is 0.083333	1 / 12 is 0.083333
1 / 13 is 0.076923	1 / 13 is 0.076923
1 / 14 is 0.071429	1 / 14 is 0.071429
1 / 15 is 0.066667	1 / 15 is 0.066667
1 / 16 is 0.062500	1 / 16 is 0.062500
1 / 17 is 0.058824	1 / 17 is 0.058824
1 / 18 is 0.055556	1 / 18 is 0.055556
1 / 19 is 0.052632	1 / 19 is 0.052632
1 / 20 is 0.050000	1 / 20 is 0.050000

**REVISION - Lab 5: 5x05c – Reciprocals****4/05/2022 –Wednesday, Week 8**

Program Aim	
Prompt the user with the ? symbol, and then wait for input. Using a loop, output “1 / (number)”, from 1 to the value input by the user.	
Implementation	
Source code	Output
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      float divisor = 0.0f; 6      const float reciprocal_base = 1.0f; 7 8      printf("?"); 9      scanf("%f", &amp;divisor); 10 11     for (float i = reciprocal_base; i &lt;= divisor; i++) 12     { 13         printf("1 / %.f is %f \n", i, reciprocal_base / i); 14     } 15 16     return 0; 17 }</pre>	<p>? 10</p> <p>1 / 1 is 1.000000      1 / 2 is 0.500000      1 / 3 is 0.333333      1 / 4 is 0.250000      1 / 5 is 0.200000      1 / 6 is 0.166667      1 / 7 is 0.142857      1 / 8 is 0.125000      1 / 9 is 0.111111      1 / 10 is 0.100000      1 / 11 is 0.090909      1 / 12 is 0.083333      1 / 13 is 0.076923      1 / 14 is 0.071429      1 / 15 is 0.066667      1 / 16 is 0.062500      1 / 17 is 0.058824      1 / 18 is 0.055556      1 / 19 is 0.052632      1 / 20 is 0.050000</p>

**REVISION - Lab 5: 5x09c – Insert Value****4/05/2022 –Wednesday, Week 8****Program Aim**

Declare an array of max 100 elements. Then Ask user how many elements they want, allow them to input a value for each element. Next, ask user for a new value to insert into the array, as well as the index. Place the new value at this position and move all subsequent values along one in the array. Print out the updated array.

**Plan (array shifting only)**

FOR i initially equals max elements, LOOP through array backwards until i equals the insert location  
(decrease the element index value from the insert location to max elements to make way for an additional empty array at insert location)- Array with index of i equals i minus 1

INSERT new element into the insert location of the array minus 1

**Implementation**

Source code	Output
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      int elements = 0; 6      int new_element = 0; 7      int insert_location = 0; 8      int array[100] = {0}; 9 10     printf("Input total number of elements required: "); 11     scanf("%d", &amp;elements); 12 13     for (int i = 0; i &lt; elements; i++) 14     { 15         printf("Input element [%d]: ", i); 16         scanf("%d", &amp;array[i]); 17     } 18 19     printf("\nBefore Insertion: \n"); 20     for (int i = 0; i &lt; elements; i++) 21     { 22         printf("Element [%d] is %d \n", i, array[i]); 23     } 24 25     printf("\nInput a new value to insert: "); 26     scanf("%d", &amp;new_element); 27 28     printf("\nInput where to insert the value %d: ", new_element); 29     scanf("%d", &amp;insert_location); 30 31     // go through array index backwards until i = insert location 32     for (int i = elements; i &gt;= insert_location; i--) 33     { 34         // shift elements forward 35         array[i] = array[i - 1]; 36     } 37 38     // insert new element at position 39     array[insert_location - 1] = new_element; 40 41     printf("\nAfter insertion: \n"); 42     for (int i = 0; i &lt; elements + 1; i++) 43     { 44         printf("Element [%d] is %d \n", i, array[i]); 45     } 46 47 }</pre>	<pre> Input total number of elements required: 5 Input element [0]: 1 Input element [1]: 2 Input element [2]: 3 Input element [3]: 4 Input element [4]: 5  Before Insertion: Element [0] is 1 Element [1] is 2 Element [2] is 3 Element [3] is 4 Element [4] is 5  Input a new value to insert: 10  Input where to insert the value 10: 3  After insertion: Element [0] is 1 Element [1] is 2 Element [2] is 10 Element [3] is 3 Element [4] is 4 Element [5] is 5 </pre>

**Revision (from lecture topics checklist)**

4/05/2022 –Wednesday, Week 8

Lab 6			
Checklist	Practice / notes		
Lecture topics in preparation for Lab Tutorial 6:			
	Confident?	Familiar?	Unknown?
<ul style="list-style-type: none"> <li>• <b>for</b> loops and arrays           <ul style="list-style-type: none"> <li>◦ Printing an array</li> <li>◦ Copying between arrays</li> <li>◦ Reversing an array</li> </ul> </li> <li>• <b>for</b> loops with nested selection           <ul style="list-style-type: none"> <li>◦ Basic searching</li> <li>◦ Criteria-based searching</li> <li>◦ Statistics computation</li> <li>◦ C-String length</li> <li>◦ C-String comparison</li> </ul> </li> <li>• How to program... good practices</li> <li>• Nested loops:           <ul style="list-style-type: none"> <li>◦ Inner loop</li> <li>◦ Outer loop</li> </ul> </li> <li>• The <b>goto</b> keyword           <ul style="list-style-type: none"> <li>◦ Coding Standards: <b>goto</b></li> </ul> </li> </ul>	✓		
		✓	
		✓	
		✓	
		✓	
		✓	
		✓	
		✓	
		✓	
		✓	
		✓	
		✓	
		✓	
			✓
			✓
	<b>Copying between arrays:</b> create FOR loop that iterates through array1, IN FOR loop, assign each iterated index element to array2  <b>Reversing an array:</b> FORMAT: <code>backward[index]=forward[array_length-index];</code>  <b>Basic searching:</b> REMEMBER, set the variable that finds the index of the value to: <code>int found_at = -1;</code> To find an element, FOR loop through array elements until an element matches what you are finding  <b>Statistics computation:</b> compute avarage element using a for loop: <code>for (int i = 0; i &lt; 6; ++i) {     sum += stats_data[i]; } -&gt; += to add each element</code> THEN COMPUTE AVERAGE USING SUM OUTPUT  <b>Find length of c-string:</b> WHILE element of c-string IS NOT null ("\0"), COUNT the number of non-null elements.  <b>TO COMPARE C-STRINGS WITHOUT strcmp:</b> <ul style="list-style-type: none"> <li>- Find length of each c-string</li> <li>- Create variable to check match that equals 1</li> <li>- COMPARE IF lengths are the same</li> <li>- FOR LOOP through index until length is reached</li> <li>- IF first array element DOESN'T EQUAL second array element, set match variable to 0 ...</li> </ul>		

**REVISION - Lab 6: 6x04e - Password Guess****4/05/2022 –Wednesday, Week 8**

<b>Program Aim</b>	
Ask the user to guess a secret hidden password. The hidden password is programmer. The user is allowed a maximum of 5 guesses.	
<b>Implementation</b>	
<b>Source code</b>	<b>Output</b>
<pre> 1 #include &lt;stdio.h&gt; 2 3 int main(void) 4 { 5     const char password[] = "programmer"; 6     char guess[100]; 7     int guess_length = 0; 8     int guesses_total = 5; 9     int guesses_count = 0; 10    int correct_guess = 1; 11 12    while (guesses_count &lt;= guesses_total) 13    { 14        printf("Guess the password: "); 15        scanf("%s", &amp;guess); 16 17        while (guess[guess_length] != '\0') 18        { 19            guess_length++; 20        } 21 22        for (int i = 0; i &lt;= guess_length; i++) 23        { 24            if (guess[i] != password[i]) 25            { 26                correct_guess = 0; 27            } 28            else 29                correct_guess = 1; 30        } 31 32        if (correct_guess == 1) 33        { 34            printf("You guessed correctly! Well done! \n"); 35            break; 36        } 37        else 38        { 39            printf("Incorrect! You have %d guesses left. \n\n", guesses_total - guesses_count); 40        } 41 42        guesses_count++; 43    } 44 45    return 0; } </pre>	<p>All incorrect guesses</p> <p>Guess the password: hello? Incorrect! You have 5 guesses left.</p> <p>Guess the password: ahhh Incorrect! You have 4 guesses left.</p> <p>Guess the password: password Incorrect! You have 3 guesses left.</p> <p>Guess the password: 12345 Incorrect! You have 2 guesses left.</p> <p>Guess the password: 54321 Incorrect! You have 1 guesses left.</p> <p>Guess the password: HELP Incorrect! You have 0 guesses left.</p> <p>Guess correctly on 3<sup>rd</sup> guess</p> <p>Guess the password: programer Incorrect! You have 5 guesses left.</p> <p>Guess the password: progammer Incorrect! You have 4 guesses left.</p> <p>Guess the password: programmer You guessed correctly! Well done!</p>

## Revision (from lecture topics checklist)

4/05/2022 –Wednesday, Week 8

Checklist				Practice / notes		
		Confident?	Familiar?	Unknown?		
Lecture topics in preparation for Lab Tutorial 7:						
● Scope: ○ Local variables		✓				
○ Variable lifespan		✓				
○ Out of scope		✓				
○ With sequence...		✓				
○ With selection...		✓				
○ With iteration...		✓				
○ With nesting...		✓				
● Functions: ○ Modularity	✓					
○ The <code>void</code> keyword	✓					
○ The <code>return</code> keyword	✓					
○ Defining a function	✓					
○ Calling a function	✓					
○ Function prototypes: Declaring	✓					
○ Linker errors		✓				
● The Call Stack		✓				
○ Stack Frames		✓				
● Visual Studio Debugger:		✓				
○ Stepping into		✓				
○ Stepping out		✓				
○ Stepping over		✓				
○ Run to next breakpoint		✓				
○ The Call Stack		✓				
Lecture topics in preparation for Lab Tutorial 7:						
● Structured programming	✓					
● Functions:	✓					
○ Arguments		✓				
○ Parameters	✓					
○ Call-by-value		✓				
○ Returning data from functions	✓					
● Variable scope:	✓					
○ Local scope	✓					
○ Global scope	✓					
● Visual Studio: Call Stack Tracing		✓				

## MIDSEM PRACTICAL REFLECTION – 5/05/2022 – Thursday, week 8

I was able to work through the first 5 questions relatively easily, however, I struggled with the ASCII depth triangle (question 6)- this may be due to my lack of revision over ASCII shape printing using loops.

### How to improve for next exam:

- Attempt alternative exercises more often! They provide more insight into each theme, and would help me to understand the concepts better!

## LAB 8: 8x01a – Student Structure

**5/05/2022 – Thursday (lab), week 8**

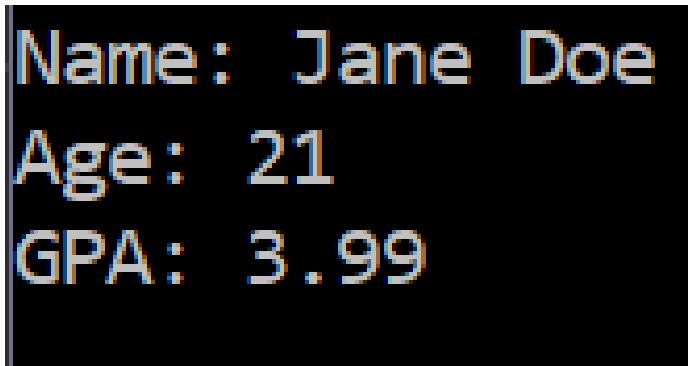
### Program aim

Create a **structure** named **Student**. All students have an age, a name and a GPA. Declare a **Student** structure variable in the main function and assign values to the members in it. Create a function named **print\_student** that prints a student's details. Call **print\_student** function to print the student details of the Jane Doe structure variable.

### Step-by-step plan

1. Create **struct** named **Student**
2. Include members: **name** (c-string), **age** (int), **gpa** (float)
3. Declare **Student** variable called **jane\_doe** in **main**
4. Assign each member of **Student** a value in **main**
5. Create a function called **print\_student** that prints each member of **student** using dot operator
6. Call **print\_student** in **main**

### Implementation

Source Code	Output/s / Testing
<pre> 1  #include &lt;stdio.h&gt; 2 3  struct Student 4  { 5      char name[20]; 6      int age; 7      float gpa; 8  }; 9 10 void print_student(struct Student student_name) 11 { 12     printf("Name: %s \n", student_name.name); 13     printf("Age: %d \n", student_name.age); 14     printf("GPA: %.2f \n", student_name.gpa); 15 } 16 17 int main(void) 18 { 19     struct Student jane_doe; 20 21     sprintf(jane_doe.name, "Jane Doe"); 22     jane_doe.age = 21; 23     jane_doe.gpa = 3.99f; 24 25     print_student(jane_doe); 26 27     return 0; 28 }</pre>	 <pre> Name: Jane Doe Age: 21 GPA: 3.99 </pre>

### Lessons learned

REMEMBER declare **structs** before functions, use **sprintf** to assign strings to **struct** members

## Week 8 lecture 024 Notes- More Pointers, Passing Arrays and C-Strings, Pointer Arithmetic

6/05/2022- Friday, week 8

Notes Retrieved from Steffan Hooper (2022)

Declaring variable that stores address of a struct: `Struct Person* p_lecturer = &lecturer`

Dereference pointer, pointer name, address location

Pointer (->) operator: gain access to a member in a structure via a pointer

`p_lecturer->age = 35;` (ASSIGN INT VALUE TO AGE MEMBER IN STRUCTURE VARIABLE)

`sprint(p_lecturer->job_title, "Lecturer");` (ASSIGN STRING TO JOB\_TITLE USING SPRINTF)

pointer variable, structure member, pointer operator

. dot operators can be used, but pointers need to be 'dereferenced'. It is better to use -> operator

`(*p_lecturer).age = 35;`

Dereference pointer, access the struct member using dot operator

Pass structure by reference: pass address of structure into function using pointer

`void print_person(Struct Person* p_person)`

{

`printf("Age: %d", p_person->age);`

}

Dereference pointer, struct variable, pointer operator (gain access to member)

...

Call function in main with struct variable address as argument: `print_person(&lecturer);`

Passing structure data create **copies** of data. Pointers/addresses allow **sharing** of data.

Using pointer operator for multiple structure members:

`printf("Name: %s %s", student->first_name, student->last_name);`

`*array_name` will only point to the first element in the array

Pass array by reference: `void array_function(int array[5]) ...`

Pass array as pointers: `void array_function(int* array) ...`

Better way to pass array as pointer: `void array_function(int* array, int size)` size = array length

Iterate through array index to modify each element: `for (int i = 0; i < size; ++i) ...`

Pointer arithmetic:

- Accessing array elements uses pointers! `local_array[0]` same as `*(local_array+0)`

- Increment pointer to move through array:

`*current++ = 0;` (INCREASES ADDRESS STORED IN POINTER)

`*current++ = 1;` ...

ORDER OF OPERATIONS: 1. Pointer/dereference, 2. assignment, 3. increment address

- `(*my_pointer)++;` (INCREASES VALUE IN RAM AT ADDRESS STORED IN POINTER)

- Revisit order of operations precedence table...

- Don't use this notation as it is confusing for programmers: `2[local_array]`

Increasing address with strings using pointer

... `char* p_first = array_name;`

FOR LOOP

... `printf("%c", *(p_first + i)) ...`

Difference between char array and pointers to char arrays:

`char greeting[] = "Hi";` READABLE AND WRITABLE (can be changed)

`char* pointer_greeting[] = "Hi";` READ ONLY! (Writing to causes access violation error)

**Week 8 lecture 024 Notes- More Pointers, Passing Arrays and C-Strings, Pointer Arithmetic**

(Continued) 6/05/2022- Friday, week 8

Notes Retrieved from Steffan Hooper (2022)

Pointer to a pointer: `int** pointer_to_pointer = &pointer;`

Double-dereferencing, second pointer name, first pointer address

Pointer to a pointer that points to a pointer: `int*** ...`**LAB 8: 8x02a – Soft Drink**

8/05/2022 – Sunday, week 8

**Program aim**

Declare a structure that represents a soft drink. The soft drink structure must store: Name (16 `char` long), Serving size (`int`), Energy content (`int`), Caffeine content (`float`), Maximum daily intake (`int`).

Declare a structure variable instance named `life_mod`, which will store the data for the energy drink named “Life Modulus”. Serving size: 250mL, energy content: 529kJ, caffeine content: 80.5mg, maximum daily intake: 500mL.

Write a function named `print_soft_drink`, which takes parameter: `soft_drink`. Call `print_soft_drink` from main.

**Pseudocode**

```

CREATE struct called soft_drink
  CREATE struct members: c-string name, int serving_size, int energy_content,
    float caffeine_content, int max_daily_intake
END STRUCT

START FUNCTION print_soft_drink with parameter soft_drink (structure variable instance)
  Print struct members of soft_drink
END print_soft_drink

START main
  DECLARE structure variable instance life_mod
  SET MEMBER VALUES: name: "Life Modulus", serving_size: 250, energy_content: 529,
    caffeine content: 80.5, max_daily_intake: 500
  CALL FUNCTION print_soft_drink
END main

```

**Implementation**

Source Code	Output/s / Testing
<pre> 1  #include &lt;stdio.h&gt; 2 3  struct soft_drink 4  { 5      char name[16]; 6      int serving_size; 7      int energy_content; 8      float caffeine_content; 9      int max_daily_intake; 10 } 11 12 void print_soft_drink(struct soft_drink drink_name) 13 { 14     printf("A soft drink... \n\n"); 15     printf("Name: %s \n", drink_name.name); 16     printf("Serving size: %d mL \n", drink_name.serving_size); 17     printf("Energy content: %d kJ \n", drink_name.energy_content); 18     printf("Caffeine content: %f mg \n", drink_name.caffeine_content); 19     printf("Maximum daily intake: %d mL \n", drink_name.max_daily_intake); 20 } 21 22 int main(void) 23 { 24     struct soft_drink life_mod; 25 26     sprintf(life_mod.name, "Life Modulus"); 27     life_mod.serving_size = 250; 28     life_mod.energy_content = 529; 29     life_mod.caffeine_content = 80.5f; 30     life_mod.max_daily_intake = 500; 31 32     print_soft_drink(life_mod); 33 34     return 0; 35 }</pre>	<pre> A soft drink...  Name: Life Modulus Serving size: 250 mL Energy content: 529 kJ Caffeine content: 80.500000 mg Maximum daily intake: 500 mL </pre>

**LAB 8: 8x03a – Distance 3D**

8/05/2022 – Sunday, week 8

**Program aim**

Declare **struct** named **Point3D** which represents a point in three-dimensions. Allocate two **Point3D** structures inside the **main** function. Assign them the test values of  $p_1 = < 1, 2, 3 >$  and  $p_2 = < 4, 5, 6 >$ . Write a function called **compute\_distance3d**, which takes as input two **Point3D** structures by value. Inside this function, calculate the distance between the two points. Return the distance.

Use **math.h** library to access square root function. Where:  $p_1 = < x_1, y_1, z_1 >$  and  $p_2 = < x_2, y_2, z_2 >$ ,

$$\text{Distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

\*Next, Ask user for two 3D points, pass values to **compute\_distance3d**, print result returned from **compute\_distance3d**.

**Pseudocode**

```

CREATE struct called Point3D
    CREATE struct members: float x, float y, float z
END STRUCT

START FUNCTION compute_distance3d with parameters Point3D p1 and Point3D p2
    COMPUTE distance using formula: distance = sqrt((x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2)
END compute_distance3d

START main
    DECLARE structure variable instance p1
    SET MEMBER VALUES: x: 1, y: 2, z: 3

    DECLARE structure variable instance p2
    SET MEMBER VALUES: x: 4, y: 5, z: 6

    CALL FUNCTION compute_distance3d
END main

```

**Implementation****Source Code**

```

1 #include <stdio.h>
2 #include <math.h>
3
4 struct Point3D
5 {
6     float x;
7     float y;
8     float z;
9 };
10
11 float compute_distance3d(struct Point3D p1, struct Point3D p2)
12 {
13     // distance = sqrt((x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2)
14     return sqrtf(powf((p2.x - p1.x), 2.0f) + powf((p2.y - p1.y), 2.0f) + powf((p2.z - p1.z), 2.0f));
15 }
16
17 int main(void)
18 {
19     struct Point3D p1;
20
21     p1.x = 1;
22     p1.y = 2;
23     p1.z = 3;
24
25     struct Point3D p2;
26
27     p2.x = 4;
28     p2.y = 5;
29     p2.z = 6;
30
31     printf("For p1 = < 1, 2, 3 > and p2 = < 4, 5, 6 > \n");
32     printf("3D distance is: %f \n", compute_distance3d(p1, p2));
33
34     return 0;
35 }

```

**Output/s / Testing**

Using test values:  
 For  $p_1 = < 1, 2, 3 >$  and  $p_2 = < 4, 5, 6 >$   
 3D distance is: 5.196152

VERIFYING IF COMPUTATION IS CORRECT:  
<https://www.calculatorsoup.com/calculators/geometry-solids/distance-two-points.php>

**3D Distance Calculator**

(X <sub>1</sub> , Y <sub>1</sub> , Z <sub>1</sub> ) =	<input type="text" value="1,2,3"/>
(X <sub>2</sub> , Y <sub>2</sub> , Z <sub>2</sub> ) =	<input type="text" value="4,5,6"/>

Answer:  
 $d = 5.196152$

For:  
 $(X_1, Y_1, Z_1) = (1, 2, 3)$   
 $(X_2, Y_2, Z_2) = (4, 5, 6)$

Distance Equation Solution:

$$d = \sqrt{(4 - 1)^2 + (5 - 2)^2 + (6 - 3)^2}$$

$$d = \sqrt{(3)^2 + (3)^2 + (3)^2}$$

$$d = \sqrt{9 + 9 + 9}$$

$$d = \sqrt{27}$$

$$d = 5.196152$$

**LAB 8: 8x03a – Distance 3D (continued)**

8/05/2022 – Sunday, week 8

**Program aim**

\*NEXT: Ask user for two 3D points, pass values to `compute_distance3d`, print result returned from `compute_distance3d`.

**Pseudocode (modified code)**

```
... Previous code same as before...
START main
DECLARE structure variable instance p1
PROMPT AND READ p1 'x', 'y', and 'z' value in FORMAT: "x, y, z"

DECLARE structure variable instance p2
PROMPT AND READ p2 'x', 'y', and 'z' value in FORMAT: "x, y, z"

CALL FUNCTION compute_distance3d
END main
```

**Implementation****Modified Source Code**

```
1 #include <stdio.h>
2 #include <math.h>
3
4 struct Point3D
5 {
6     float x;
7     float y;
8     float z;
9 };
10
11 float compute_distance3d(struct Point3D p1, struct Point3D p2)
12 {
13     // distance = sqrt((x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2)
14     return sqrtf(powf((p2.x - p1.x), 2.0f) + powf((p2.y - p1.y), 2.0f) + powf((p2.z - p1.z), 2.0f));
15 }
16
17 int main(void)
18 {
19     struct Point3D p1;
20
21     printf("Enter the 3D points for distance #1 (x,y,z): ");
22     scanf("%f, %f, %f", &p1.x, &p1.y, &p1.z);
23
24     struct Point3D p2;
25
26     printf("Enter the 3D points for distance #2 (x,y,z): ");
27     scanf("%f, %f, %f", &p2.x, &p2.y, &p2.z);
28
29     float distance = compute_distance3d(p1, p2);
30
31     printf("3D distance is: %f \n", distance);
32
33     return 0;
34 }
```

**Output/s / Testing**

Output for:  $p_1 = <1, 2, 3>$  and  $p_2 = <4, 5, 6>$   
 Enter the 3D points for distance #1 (x,y,z): 1, 2, 3  
 Enter the 3D points for distance #2 (x,y,z): 4, 5, 6  
 3D distance is: 5.196152

Output for:  $p_1 = <0, 3, 4>$  and  $p_2 = <5, 8, 10>$   
 Enter the 3D points for distance #1 (x,y,z): 0,3,4  
 Enter the 3D points for distance #2 (x,y,z): 5,8,10  
 3D distance is: 9.273619

Output for:  $p_1 = <-1, -2, -3>$  and  $p_2 = <1, 2, 3>$   
 Enter the 3D points for distance #1 (x,y,z): -1,-2,-3  
 Enter the 3D points for distance #2 (x,y,z): 1,2,3  
 3D distance is: 7.483315

Output for:  $p_1 = <0.5, 0.1, 0.2>$  and  $p_2 = <1, 1, 1>$   
 Enter the 3D points for distance #1 (x,y,z): 0.5,0.1,0.2  
 Enter the 3D points for distance #2 (x,y,z): 1,1,1  
 3D distance is: 1.303841

Output for:  $p_1 = <5, 5, 5>$  and  $p_2 = <5, 5, 5>$   
 Enter the 3D points for distance #1 (x,y,z): 5,5,5  
 Enter the 3D points for distance #2 (x,y,z): 5,5,5  
 3D distance is: 0.000000

**Lessons learned**

- I learnt how to utilise multiple `struct` variable instances in functions, computations, and user input.
- REMEMBER for `math.h` library, `powf()` is float power to, `sqrtf()` is float square root
- I remembered how to utilise multiple inputs in a single line

# **WEEK 9**

## Week 9 lecture 025 Notes- File Input and Output, Hex Editor

9/05/2022- Monday, week 9

Notes Retrieved from Steffan Hooper (2022)

All the data stored, from processes, in our programs are saved in RAM. The data disappears when the program exits.

### **File input and output:**

With files, we can save data to the disk and read it to the programs.

Input and output streams: sequence of data that becomes available over time. Input buffer (keyboard input) is a stream.

- Text files
- Binary files

**FILE**- structure representing file data type. stdio.h functions use **FILE\***

**fopen(char\* filename, char\* mode)** - returns a **FILE\*** (pointer to file).

c-string representing filename, c-string representing mode ("r": reading, "w": open for writing, "a": append to file, "r+": reading and writing... "b": open in binary mode, "t": open in text mode...)

**fclose(FILE\* stream)** - release resource when done with file. "0" return = buffer flushed successfully. Else: error. Parameter: **stream to close**.

**fprintf(FILE\* stream, char\* format, ...)** - print formatted text to stream.

File stream pointer, string literal containing formatted-string.

**fflush(FILE\* stream)** - flush a stream. Return of "0": success, return **EOF**: failure.

### **EXAMPLE CODE (file input and output):**

```
...
FILE* p_file = 0;

p_file = fopen("example.txt", "w");

fprintf(p_file, "Hello\n");
fprintf(p_file, "COMP500\n");
fprintf(p_file, "and\n");
fprintf(p_file, "ENSE501\n");
fprintf(p_file, "students\n");

fclose(p_file);

return 0;
...
```

To access the text file: right click **source.c** tab > open containing folder > find txt file in directory

**fscanf(FILE\* stream, char\* format, variable/pointer)**;  
File stream pointer, string literal containing formatted-string, to be scanned into

REMEMBER %s[^\\n]: [^\\n] means read everything that is not newline.

Throw away: scan newline by "%\*c"

## Week 9 lecture 025 Notes- File Input and Output, Hex Editor (continued)

9/05/2022- Monday, week 9

Notes Retrieved from Steffan Hooper (2022)

`int fgetc(FILE* stream)` ; - read single character from a file stream.  
FILE stream pointer

**EOF:** end of file

EXAMPLE USE OF **EOF**: ... `while (EOF != char_read)` ... checking if end of file hasn't been reached

`int fputc(int c, FILE* stream)` ; - write single character to a file stream  
FILE stream pointer

`int fputs(const char* str, FILE* stream)` ; - write a char array to a file stream  
str pointer of char array to read from, FILE stream pointer

`char* fgets(char* str, int n, FILE* stream)` ; - read an array of chars from a file stream  
str pointer of char array to write to, max num of chars to read, FILE stream pointer

**Binary Files:** uses same reading/writing functions as files, but in binary form.

`size_t fwrite(void* ptr, size_t size, size_t nmemb, FILE* stream);`  
pointer to written array elements, size in bytes for written array, num elements, FILE stream pointer

**Hex editor/binary file editor/byte editor:** program allowing viewing and manipulation of binary data in a file.

We can manipulate file pointers using additional functions...

**LAB 8: 8x04a – Returning a Student**

9/05/2022 – Monday, week 9

**Program aim**

Complete the given instructions in the TODO statements

**Implementation****Given Source Code**

```

1  #include <stdio.h>
2
3  struct Student
4  {
5      char first_name[32];
6      char last_name[32];
7      int stream_code;
8  };
9
10 struct Student query_student(void)
11 {
12     // TODO: Declare a local Student variable.
13
14     printf("Input first name: ");
15     // TODO: Scan for user input...
16
17     printf("Input last name: ");
18     // TODO: Scan for user input...
19
20     printf("Input stream code: ");
21     // TODO: Scan for user input...
22
23     // TODO: Return the local Student variable.
24 }
25
26 int main(void)
27 {
28     struct Student query;
29
30     query = query_student();
31
32     printf("%s ", query.first_name);
33     printf("%s ", query.last_name);
34     printf("is in stream %d.", query.stream_code);
35
36     return 0;
37 }
```

**My Source Code**

```

1  #include <stdio.h>
2
3  struct Student
4  {
5      char first_name[32];
6      char last_name[32];
7      int stream_code;
8  };
9
10 struct Student query_student(void)
11 {
12     // TODO: Declare a local Student variable.
13     struct Student a_student;
14
15     printf("Input first name: ");
16     // TODO: Scan for user input...
17     scanf("%31s", &a_student.first_name);
18
19     printf("Input last name: ");
20     // TODO: Scan for user input...
21     scanf("%31s", &a_student.last_name);
22
23     printf("Input stream code: ");
24     // TODO: Scan for user input...
25     scanf("%d", &a_student.stream_code);
26
27     // TODO: Return the local Student variable.
28     return a_student;
29 }
30
31 int main(void)
32 {
33     struct Student query;
34
35     query = query_student();
36
37     printf("%s ", query.first_name);
38     printf("%s ", query.last_name);
39     printf("is in stream %d.", query.stream_code);
40
41     return 0;
42 }
```

**Lessons learned**There is no need for `sprintf()` when you're scanning user input into a `struct` member that is a c-string...

**LAB 8: 8x05a – Fractions**

9/05/2022 – Monday, week 9

Practice the following fraction operations by hand with pen and paper – avoid using your calculator:

Questions	Working-out / Answers
1) Compute $\frac{1}{2} + \frac{1}{4}$	$1) \frac{1}{2} + \frac{1}{4} = \frac{2}{4} + \frac{1}{4} = \frac{3}{4}$
2) Compute $\frac{1}{2} - \frac{1}{4}$	$2) \frac{1}{2} - \frac{1}{4} = \frac{2}{4} - \frac{1}{4} = \frac{1}{4}$
3) Compute $\frac{1}{2} \times \frac{1}{4}$	$3) \frac{1}{2} \times \frac{1}{4} = \frac{1}{8}$
4) Compute $\frac{1}{2} \div \frac{1}{4}$	$4) \frac{1}{2} \div \frac{1}{4} = \frac{1}{2} \times \frac{4}{1} = \frac{4}{2} = 2$

**Program aim**

Create a **struct** called **Fraction** with int-only members: **numerator** and **denominator**. Create 4 functions that compute addition, subtraction, multiplication, and division of two fractions:  $\frac{1}{2}$  and  $\frac{1}{4}$ . These functions are not required to simplify the resulting fraction. Create a function that prints the computed fraction in the format: "2/3".

**Pseudocode**

```

DECLARE struct Fraction
    CREATE struct members numerator (int) and denominator (int)
END struct Fraction

DECLARE COMPUTATION functions with parameters Fraction a and Fraction b:
    add_fractions, subtract_fractions, multiply_fractions, divide_fractions
DECLARE print_function with parameter Fraction c

START main
    Declare struct variable instances a and b
    Assign Fraction a numerator to 1 and Fraction a denominator to 2
    Assign Fraction b numerator to 1 and Fraction b denominator to 4

    Declare struct variable instances add, subtract, multiply, and divide
    Assign each struct variable to the returned struct of respective computation functions

    PRINT each computation result using print_function
END main

START add_fractions
    Declare struct variable instance: add

    COMPUTE: Make denominator the same and multiply numerators accordingly
    COMPUTE: add numerators of a and b to get addition (add)

    RETURN add
END add_fractions

```

**LAB 8: 8x05a – Fractions (continued)**

9/05/2022 – Monday, week 9

**Pseudocode (continued)**

```

START subtract_fractions
    Declare struct variable instance: subtract

    COMPUTE: Make denominator the same and multiply numerators accordingly
    COMPUTE: subtract numerators of a and b to get subtraction (subtract)

    RETURN subtract
END subtract_fractions

START multiply_fractions
    Declare struct variable instance: multiply

    COMPUTE: multiply numerators a * b
    COMPUTE: multiply denominators a * b

    RETURN multiply
END multiply_fractions

START print_function
    PRINT resulting fraction in format "(resulting numerator)/(resulting denominator)"
END print_function

```

**Implementation (no additional functions)****Source Code 1-51 (structs, function prototypes, main)**

```

1 #include <stdio.h>
2
3 struct Fraction
4 {
5     int numerator;
6     int denominator;
7 };
8
9 struct Fraction add_fractions(struct Fraction a, struct Fraction b);
10 struct Fraction subtract_fractions(struct Fraction a, struct Fraction b);
11 struct Fraction multiply_fractions(struct Fraction a, struct Fraction b);
12 struct Fraction divide_fractions(struct Fraction a, struct Fraction b);
13
14 void print_function(struct Fraction c);
15
16 int main(void)
17 {
18     struct Fraction a;
19     a.numerator = 1;
20     a.denominator = 2;
21
22     struct Fraction b;
23     b.numerator = 1;
24     b.denominator = 4;
25
26     struct Fraction add;
27     add = add_fractions(a, b);
28
29     struct Fraction subtract;
30     subtract = subtract_fractions(a, b);
31
32     struct Fraction multiply;
33     multiply = multiply_fractions(a, b);
34
35     struct Fraction divide;
36     divide = divide_fractions(a, b);
37
38     printf("ADDITION: ");
39     print_function(add);
40
41     printf("SUBTRACTION: ");
42     print_function(subtract);
43
44     printf("MULTIPLICATION: ");
45     print_function(multiply);
46
47     printf("DIVIDE: ");
48     print_function(divide);
49
50     return 0;
51 }

```

**Source Code 52-85 (add\_fractions, subtract\_fractions)**

```

52 struct Fraction add_fractions(struct Fraction a, struct Fraction b)
53 {
54     struct Fraction add;
55
56     // Make denominators the same
57     add.denominator = a.denominator * b.denominator;
58
59     // Multiply numerators accordingly
60     a.numerator = a.numerator * b.denominator;
61     b.numerator = b.numerator * a.denominator;
62
63     // Add numerators for addition
64     add.numerator = a.numerator + b.numerator;
65
66     return add;
67 }
68
69
70 struct Fraction subtract_fractions(struct Fraction a, struct Fraction b)
71 {
72     struct Fraction subtract;
73
74     // Make denominators the same
75     subtract.denominator = a.denominator * b.denominator;
76
77     // Multiply numerators accordingly
78     a.numerator = a.numerator * b.denominator;
79     b.numerator = b.numerator * a.denominator;
80
81     // Subtract numerators for Subtraction
82     subtract.numerator = a.numerator - b.numerator;
83
84     return subtract;
85 }

```

**LAB 8: 8x05a – Fractions (continued)**

9/05/2022 – Monday, week 9

**Implementation (no additional functions)****Source Code 86-114 (multiply\_fractions, divide\_fractions, print\_function)**

```

86 struct Fraction multiply_fractions(struct Fraction a, struct Fraction b)
87 {
88     struct Fraction multiply;
89
90     // Multiply numerators and denominators
91     multiply.numerator = a.numerator * b.numerator;
92     multiply.denominator = a.denominator * b.denominator;
93
94     return multiply;
95 }
96
97
98 struct Fraction divide_fractions(struct Fraction a, struct Fraction b)
99 {
100    struct Fraction divide;
101
102    // cross-multiply numerators and denominators
103    divide.numerator = a.numerator * b.denominator;
104    divide.denominator = a.denominator * b.numerator;
105
106    return divide;
107 }
108
109 void print_function(struct Fraction c)
110 {
111     printf("%d/%d \n", c.numerator, c.denominator);
112 }
113
114

```

**Output/s**

\*\*\* These functions are not required to simplify the resulting fraction \*\*\*

The fractions are not simplified. However, they are correct when simplified.

ADDITION:  $\frac{6}{8}$

SUBTRACTION:  $\frac{2}{8}$

MULTIPLICATION:  $\frac{1}{8}$

DIVIDE:  $\frac{4}{2}$

$$\begin{aligned}
 1) & \frac{1^{\cancel{x}^2}}{2^{\cancel{x}^2}} + \frac{1}{4} = \frac{2}{4} + \frac{1}{4} = \frac{3}{4} \\
 2) & \frac{1^{\cancel{x}^2}}{2^{\cancel{x}^2}} - \frac{1}{4} = \frac{2}{4} - \frac{1}{4} = \frac{1}{4} \\
 3) & \frac{1}{2} \times \frac{1}{4} = \frac{1}{8} \\
 4) & \frac{1}{2} \div \frac{1}{4} = \frac{1}{2} \times \frac{4}{1} = \frac{4}{2} = 2
 \end{aligned}$$

Same as:

I found a way to simplify these fractions, so I am going to create a simplifying function that will be called within each fraction computation function. I will also attempt to move the process of making the denominators the same to a separate function as this is a repeated process.

**Pseudocode of simplify\_fraction ('c' represents resultant fraction)**

```

START simplify_fraction with parameter: Fraction c

IF Fraction c's denominator AND numerator has no remainder when divided by 2

LOOP WHILE Fraction c's numerator is less than its denominator AND
Fraction c's denominator AND numerator has no remainder when divided by 2

    COMPUTE: divide denominator c by 2
    COMPUTE: divide numerator c by 2

ENDWHILE
...

```

## LAB 8: 8x05a – Fractions (continued)

9/05/2022 – Monday, week 9

### Pseudocode of simplify\_fraction CONTINUED

```
IF numerator c is greater than its denominator (THIS IS FOR WHOLE NUMBERS)

    COMPUTE: divide numerator c by denominator c
    COMPUTE: divide denominator c by itself to get 1

ENDIF
ENDIF

RETURN c
END simplify_fraction
```

### Implementation of added function: simplify\_fraction

#### Source Code FOR simplify\_fraction

Prototype:

14     **struct Fraction simplify\_fraction(struct Fraction c);**

... Function definition:

```
55     struct Fraction simplify_fraction(struct Fraction c)
56     {
57         if ((c.denominator % 2) == 0 && (c.numerator % 2) == 0)
58         {
59             while (c.numerator < c.denominator && (c.denominator % 2) == 0 && (c.numerator % 2) == 0)
60             {
61                 c.denominator = c.denominator / 2;
62                 c.numerator = c.numerator / 2;
63             }
64             if (c.numerator > c.denominator)
65             {
66                 c.numerator = c.numerator / c.denominator;
67                 c.denominator = c.denominator / c.denominator;
68             }
69         }
70         return c;
71 }
```

simplify\_fraction is used in each computation function to simplify the return value's fraction:

```
89     return simplify_fraction(add); , 106     return simplify_fraction(subtract);
117     return simplify_fraction(multiply); , 128     return simplify_fraction(divide);
```

### Output

Fractions are now fully simplified, including the whole number result from the divided fraction!

```
ADDITION: 3/4
SUBTRACTION: 1/4
MULTIPLICATION: 1/8
DIVIDE: 2/1
```

### Lessons learned

I was unable to implement a function to make the denominators the same + multiply corresponding numerators. I may learn how to do this when I learn to return multiple values using address pointers!

**LAB 8: 8x06a – sizeof Person**

10/05/2022 – Tuesday, week 9

**Program aim**

Declare the given structure **Person**, then sort it into a different function called **Optimised\_Person** so that the **sizeof** function returned in **main** is a smaller memory value.

**Step-by-step plan**

1. Declare struct: **Person**, with its members sorted as stated in the exercise
2. Declare another struct: **Optimised\_Person**, with the same members as Person, but sorted by variable type: int variables > char variables > c-string variable...
3. Print the **sizeof** of each struct in main to compare the size in bytes of each structure.

**Implementation**

Source Code	Output/s / Testing
<pre> 1  #include &lt;stdio.h&gt; 2 3  struct Person 4  { 5      char first_initial; 6      char last_initial; 7      int age; 8      int weight; 9      char sex; 10     int height; 11     char blood_type[4]; 12     int shoe_type; 13 }; 14 15 struct Optimised_Person 16 { 17     int age; 18     int weight; 19     int height; 20     int shoe_type; 21     char first_initial; 22     char last_initial; 23     char sex; 24     char blood_type[4]; 25 }; 26 27 int main(void) 28 { 29     printf("Person size: %d \n", sizeof(struct Person)); 30     printf("Optimised_person size: %d", sizeof(struct Optimised_Person)); 31 32     return 0; 33 }</pre>	<pre> Person size: 28 Optimised_person size: 24 </pre>

**Lessons learned**

I learnt how to use the **sizeof()** function to find the size of structs in bytes.

I also learnt how reduce the size of structs via sorting their members by type.

**LAB 8: 8x07a – Array of Climate Data**

10/05/2022 – Tuesday, week 9

**Program aim**

for Auckland and Christchurch, create and store data as a **struct** array for record high, daily mean, and record low temperatures for a whole year (12 months = 12-length array). Write functions to: print climate data, print yearly climate data, compute average climate data, and print average climate. Output the data and data averages for each city in **main**.

\*Lastly, expand the program to add modular functionality...

**Implementation (I did not write pseudocode, I wrote code following the instructions as it was sufficient)**

Instructions from exercise	Source code																																																				
<p>Create a structure named <b>Climate_Data</b> which contains the following members:</p> <ul style="list-style-type: none"> <li>- Month (textual name)</li> <li>- Record High (in °C)</li> <li>- Daily Mean (in °C)</li> <li>- Record Low (In °C)</li> </ul>	<p>NOTE: I declared the degree symbol as a global variable</p> <pre> 1  #include &lt;stdio.h&gt; 2 3  const char DEGREE_SYMBOL = 248; // degrees ASCII char 4 5  struct Climate_Data 6  { 7      float rec_high; 8      float rec_low; 9      float daily_mean; 10     char month[10]; 11 }; </pre>																																																				
<p>In the main function, create an array of twelve <b>Climate_Data</b> structure variables, one for each month of the year, for the city of Auckland.</p> <p>Auckland's climate data is as follows:</p> <table border="1" data-bbox="80 1089 790 1170"> <thead> <tr> <th>Auckland</th><th>Jan</th><th>Feb</th><th>Mar</th><th>Apr</th><th>May</th><th>Jun</th><th>Jul</th><th>Aug</th><th>Sep</th><th>Oct</th><th>Nov</th><th>Dec</th></tr> </thead> <tbody> <tr> <td>Record High</td><td>30.0</td><td>30.5</td><td>29.8</td><td>26.0</td><td>24.6</td><td>23.8</td><td>19.0</td><td>20.6</td><td>22.0</td><td>23.6</td><td>25.9</td><td>28.3</td></tr> <tr> <td>Daily Mean</td><td>19.1</td><td>19.7</td><td>18.4</td><td>16.1</td><td>14.0</td><td>11.8</td><td>10.9</td><td>11.3</td><td>12.7</td><td>14.2</td><td>15.7</td><td>17.8</td></tr> <tr> <td>Record Low</td><td>5.6</td><td>8.7</td><td>6.6</td><td>3.9</td><td>0.9</td><td>-1.1</td><td>-3.9</td><td>-1.7</td><td>1.7</td><td>-0.6</td><td>4.4</td><td>7.0</td></tr> </tbody> </table> <p>...</p>	Auckland	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Record High	30.0	30.5	29.8	26.0	24.6	23.8	19.0	20.6	22.0	23.6	25.9	28.3	Daily Mean	19.1	19.7	18.4	16.1	14.0	11.8	10.9	11.3	12.7	14.2	15.7	17.8	Record Low	5.6	8.7	6.6	3.9	0.9	-1.1	-3.9	-1.7	1.7	-0.6	4.4	7.0	<pre> 25 int main(void) 26 { 27     struct Climate_Data auckland_data[12]; 28 29     sprintf(auckland_data[0].month, "January"); 30     auckland_data[0].rec_high = 30.0f; 31     auckland_data[0].daily_mean = 19.1f; 32     auckland_data[0].rec_low = 5.6f; 33 34     sprintf(auckland_data[1].month, "February"); 35     auckland_data[1].rec_high = 30.5f; 36     auckland_data[1].daily_mean = 19.7f; 37     auckland_data[1].rec_low = 8.7f; 38 39     sprintf(auckland_data[2].month, "March"); 40     auckland_data[2].rec_high = 29.8f; 41     auckland_data[2].daily_mean = 18.4f; 42     auckland_data[2].rec_low = 6.6f; 43 44     sprintf(auckland_data[3].month, "April"); 45     auckland_data[3].rec_high = 26.0f; 46     auckland_data[3].daily_mean = 16.1f; 47     auckland_data[3].rec_low = 3.9f; 48 49     sprintf(auckland_data[4].month, "May"); 50     auckland_data[4].rec_high = 24.6f; 51     auckland_data[4].daily_mean = 14.0f; 52     auckland_data[4].rec_low = 0.9f; 53 54     sprintf(auckland_data[5].month, "June"); 55     auckland_data[5].rec_high = 23.8f; 56     auckland_data[5].daily_mean = 11.8f; 57     auckland_data[5].rec_low = -1.1f; 58 59     sprintf(auckland_data[6].month, "June"); 60     auckland_data[6].rec_high = 19.0f; 61     auckland_data[6].daily_mean = 11.3f; 62     auckland_data[6].rec_low = -3.9f; 63 64     sprintf(auckland_data[7].month, "August"); 65     auckland_data[7].rec_high = 20.6f; 66     auckland_data[7].daily_mean = 11.3f; 67     auckland_data[7].rec_low = -1.7f; 68 69     sprintf(auckland_data[8].month, "September"); 70     auckland_data[8].rec_high = 22.0f; 71     auckland_data[8].daily_mean = 12.7f; 72     auckland_data[8].rec_low = 1.7f; 73 74     sprintf(auckland_data[9].month, "October"); 75     auckland_data[9].rec_high = 23.6f; 76     auckland_data[9].daily_mean = 14.2f; 77     auckland_data[9].rec_low = -0.6f; 78 79     sprintf(auckland_data[10].month, "November"); 80     auckland_data[10].rec_high = 25.9f; 81     auckland_data[10].daily_mean = 15.7f; 82     auckland_data[10].rec_low = 4.4f; 83 84     sprintf(auckland_data[11].month, "December"); 85     auckland_data[11].rec_high = 28.3f; 86     auckland_data[11].daily_mean = 17.8f; 87     auckland_data[11].rec_low = 7.0f; </pre>
Auckland	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec																																									
Record High	30.0	30.5	29.8	26.0	24.6	23.8	19.0	20.6	22.0	23.6	25.9	28.3																																									
Daily Mean	19.1	19.7	18.4	16.1	14.0	11.8	10.9	11.3	12.7	14.2	15.7	17.8																																									
Record Low	5.6	8.7	6.6	3.9	0.9	-1.1	-3.9	-1.7	1.7	-0.6	4.4	7.0																																									

**LAB 8: 8x07a – Array of Climate Data (continued)**

10/05/2022 – Tuesday, week 9

**Implementation (I did not write pseudocode, I wrote code following the instructions as it was sufficient)**

Instructions from exercise	Source code																																																				
<p>Write a function named <code>print_climate_data</code> which takes in a single <code>Climate_Data</code> structure variable by value, and prints out the following style of output:</p> <p><b>August</b></p> <ul style="list-style-type: none"> <li>- Record High: 20.6 °C</li> <li>- Daily Mean: 11.3 °C</li> <li>- Record Low: -1.7 °C</li> </ul>	<p>Function prototype</p> <pre>20 void print_climate_data(struct Climate_Data city);</pre> <p>Function definition</p> <pre>182 void print_climate_data(struct Climate_Data city) 183 { 184     printf("%s \n", city.month); 185     printf(" - Record High : %.1f %cC \n", city.rec_high, DEGREE_SYMBOL); 186     printf(" - Daily Mean : %.1f %cC \n", city.daily_mean, DEGREE_SYMBOL); 187     printf(" - Record Low : %.1f %cC \n", city.rec_low, DEGREE_SYMBOL); 188 }</pre>																																																				
<p>Write a function named <code>print_yearly_climate</code> which takes in an array of twelve <code>Climate_Data</code> structure variables, and then iterates through each the array, calling <code>print_climate_data</code> with each structure in the array.</p>	<p>Function prototype</p> <pre>21 void print_yearly_climate(struct Climate_Data city[]);</pre> <p>Function definition</p> <pre>177 void print_yearly_climate(struct Climate_Data city[]) 178 { 179     for (int i = 0; i &lt; 12; i++) 180     { 181         print_climate_data(city[i]); 182     } 183 }</pre>																																																				
<p>In the <code>main</code> function, create another array of twelve <code>Climate_Data</code> structure variables, one for each month of the year, for the city of Christchurch.</p>	<pre>89 struct Climate_Data christchurch_data[12]; 90 91 sprintf(christchurch_data[0].month, "January"); 92 christchurch_data[0].rec_high = 35.9f; 93 christchurch_data[0].daily_mean = 17.1f; 94 christchurch_data[0].rec_low = 3.6f; 95 96 sprintf(christchurch_data[1].month, "February"); 97 christchurch_data[1].rec_high = 40.0f; 98 christchurch_data[1].daily_mean = 16.8f; 99 christchurch_data[1].rec_low = 1.5f; 100 101 sprintf(christchurch_data[2].month, "March"); 102 christchurch_data[2].rec_high = 25.9f; 103 christchurch_data[2].daily_mean = 15.0f; 104 christchurch_data[2].rec_low = -0.2f; 105 106 sprintf(christchurch_data[3].month, "April"); 107 christchurch_data[3].rec_high = 29.9f; 108 christchurch_data[3].daily_mean = 12.1f; 109 christchurch_data[3].rec_low = -4.0f; 110 111 sprintf(christchurch_data[4].month, "May"); 112 christchurch_data[4].rec_high = 27.3f; 113 christchurch_data[4].daily_mean = 9.0f; 114 christchurch_data[4].rec_low = -6.4f; 115 116 sprintf(christchurch_data[5].month, "June"); 117 christchurch_data[5].rec_high = 22.5f; 118 christchurch_data[5].daily_mean = 6.2f; 119 christchurch_data[5].rec_low = -7.2f; 120 121 sprintf(christchurch_data[6].month, "July"); 122 christchurch_data[6].rec_high = 22.4f; 123 christchurch_data[6].daily_mean = 5.4f; 124 christchurch_data[6].rec_low = -6.8f; 125 126 sprintf(christchurch_data[7].month, "August"); 127 christchurch_data[7].rec_high = 22.8f; 128 christchurch_data[7].daily_mean = 7.1f; 129 christchurch_data[7].rec_low = -6.7f; 130 131 sprintf(christchurch_data[8].month, "September"); 132 christchurch_data[8].rec_high = 26.2f; 133 christchurch_data[8].daily_mean = 9.3f; 134 christchurch_data[8].rec_low = -4.4f; 135 136 sprintf(christchurch_data[9].month, "October"); 137 christchurch_data[9].rec_high = 30.1f; 138 christchurch_data[9].daily_mean = 11.5f; 139 christchurch_data[9].rec_low = -4.2f; 140 141 sprintf(christchurch_data[10].month, "November"); 142 christchurch_data[10].rec_high = 32.0f; 143 christchurch_data[10].daily_mean = 13.6f; 144 christchurch_data[10].rec_low = -2.6f; 145 146 sprintf(christchurch_data[11].month, "December"); 147 christchurch_data[11].rec_high = 36.0f; 148 christchurch_data[11].daily_mean = 15.7f; 149 christchurch_data[11].rec_low = 0.1f;</pre>																																																				
<p>Christchurch's climate data is as follows:</p> <table border="1" data-bbox="85 1192 774 1275"> <tr> <th>Auckland</th><th>Jan</th><th>Feb</th><th>Mar</th><th>Apr</th><th>May</th><th>Jun</th><th>Jul</th><th>Aug</th><th>Sep</th><th>Oct</th><th>Nov</th><th>Dec</th></tr> <tr> <td>Record High</td><td>30.0</td><td>30.5</td><td>29.8</td><td>26.0</td><td>24.6</td><td>23.8</td><td>19.0</td><td>20.6</td><td>22.0</td><td>23.6</td><td>25.9</td><td>28.3</td></tr> <tr> <td>Daily Mean</td><td>19.1</td><td>19.7</td><td>18.4</td><td>16.1</td><td>14.0</td><td>11.8</td><td>10.9</td><td>11.3</td><td>12.7</td><td>14.2</td><td>15.7</td><td>17.8</td></tr> <tr> <td>Record Low</td><td>5.6</td><td>8.7</td><td>6.6</td><td>3.9</td><td>0.9</td><td>-1.1</td><td>-3.9</td><td>-1.7</td><td>1.7</td><td>-0.6</td><td>4.4</td><td>7.0</td></tr> </table> <p>...</p>	Auckland	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Record High	30.0	30.5	29.8	26.0	24.6	23.8	19.0	20.6	22.0	23.6	25.9	28.3	Daily Mean	19.1	19.7	18.4	16.1	14.0	11.8	10.9	11.3	12.7	14.2	15.7	17.8	Record Low	5.6	8.7	6.6	3.9	0.9	-1.1	-3.9	-1.7	1.7	-0.6	4.4	7.0	
Auckland	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec																																									
Record High	30.0	30.5	29.8	26.0	24.6	23.8	19.0	20.6	22.0	23.6	25.9	28.3																																									
Daily Mean	19.1	19.7	18.4	16.1	14.0	11.8	10.9	11.3	12.7	14.2	15.7	17.8																																									
Record Low	5.6	8.7	6.6	3.9	0.9	-1.1	-3.9	-1.7	1.7	-0.6	4.4	7.0																																									

**LAB 8: 8x07a – Array of Climate Data (continued)**

10/05/2022 – Tuesday, week 9

**Implementation (I did not write pseudocode, I wrote code following the instructions as it was sufficient)**

Instructions from exercise	Source code / output
Call <code>print_yearly_climate</code> with the Christchurch's climate data.	<pre>163 print_yearly_climate(christchurch_data);</pre> <p style="text-align: center;"><b>OUTPUT OF CHRISTCHURCH'S CLIMATE DATA:</b></p> <pre> January - Record High : 35.9 °C - Daily Mean : 17.1 °C - Record Low : 3.6 °C February - Record High : 40.0 °C - Daily Mean : 16.8 °C - Record Low : 1.5 °C March - Record High : 25.9 °C - Daily Mean : 15.0 °C - Record Low : -0.2 °C April - Record High : 29.9 °C - Daily Mean : 12.1 °C - Record Low : -4.0 °C May - Record High : 27.3 °C - Daily Mean : 9.0 °C - Record Low : -6.4 °C June - Record High : 22.5 °C - Daily Mean : 6.2 °C - Record Low : -7.2 °C July - Record High : 22.4 °C - Daily Mean : 5.4 °C - Record Low : -6.8 °C August - Record High : 22.8 °C - Daily Mean : 7.1 °C - Record Low : -6.7 °C September - Record High : 26.2 °C - Daily Mean : 9.3 °C - Record Low : -4.4 °C October - Record High : 30.1 °C - Daily Mean : 11.5 °C - Record Low : -4.2 °C November - Record High : 32.0 °C - Daily Mean : 13.6 °C - Record Low : -2.6 °C December - Record High : 36.0 °C - Daily Mean : 15.7 °C - Record Low : 0.1 °C </pre>
Next, declare a structure named <code>Average_Climate_Data</code> which contains the following members: <ul style="list-style-type: none"> <li>- Average Record High (in °C)</li> <li>- Average Daily Mean (in °C)</li> <li>- Average Record Low (In °C)</li> </ul>	<pre>13 struct Average_Climate_Data 14 { 15     float average_rec_high; 16     float average_rec_low; 17     float average_daily_mean; 18 };</pre>

**LAB 8: 8x07a – Array of Climate Data (continued)**

10/05/2022 – Tuesday, week 9

Implementation (I did not write pseudocode, I wrote code following the instructions as it was sufficient)			
Instructions from exercise	Source code / output		
<p>Write a function named <b>compute_average_climate</b> which takes in an array of twelve <b>Climate_Data</b> structure variables and returns a single <b>Average_Climate_Data</b> structure containing the averages for the yearly record high, daily mean, and record low.</p>	<p>Function prototype</p> <pre>22   struct Average_Climate_Data compute_average_climate(struct Climate_Data city[]);</pre> <p>Function definition</p> <pre>185   struct Average_Climate_Data compute_average_climate(struct Climate_Data city[]) 186   { 187       struct Average_Climate_Data average_data; 188   189       average_data.average_rec_high = 0.0f; 190       average_data.average_daily_mean = 0.0f; 191       average_data.average_rec_low = 0.0f; 192   193       for (int i = 0; i &lt; 12; i++) 194       { 195           // SUM OF ALL MONTHS 196           average_data.average_rec_high = average_data.average_rec_high + city[i].rec_high; 197           average_data.average_daily_mean = average_data.average_daily_mean + city[i].daily_mean; 198           average_data.average_rec_low = average_data.average_rec_low + city[i].rec_low; 199   200       } 201       // DIVIDE TO FIND AVERAGE 202       average_data.average_rec_high = average_data.average_rec_high / 12; 203       average_data.average_daily_mean = average_data.average_daily_mean / 12; 204       average_data.average_rec_low = average_data.average_rec_low / 12; 205       // RETURN Average_Climate_Data STRUCTURE 206       return average_data; 207   }</pre>		
<p>Call <b>compute_average_climate</b> from the <b>main</b> function for both Auckland and Christchurch, storing the resulting structure as a local variable in main.</p>	<pre>151   // ASSIGN compute_average_climate to Average Climate Data of christchurch and auckland 152   struct Average_Climate_Data average_christchurch_data = compute_average_climate(christchurch_data); 153   struct Average_Climate_Data average_auckland_data = compute_average_climate(auckland_data);</pre>		
<p>Next, write a function named <b>print_average_climate_data</b> which takes in a single <b>Average_Climate_Data</b> structure variable by value, and prints in a style similar to <b>print_climate_data</b>.</p>	<pre>206   void print_average_climate_data(struct Average_Climate_Data city) 207   { 208       printf("Avarage Data \n"); 209       printf(" - Average Record High : %.1f °C \n", city.average_rec_high, DEGREE_SYMBOL); 210       printf(" - Average Daily Mean : %.1f °C \n", city.average_daily_mean, DEGREE_SYMBOL); 211       printf(" - Average Record Low : %.1f °C \n", city.average_rec_low, DEGREE_SYMBOL); 212   }</pre>		
<p>In <b>main</b>, call <b>print_average_climate_data</b> to output the computed averages for Auckland and Christchurch.</p>	<p>Calling <b>print_average_climate_data</b> in <b>main</b></p> <pre>157   print_average_climate_data(average_auckland_data); 162   print_average_climate_data(average_christchurch_data);</pre> <p>Outputs:</p> <table border="1"> <tr> <td style="vertical-align: top;"> <b>average_auckland_data</b>            Avarage Data            - Average Record High : 24.4 °C            - Average Daily Mean : 15.2 °C            - Average Record Low : 2.6 °C         </td> <td style="vertical-align: top;"> <b>average_christchurch_data</b>            Avarage Data            - Average Record High : 29.3 °C            - Average Daily Mean : 11.6 °C            - Average Record Low : -3.1 °C         </td> </tr> </table>	<b>average_auckland_data</b> Avarage Data - Average Record High : 24.4 °C - Average Daily Mean : 15.2 °C - Average Record Low : 2.6 °C	<b>average_christchurch_data</b> Avarage Data - Average Record High : 29.3 °C - Average Daily Mean : 11.6 °C - Average Record Low : -3.1 °C
<b>average_auckland_data</b> Avarage Data - Average Record High : 24.4 °C - Average Daily Mean : 15.2 °C - Average Record Low : 2.6 °C	<b>average_christchurch_data</b> Avarage Data - Average Record High : 29.3 °C - Average Daily Mean : 11.6 °C - Average Record Low : -3.1 °C		

**LAB 8: 8x07a – Array of Climate Data- FULL SOURCE CODE (1/3)**

10/05/2022 – Tuesday, week 9

```
#include <stdio.h>

const char DEGREE_SYMBOL = 248; // degrees ASCII char

struct Climate_Data
{
    float rec_high;
    float rec_low;
    float daily_mean;
    char month[10];
};

struct Average_Climate_Data
{
    float average_rec_high;
    float average_rec_low;
    float average_daily_mean;
};

void print_climate_data(struct Climate_Data city);
// use [] brackets when using Climate_Data array variables by element
void print_yearly_climate(struct Climate_Data city[]);
struct Average_Climate_Data compute_average_climate(struct Climate_Data city[]);
void print_average_climate_data(struct Average_Climate_Data city);

int main(void)
{
    struct Climate_Data auckland_data[12];

    sprintf(auckland_data[0].month, "January");
    auckland_data[0].rec_high = 30.0f;
    auckland_data[0].daily_mean = 19.1f;
    auckland_data[0].rec_low = 5.6f;

    sprintf(auckland_data[1].month, "February");
    auckland_data[1].rec_high = 30.5f;
    auckland_data[1].daily_mean = 19.7f;
    auckland_data[1].rec_low = 8.7f;

    sprintf(auckland_data[2].month, "March");
    auckland_data[2].rec_high = 29.8f;
    auckland_data[2].daily_mean = 18.4f;
    auckland_data[2].rec_low = 6.6f;

    sprintf(auckland_data[3].month, "April");
    auckland_data[3].rec_high = 26.0f;
    auckland_data[3].daily_mean = 16.1f;
    auckland_data[3].rec_low = 3.9f;

    sprintf(auckland_data[4].month, "May");
    auckland_data[4].rec_high = 24.6f;
    auckland_data[4].daily_mean = 14.0f;
    auckland_data[4].rec_low = 0.9f;

    sprintf(auckland_data[5].month, "June");
    auckland_data[5].rec_high = 23.8f;
    auckland_data[5].daily_mean = 11.8f;
    auckland_data[5].rec_low = -1.1f;

    sprintf(auckland_data[6].month, "July");
    auckland_data[6].rec_high = 19.0f;
    auckland_data[6].daily_mean = 11.8f;
    auckland_data[6].rec_low = -3.9f;

    sprintf(auckland_data[7].month, "August");
    auckland_data[7].rec_high = 20.6f;
    auckland_data[7].daily_mean = 11.3f;
    auckland_data[7].rec_low = -1.7f;

    sprintf(auckland_data[8].month, "September");
    auckland_data[8].rec_high = 22.0f;
    auckland_data[8].daily_mean = 12.7f;
    auckland_data[8].rec_low = 1.7f;

    sprintf(auckland_data[9].month, "October");
    auckland_data[9].rec_high = 22.0f;
    auckland_data[9].daily_mean = 14.2f;
    auckland_data[9].rec_low = -0.6f;

    sprintf(auckland_data[10].month, "November");
    auckland_data[10].rec_high = 22.0f;
    auckland_data[10].daily_mean = 15.7f;
    auckland_data[10].rec_low = 4.4f;

    sprintf(auckland_data[11].month, "December");
    auckland_data[11].rec_high = 22.0f;
    auckland_data[11].daily_mean = 17.8f;
    auckland_data[11].rec_low = 7.0f;
}

// Continued on page 2
```

**LAB 8: 8x07a – Array of Climate Data- FULL SOURCE CODE (2/3)**

10/05/2022 – Tuesday, week 9

```

struct Climate_Data christchurch_data[12];

sprintf(christchurch_data[0].month, "January");
christchurch_data[0].rec_high = 35.9f;
christchurch_data[0].daily_mean = 17.1f;
christchurch_data[0].rec_low = 3.6f;

sprintf(christchurch_data[1].month, "February");
christchurch_data[1].rec_high = 40.0f;
christchurch_data[1].daily_mean = 16.8f;
christchurch_data[1].rec_low = 1.5f;

sprintf(christchurch_data[2].month, "March");
christchurch_data[2].rec_high = 25.9f;
christchurch_data[2].daily_mean = 15.0f;
christchurch_data[2].rec_low = -0.2f;

sprintf(christchurch_data[3].month, "April");
christchurch_data[3].rec_high = 29.9f;
christchurch_data[3].daily_mean = 12.1f;
christchurch_data[3].rec_low = -4.0f;

sprintf(christchurch_data[4].month, "May");
christchurch_data[4].rec_high = 27.3f;
christchurch_data[4].daily_mean = 9.0f;
christchurch_data[4].rec_low = -6.4f;

sprintf(christchurch_data[5].month, "June");
christchurch_data[5].rec_high = 22.5f;
christchurch_data[5].daily_mean = 6.2f;
christchurch_data[5].rec_low = -7.2f;

sprintf(christchurch_data[6].month, "June");
christchurch_data[6].rec_high = 22.4f;
christchurch_data[6].daily_mean = 5.4f;
christchurch_data[6].rec_low = -6.8f;

sprintf(christchurch_data[7].month, "August");
christchurch_data[7].rec_high = 22.8f;
christchurch_data[7].daily_mean = 7.1f;
christchurch_data[7].rec_low = -6.7f;

sprintf(christchurch_data[8].month, "September");
christchurch_data[8].rec_high = 26.2f;
christchurch_data[8].daily_mean = 9.3f;
christchurch_data[8].rec_low = -4.4f;

sprintf(christchurch_data[9].month, "October");
christchurch_data[9].rec_high = 30.1f;
christchurch_data[9].daily_mean = 11.5f;
christchurch_data[9].rec_low = -4.2f;

sprintf(christchurch_data[10].month, "November");
christchurch_data[10].rec_high = 32.0f;
christchurch_data[10].daily_mean = 13.6f;
christchurch_data[10].rec_low = -2.6f;

sprintf(christchurch_data[11].month, "December");
christchurch_data[11].rec_high = 36.0f;
christchurch_data[11].daily_mean = 15.7f;
christchurch_data[11].rec_low = 0.1f;

// ASSIGN compute_average_climate to Average_Climate_Data of christchurch and auckland
struct Average_Climate_Data average_christchurch_data = compute_average_climate(christchurch_data);
struct Average_Climate_Data average_auckland_data = compute_average_climate(auckland_data);

// PRINT YEARLY CLIMATE & AVERAGE CLIMATE FOR EACH CITY DATA
print_yearly_climate(auckland_data);
print_average_climate_data(average_auckland_data);

printf("\n");

print_yearly_climate(christchurch_data);
print_average_climate_data(average_christchurch_data);

return 0;
}

void print_climate_data(struct Climate_Data city)
{
    printf("%s \n", city.month);
    printf(" - Record High : %.1f °C \n", city.rec_high, DEGREE_SYMBOL);
    printf(" - Daily Mean : %.1f °C \n", city.daily_mean, DEGREE_SYMBOL);
    printf(" - Record Low : %.1f °C \n", city.rec_low, DEGREE_SYMBOL);
}
// Continued on page 3

```

**LAB 8: 8x07a – Array of Climate Data- FULL SOURCE CODE (3/3)**

10/05/2022 – Tuesday, week 9

```

void print_yearly_climate(struct Climate_Data city[])
{
    // iterate through each element of the Climate_Data array variable
    for (int i = 0; i < 12; i++)
    {
        print_climate_data(city[i]);
    }
}

struct Average_Climate_Data compute_average_climate(struct Climate_Data city[])
{
    struct Average_Climate_Data average_data;

    average_data.average_rec_high = 0.0f;
    average_data.average_daily_mean = 0.0f;
    average_data.average_rec_low = 0.0f;

    for (int i = 0; i < 12; i++)
    {
        // SUM OF ALL MONTHS
        average_data.average_rec_high = average_data.average_rec_high + city[i].rec_high;
        average_data.average_daily_mean = average_data.average_daily_mean + city[i].daily_mean;
        average_data.average_rec_low = average_data.average_rec_low + city[i].rec_low;
    }
    // DIVIDE TO FIND AVERAGE
    average_data.average_rec_high = average_data.average_rec_high / 12;
    average_data.average_daily_mean = average_data.average_daily_mean / 12;
    average_data.average_rec_low = average_data.average_rec_low / 12;
    // RETURN Average_Climate_Data STRUCTURE
    return average_data;
}

void print_average_climate_data(struct Average_Climate_Data city)
{
    printf("Avarage Data \n");
    printf(" - Average Record High : %.1f °C \n", city.average_rec_high, DEGREE_SYMBOL);
    printf(" - Average Daily Mean : %.1f °C \n", city.average_daily_mean, DEGREE_SYMBOL);
    printf(" - Average Record Low : %.1f °C \n", city.average_rec_low, DEGREE_SYMBOL);
}

```

**OUTPUT:****Auckland data**

January	- Record High : 30.0 °C
	- Daily Mean : 19.1 °C
	- Record Low : 5.6 °C
February	- Record High : 30.5 °C
	- Daily Mean : 19.7 °C
	- Record Low : 8.7 °C
March	- Record High : 29.8 °C
	- Daily Mean : 18.4 °C
	- Record Low : 6.6 °C
April	- Record High : 26.0 °C
	- Daily Mean : 16.1 °C
	- Record Low : 3.9 °C
May	- Record High : 24.6 °C
	- Daily Mean : 14.0 °C
	- Record Low : 0.9 °C
June	- Record High : 23.8 °C
	- Daily Mean : 11.8 °C
	- Record Low : -1.1 °C
July	- Record High : 19.0 °C
	- Daily Mean : 11.8 °C
	- Record Low : -3.9 °C
August	- Record High : 20.6 °C
	- Daily Mean : 11.3 °C
	- Record Low : -1.7 °C
September	- Record High : 22.0 °C
	- Daily Mean : 12.7 °C
	- Record Low : 1.7 °C
October	- Record High : 22.0 °C
	- Daily Mean : 14.2 °C
	- Record Low : -0.6 °C
November	- Record High : 22.0 °C
	- Daily Mean : 15.7 °C
	- Record Low : 4.4 °C
December	- Record High : 22.0 °C
	- Daily Mean : 17.8 °C
	- Record Low : 7.0 °C
Average Data	- Average Record High : 24.4 °C
	- Average Daily Mean : 15.2 °C
	- Average Record Low : 2.6 °C

**Christchurch data**

January	- Record High : 35.9 °C
	- Daily Mean : 17.1 °C
	- Record Low : 3.6 °C
February	- Record High : 40.0 °C
	- Daily Mean : 16.8 °C
	- Record Low : 1.5 °C
March	- Record High : 25.9 °C
	- Daily Mean : 15.0 °C
	- Record Low : -0.2 °C
April	- Record High : 29.9 °C
	- Daily Mean : 12.1 °C
	- Record Low : -4.0 °C
May	- Record High : 27.3 °C
	- Daily Mean : 9.0 °C
	- Record Low : -6.4 °C
June	- Record High : 22.5 °C
	- Daily Mean : 6.2 °C
	- Record Low : -7.2 °C
July	- Record High : 22.4 °C
	- Daily Mean : 5.4 °C
	- Record Low : -6.8 °C
August	- Record High : 22.8 °C
	- Daily Mean : 7.1 °C
	- Record Low : -6.7 °C
September	- Record High : 26.2 °C
	- Daily Mean : 9.3 °C
	- Record Low : -4.4 °C
October	- Record High : 30.1 °C
	- Daily Mean : 11.5 °C
	- Record Low : -4.2 °C
November	- Record High : 32.0 °C
	- Daily Mean : 13.6 °C
	- Record Low : -2.6 °C
December	- Record High : 36.0 °C
	- Daily Mean : 15.7 °C
	- Record Low : 0.1 °C
Average Data	- Average Record High : 29.3 °C
	- Average Daily Mean : 11.6 °C
	- Average Record Low : -3.1 °C

## Week 9 lecture 026 Notes- Multi-Dimensional Arrays, main Parameters, Casting, void Pointers

11/05/2022- Wednesday, week 9

Notes Retrieved from Steffan Hooper (2022)

**Multi-dimensional arrays** involve: 2D arrays, 3D arrays, c-string arrays.

Syntax: `type array_name[dimension1] [dimension2]`

Example: `int table[20][10]`; has 200 integers as it is 20 x 10 large.

Accessing data example- look at it like: `table[x][y]`:

```
table[0][0] = 1; -> access 0,0  
table[0][1] = 2; -> access 0,1  
table[1][0] = 5; -> access 1,0...
```

2D arrays are useful for storing tabular data- like a tic-tac-toe board.

3D arrays: `int table[10][10][10]` -> 1000 elements in total!

You can add even more dimensions...

Arrays of `char*` pointers c-strings. Initialising multiple pointers:

```
char* words[] =  
{  
    "one",  
    "two"  
};  
printf("%s", words[1]); -> will print "two"
```

To access a single character: `words[1][1]` -> will access char "w" in "two"

... `printf("%s", text[demo[x][y]])` ... -> print text at a position in a table

c-string arrays can be declared alongside an `enum`. (ex. converting `enum` values to text)

Parameters of main: `int main(int argc, char* argv[])`

Count for the number command line arguments passed in program, address pointer to char (each element is c-string)

Programs can be run using cmd (command prompt). Arguments can be passed into main from cmd...

**Type casting:** explicit conversion- control whether a conversion can happen. Implicit: automatic conversion.

If total is an integer, it can be converted into a float for division:

```
float average = (float)total / element_count;  
an int value can be converted into a char value using type casting.
```

**void\*** pointers: a general-purpose pointer. can be used to store the address of any type of information (a generic address).

```
void* ptr = 0;
```

Use cast to tell program void pointer is pointing to a float: `*((float*)ptr)`

**LAB 8: 8x07a – Array of Climate Data (continued)**

11/05/2022 – Wednesday, week 9

**Program aim**

\*Finally, expand the program to add modular functionality

**Implementation (I did not write pseudocode, I wrote code following the instructions as it was sufficient)**

Instructions from exercise	Source code
Compute the highest, record high, daily mean and record low, for the year.	<p>Function Prototype</p> <pre>33   struct Climate_Data compute_highest_data(struct Climate_Data city[]);</pre>
<p>FOR REFERENCE:</p> <p>Christchurch's highest data</p> <p>Record high: 40.0</p> <p>Daily mean: 17.1</p> <p>Record low: 3.0</p> <p>*I REALISED I ACCIDENTALLY SET JAN'S RECORD LOW TO 3.6 INSTEAD OF 3.0, I CHANGED THIS VALUE BACK TO 3.0 AS I CREATED THE <code>compute_highest_data</code> FUNCTION</p>	<p>Function Definition</p> <pre>234 // Compute the highest, record high, daily mean and record low, for the year. 235 struct Climate_Data compute_highest_data(struct Climate_Data city[]) 236 { 237     struct Climate_Data highest_data; 238 239     sprintf(highest_data.month, "Highest Climate Data"); 240     highest_data.daily_mean = city[0].daily_mean; 241     highest_data.rec_high = city[0].rec_high; 242     highest_data.rec_low = city[0].rec_low; 243 244     for (int i = 0; i &lt; 12; ++i) 245     { 246         // replace highest_data variable with higher data if it's value is less than... 247         if (highest_data.daily_mean &lt; city[i].daily_mean) 248         { 249             highest_data.daily_mean = city[i].daily_mean; 250         } 251         if (highest_data.rec_high &lt; city[i].rec_high) 252         { 253             highest_data.rec_high = city[i].rec_high; 254         } 255         if (highest_data.rec_low &lt; city[i].rec_low) 256         { 257             highest_data.rec_low = city[i].rec_low; 258         } 259     } 260 261     return highest_data; 262 }</pre>
	<p>Function called in <code>main</code></p> <pre>176   struct Climate_Data highest_christchurch_data = compute_highest_data(christchurch_data); 177   print_climate_data(highest_christchurch_data);</pre> <p>Output (<code>christchurch_data</code>)</p> <pre>Highest Climate Data - Record High : 40.0 °C - Daily Mean : 17.1 °C - Record Low : 3.0 °C</pre>
<p>Compute the lowest, record high, daily mean and record low, for the year.</p> <p>FOR REFERENCE:</p> <p>Christchurch's lowest data</p> <p>Record high: 22.4</p> <p>Daily mean: 5.4</p> <p>Record low: -7.2</p>	<p>Function Prototype</p> <pre>35   struct Climate_Data compute_lowest_data(struct Climate_Data city[]);</pre> <p>Function Definition</p> <pre>264 // Compute the lowest, record high, daily mean and record low, for the year. 265 struct Climate_Data compute_lowest_data(struct Climate_Data city[]) 266 { 267     struct Climate_Data lowest_data; 268 269     sprintf(lowest_data.month, "Lowest Climate Data"); 270     lowest_data.daily_mean = city[0].daily_mean; 271     lowest_data.rec_high = city[0].rec_high; 272     lowest_data.rec_low = city[0].rec_low; 273 274     for (int i = 0; i &lt; 12; ++i) 275     { 276         // replace lowest_data variable with lower data if it's value is higher than... 277         if (lowest_data.daily_mean &gt; city[i].daily_mean) 278         { 279             lowest_data.daily_mean = city[i].daily_mean; 280         } 281         if (lowest_data.rec_high &gt; city[i].rec_high) 282         { 283             lowest_data.rec_high = city[i].rec_high; 284         } 285         if (lowest_data.rec_low &gt; city[i].rec_low) 286         { 287             lowest_data.rec_low = city[i].rec_low; 288         } 289     } 290 291     return lowest_data; 292 }</pre> <p>Function called in <code>main</code></p> <pre>180   struct Climate_Data lowest_christchurch_data = compute_lowest_data(christchurch_data); 181   print_climate_data(lowest_christchurch_data);</pre> <p>Output (<code>christchurch_data</code>)</p> <pre>Lowest Climate Data - Record High : 22.4 °C - Daily Mean : 5.4 °C - Record Low : -7.2 °C</pre>

**LAB 8: 8x07a – Array of Climate Data (continued)**

11/05/2022 – Wednesday, week 9

**Program aim**

\*Finally, expand the program to add modular functionality

**Implementation (I did not write pseudocode, I wrote code following the instructions as it was sufficient)**

Instructions from exercise	Source code
Compute the difference in temperature range between the record low and record high for each month of the year.	<p>Function Prototype</p> <pre>30   struct Climate_Data compute_difference_data(struct Climate_Data highest, struct Climate_Data lowest);</pre> <p>Function Definition</p> <pre>292   struct Climate_Data compute_difference_data(struct Climate_Data highest, struct Climate_Data lowest) 293   { 294       struct Climate_Data difference_data; 295   296       sprintf(difference_data.month, "Difference of Climate Data"); 297       difference_data.daily_mean = highest.daily_mean - lowest.daily_mean; 298       difference_data.rec_high = highest.rec_high - lowest.rec_high; 299       difference_data.rec_low = highest.rec_low - lowest.rec_low; 300   301       return difference_data; 302   }</pre> <p>Function called in <b>main</b></p> <pre>178       struct Climate_Data diff_christchurch_data; 179       diff_christchurch_data = compute_difference_data(highest_christchurch_data, lowest_christchurch_data); 180       print_climate_data(diff_christchurch_data);</pre> <div style="border: 1px solid black; padding: 5px; margin-left: 20px;"> <b>Output(christchurch_data)</b>  <b>Difference of Climate Data</b>        - Record High : 17.6 °C        - Daily Mean : 11.7 °C        - Record Low : 10.2 °C     </div>
Allow the user to enter the year's data for a city of their choosing at runtime.	<p>Function Prototype</p> <pre>32   void user_input_data(struct Climate_Data city[]);</pre> <p>Function Definition</p> <pre>346   // Allow the user to enter the year's data for a city of their choosing at runtime. 347   void user_input_data(struct Climate_Data city[]) 348   { 349       for (int i = 0; i &lt; 12; i++) 350       { 351           printf("%s \n", city[i].month); 352   353           printf("Record High : "); 354           scanf("%f", &amp;city[i].rec_high); 355   356           printf("Daily Mean : "); 357           scanf("%f", &amp;city[i].daily_mean); 358   359           printf("Record Low : "); 360           scanf("%f", &amp;city[i].rec_low); 361       } 362   }</pre> <p>Function called in <b>main</b> – <b>user_input_data(christchurch_data)</b> ;</p> <pre>191       char choice = '\0'; 192   193       printf("Christchurch or Auckland (C/A)? "); 194       scanf("%c", &amp;choice); 195   196       if (choice == 'c'    choice == 'C') 197       { 198           user_input_data(christchurch_data); 199           printf("\n"); 200           print_yearly_climate(christchurch_data); 201       } 202       else if (choice == 'a'    choice == 'A') 203       { 204           user_input_data(aukland_data); 205           printf("\n"); 206           print_yearly_climate(aukland_data); 207       }</pre>

-&gt;

I had to call the computation functions (average climate, highest data, lowest data, difference in data...) again so that the newly inputted data could be computed with their updated values.

**LAB 8: 8x07a – Array of Climate Data (continued)**

11/05/2022 – Wednesday, week 9

**Program aim**

\*Finally, expand the program to add modular functionality

**OUTPUT OF USER INPUT**

Print original Christchurch data	User choosing and inputting values for christchurch	Print user-inputted data for christchurch
<p>January</p> <ul style="list-style-type: none"> <li>- Record High : 30.0 °C</li> <li>- Daily Mean : 19.1 °C</li> <li>- Record Low : 5.6 °C</li> </ul> <p>February</p> <ul style="list-style-type: none"> <li>- Record High : 30.5 °C</li> <li>- Daily Mean : 19.7 °C</li> <li>- Record Low : 8.7 °C</li> </ul> <p>March</p> <ul style="list-style-type: none"> <li>- Record High : 29.8 °C</li> <li>- Daily Mean : 18.4 °C</li> <li>- Record Low : 6.6 °C</li> </ul> <p>April</p> <ul style="list-style-type: none"> <li>- Record High : 26.0 °C</li> <li>- Daily Mean : 16.1 °C</li> <li>- Record Low : 3.9 °C</li> </ul> <p>May</p> <ul style="list-style-type: none"> <li>- Record High : 24.6 °C</li> <li>- Daily Mean : 14.0 °C</li> <li>- Record Low : 0.9 °C</li> </ul> <p>June</p> <ul style="list-style-type: none"> <li>- Record High : 23.8 °C</li> <li>- Daily Mean : 11.8 °C</li> <li>- Record Low : -1.1 °C</li> </ul> <p>July</p> <ul style="list-style-type: none"> <li>- Record High : 19.0 °C</li> <li>- Daily Mean : 11.8 °C</li> <li>- Record Low : -3.9 °C</li> </ul> <p>August</p> <ul style="list-style-type: none"> <li>- Record High : 20.6 °C</li> <li>- Daily Mean : 11.3 °C</li> <li>- Record Low : -1.7 °C</li> </ul> <p>September</p> <ul style="list-style-type: none"> <li>- Record High : 22.0 °C</li> <li>- Daily Mean : 12.7 °C</li> <li>- Record Low : 1.7 °C</li> </ul> <p>October</p> <ul style="list-style-type: none"> <li>- Record High : 22.0 °C</li> <li>- Daily Mean : 14.2 °C</li> <li>- Record Low : -0.6 °C</li> </ul> <p>November</p> <ul style="list-style-type: none"> <li>- Record High : 22.0 °C</li> <li>- Daily Mean : 15.7 °C</li> <li>- Record Low : 4.4 °C</li> </ul> <p>December</p> <ul style="list-style-type: none"> <li>- Record High : 22.0 °C</li> <li>- Daily Mean : 17.8 °C</li> <li>- Record Low : 7.0 °C</li> </ul> <p>Average Data</p> <ul style="list-style-type: none"> <li>- Average Record High : 24.4 °C</li> <li>- Average Daily Mean : 15.2 °C</li> <li>- Average Record Low : 2.6 °C</li> </ul>	<p>Christchurch or Auckland (C/A)? c</p> <p>January</p> <ul style="list-style-type: none"> <li>- Record High : 9</li> <li>- Daily Mean : 6</li> <li>- Record Low : 2</li> </ul> <p>February</p> <ul style="list-style-type: none"> <li>- Record High : 8</li> <li>- Daily Mean : 5</li> <li>- Record Low : 2</li> </ul> <p>March</p> <ul style="list-style-type: none"> <li>- Record High : 9</li> <li>- Daily Mean : 6</li> <li>- Record Low : 1</li> </ul> <p>April</p> <ul style="list-style-type: none"> <li>- Record High : 9</li> <li>- Daily Mean : 5</li> <li>- Record Low : 1</li> </ul> <p>May</p> <ul style="list-style-type: none"> <li>- Record High : 8</li> <li>- Daily Mean : 5</li> <li>- Record Low : 1</li> </ul> <p>June</p> <ul style="list-style-type: none"> <li>- Record High : 9</li> <li>- Daily Mean : 5</li> <li>- Record Low : 1</li> </ul> <p>July</p> <ul style="list-style-type: none"> <li>- Record High : 9</li> <li>- Daily Mean : 5</li> <li>- Record Low : 1</li> </ul> <p>August</p> <ul style="list-style-type: none"> <li>- Record High : 10</li> <li>- Daily Mean : 8</li> <li>- Record Low : -10</li> </ul> <p>September</p> <ul style="list-style-type: none"> <li>- Record High : 8</li> <li>- Daily Mean : 5</li> <li>- Record Low : 1</li> </ul> <p>October</p> <ul style="list-style-type: none"> <li>- Record High : 8</li> <li>- Daily Mean : 5</li> <li>- Record Low : 1</li> </ul> <p>November</p> <ul style="list-style-type: none"> <li>- Record High : 9</li> <li>- Daily Mean : 5</li> <li>- Record Low : 1</li> </ul> <p>December</p> <ul style="list-style-type: none"> <li>- Record High : 9</li> <li>- Daily Mean : 6</li> <li>- Record Low : 1</li> </ul>	<p>January</p> <ul style="list-style-type: none"> <li>- Record High : 9.0 °C</li> <li>- Daily Mean : 6.0 °C</li> <li>- Record Low : 2.0 °C</li> </ul> <p>February</p> <ul style="list-style-type: none"> <li>- Record High : 8.0 °C</li> <li>- Daily Mean : 5.0 °C</li> <li>- Record Low : 2.0 °C</li> </ul> <p>March</p> <ul style="list-style-type: none"> <li>- Record High : 9.0 °C</li> <li>- Daily Mean : 6.0 °C</li> <li>- Record Low : 1.0 °C</li> </ul> <p>April</p> <ul style="list-style-type: none"> <li>- Record High : 9.0 °C</li> <li>- Daily Mean : 5.0 °C</li> <li>- Record Low : 1.0 °C</li> </ul> <p>May</p> <ul style="list-style-type: none"> <li>- Record High : 8.0 °C</li> <li>- Daily Mean : 5.0 °C</li> <li>- Record Low : 2.0 °C</li> </ul> <p>June</p> <ul style="list-style-type: none"> <li>- Record High : 9.0 °C</li> <li>- Daily Mean : 5.0 °C</li> <li>- Record Low : 1.0 °C</li> </ul> <p>July</p> <ul style="list-style-type: none"> <li>- Record High : 9.0 °C</li> <li>- Daily Mean : 5.0 °C</li> <li>- Record Low : 1.0 °C</li> </ul> <p>August</p> <ul style="list-style-type: none"> <li>- Record High : 10.0 °C</li> <li>- Daily Mean : 8.0 °C</li> <li>- Record Low : -10.0 °C</li> </ul> <p>September</p> <ul style="list-style-type: none"> <li>- Record High : 8.0 °C</li> <li>- Daily Mean : 5.0 °C</li> <li>- Record Low : 1.0 °C</li> </ul> <p>October</p> <ul style="list-style-type: none"> <li>- Record High : 8.0 °C</li> <li>- Daily Mean : 5.0 °C</li> <li>- Record Low : 1.0 °C</li> </ul> <p>November</p> <ul style="list-style-type: none"> <li>- Record High : 9.0 °C</li> <li>- Daily Mean : 5.0 °C</li> <li>- Record Low : 1.0 °C</li> </ul> <p>December</p> <ul style="list-style-type: none"> <li>- Record High : 9.0 °C</li> <li>- Daily Mean : 6.0 °C</li> <li>- Record Low : 1.0 °C</li> </ul>

**Print user-inputted data for christchurch (compute average, highest, lowest, difference...)**

```

211 |     average_christchurch_data = compute_average_climate(christchurch_data);
212 |     print_average_climate_data(average_christchurch_data);
213 |
214 |     highest_christchurch_data = compute_highest_data(christchurch_data);
215 |     lowest_christchurch_data = compute_lowest_data(christchurch_data);
216 |     diff_christchurch_data = compute_difference_data(highest_christchurch_data, lowest_christchurch_data);
217 |
218 |     print_climate_data(highest_christchurch_data);
219 |     print_climate_data(lowest_christchurch_data);
220 |     print_climate_data(diff_christchurch_data);

```

By calling the computation functions again... ^ (this can probably be modularised)

Average Data

- Average Record High : 8.8 °C
- Average Daily Mean : 5.5 °C
- Average Record Low : 0.3 °C

Highest Climate Data

- Record High : 10.0 °C
- Daily Mean : 8.0 °C
- Record Low : 2.0 °C

Lowest Climate Data

- Record High : 8.0 °C
- Daily Mean : 5.0 °C
- Record Low : -10.0 °C

Difference of Climate Data

- Record High : 2.0 °C
- Daily Mean : 3.0 °C
- Record Low : 12.0 °C

**LAB 8: 8x08a – Enumeration Animals**

11/05/2022 – Wednesday, week 9

**Program aim**

Using **enums** and functions, create a program that uses a function to output the ‘noise’ an animal makes by enumerating through an **enum** of different animals...

**Step-by-step plan**

- declare enumerated type named **Animal** with 10 animals in this order: **Antelope, Bat, Cat, Dog, Dolphin, Duck, Horse, Mouse, Owl, Snake**
- declare and define a function called **make\_animal\_noise()** that prints animal noises based on the corresponding animal. Use if-else statements to output the noise text for each **enum Animal** type.
- Call **make\_animal\_noise()** in main for two different **enum Animal** types.

**Implementation****Source Code****Output/s / Testing**

```

3   enum Animal
4   {
5     Antelope,
6     Bat,
7     Cat,
8     Dog,
9     Dolphin,
10    Duck,
11    Horse,
12    Mouse,
13    Owl,
14    Snake
15  };
16
17  void make_animal_noise(enum Animal current);
18
19  int main(void)
20  {
21    enum Animal doggo = Dog;
22    printf("Dog noise: ");
23    make_animal_noise(doggo);
24
25    printf("\n");
26
27    enum Animal ducky = Duck;
28    printf("Duck noise: ");
29    make_animal_noise(ducky);
30
31    return 0;
32  }
33
34  void make_animal_noise(enum Animal current)
35  {
36    if (current == Antelope)
37    {
38      printf("Snort");
39    }
34  else if (current == Bat)
41  {
42    printf("Screech");
43  }
44  else if (current == Cat)
45  {
46    printf("Meow");
47  }
48  else if (current == Dog)
49  {
50    printf("Woof");
51  }
52  else if (current == Dolphin)
53  {
54    printf("Click");
55  }
56  else if (current == Duck)
57  {
58    printf("Quack");
59  }
60  else if (current == Horse)
61  {
62    printf("Neigh");
63  }
64  else if (current == Mouse)
65  {
66    printf("Squeak");
67  }
68  else if (current == Owl)
69  {
70    printf("Hoot");
71  }
72  else if (current == Snake)
73  {
74    printf("Hiss");
75  }
76 }
```

Dog noise: Woof  
Duck noise: Quack

**Lessons learned**

I learnt how to implement enumerated types- each enum type kind-of functions like an array element...

## LAB 8: 8x09a – Refactor R-M-T

12/05/2022 – Thursday (lab), week 9

### Program aim

Given the Random-Match-Three program source code from a Week 7 lecture, improve the robustness of the `check_stars`, `check_risk` and `play` functions by refactoring them so that each only utilise a single return statement. Next, refactor the `check_stars` and `check_risk` functions such that they return an enumerated type value instead of a hardcoded magic number of 0 or 1.

Improve the overall modularity by ensuring the program code has: prototypes for each function and additional modular functions to divide up large functions.

### Implementation – refactor these functions so they only utilise a single return

Exercise Source Code	Changed Source Code
<pre> 60  int check_stars(int stars) 61  { 62      if (stars == 0) 63      { 64          printf("Oh no!!! You have run out of \"stars\"!!!\n"); 65          return 0; 66      } 67 68      return 1; 69  } </pre>	<p><code>check_stars</code> with a single return &amp; removed side effect</p> <pre> 61  int check_stars(int stars) 62  { 63      int check = 1; 64 65      if (stars == 0) 66      { 67          check = 0; 68      } 69 70      return check; 71  } </pre>
<pre> 76  int check_risk(int stars, int risk) 77  { 78      const int LIMIT = 100; 79      if (risk &gt; stars) 80      { 81          printf("That's more \"stars\" than you have! (%d stars)\n", stars); 82          return 1; 83      } 84      if (risk &gt; LIMIT) 85      { 86          printf("That's more \"stars\" than the round limit! "); 87          printf("(%d stars)\n", LIMIT); 88          return 1; 89      } 90      if (risk &lt; 0) 91      { 92          printf("You can't risk that number of \"Stars\"!!!\n"); 93          return 1; 94      } 95      return 0; 96  } </pre>	<p><code>check_risk</code> with a single return</p> <pre> 88  int check_risk(int stars, int risk) 89  { 90      const int LIMIT = 100; 91      int check = 0; 92 93      if (risk &gt; stars) 94      { 95          printf("That's more \"stars\" than you have! (%d stars)\n", stars); 96          check = 1; 97      } 98      if (risk &gt; LIMIT) 99      { 100         printf("That's more \"stars\" than the round limit! "); 101         printf("(%d stars)\n", LIMIT); 102         check = 1; 103     } 104     if (risk &lt; 0) 105     { 106         printf("You can't risk that number of \"Stars\"!!!\n"); 107         check = 1; 108     } 109 110     return check; </pre>
<p><code>play</code> ...</p> <pre> 112  int play(int risk) 113  { 114      int dice[3]; 115      dice[0] = get_random(1, 7); 116      dice[1] = get_random(1, 7); 117      dice[2] = get_random(1, 7); 118      printf("The magic happens... the dice roll...\n"); 119      Sleep(1000); 120      printf("The first die settles on %d...\n", dice[0]); 121      Sleep(1000); 122      printf("The second die settles on %d...\n", dice[1]); 123      Sleep(1000); 124      printf("The third die settles on %d...\n", dice[2]); 125 126 </pre>	<p><code>play</code> with a single return ... (new <code>reward</code> variable replaces repeated return statements) ...</p> <pre> 112  int play(int risk) 113  { 114      int dice[3]; 115      dice[0] = get_random(1, 7); 116      dice[1] = get_random(1, 7); 117      dice[2] = get_random(1, 7); 118      int reward = 0; // NEW VARIABLE 119      printf("The magic happens... the dice roll...\n"); 120      Sleep(1000); 121      printf("The first die settles on %d...\n", dice[0]); 122      Sleep(1000); 123      printf("The second die settles on %d...\n", dice[1]); 124      Sleep(1000); 125      printf("The third die settles on %d...\n", dice[2]); 126 </pre>

**LAB 8: 8x09a – Refactor R-M-T (continued)**

12/05/2022 – Thursday (lab), week 9

**Implementation – refactor these functions so they only utilise a single return**

Exercise Source Code	Changed Source Code
<pre style="font-family: monospace; font-size: 0.8em;"> ... Play  127 printf("... [%d]-[%d]-[%d] ...\n", dice[0], dice[1], dice[2]); 128 if (dice[0] == dice[1]) 129 { 130     if (dice[1] == dice[2]) 131     { 132         // Match three! 133         if (dice[0] == 7) 134         { 135             printf("Well done!!! Three sevens!!! "); 136             printf("Here's a big reward!!!\n"); 137             return risk * 11; 138         } 139         else 140         { 141             printf("Match three!!! Well done!!! "); 142             printf("Here's a big reward!!!\n"); 143             return risk * 11; 144         } 145     } 146     else 147     { 148         printf("Wow, a pair! Here's a small reward!\n"); 149         return risk * 4; 150     } 151 } 152 else 153 { 154     if (dice[0] == dice[2]) 155     { 156         printf("Look, a pair! Here's a small reward!\n"); 157         return risk * 4; 158     } 159     else if (dice[1] == dice[2]) 160     { 161         printf("Yes, a pair! Here's a small reward!\n"); 162         return risk * 4; 163     } 164     else 165     { 166         printf("No matches! Sorry!\n"); 167         return -risk; 168     } 169 } </pre>	<pre style="font-family: monospace; font-size: 0.8em;"> ... play with a single return  127 printf("... [%d]-[%d]-[%d] ...\n", dice[0], dice[1], dice[2]); 128 if (dice[0] == dice[1]) 129 { 130     if (dice[1] == dice[2]) 131     { 132         // Match three! 133         if (dice[0] == 7) 134         { 135             printf("Well done!!! Three sevens!!! "); 136             printf("Here's a big reward!!!\n"); 137             reward = risk * 11; 138         } 139         else 140         { 141             printf("Match three!!! Well done!!! "); 142             printf("Here's a big reward!!!\n"); 143             reward = risk * 11; 144         } 145     } 146     else 147     { 148         printf("Wow, a pair! Here's a small reward!\n"); 149         reward = risk * 4; 150     } 151 } 152 else 153 { 154     if (dice[0] == dice[2]) 155     { 156         printf("Look, a pair! Here's a small reward!\n"); 157         reward = risk * 4; 158     } 159     else if (dice[1] == dice[2]) 160     { 161         printf("Yes, a pair! Here's a small reward!\n"); 162         reward = risk * 4; 163     } 164     else 165     { 166         printf("No matches! Sorry!\n"); 167         reward = -risk; 168     } 169 } 170 171 return reward; 172 </pre>

**Implementation – refactor these functions so they return an enumerated type value instead of a hardcoded magic number of 0 or 1.****Check\_enum declaration**

```

6 enum Check_enum
7 {
8     do_check,
9     dont_check
10};

```

**do\_check** = 0 (invalid) and **dont\_check** = 1 (valid)

(FROM **main**) **check\_stars** and **check\_risk** functions need different **enum** variables because **check\_stars** initialises check with a value of 0 and returns 1 if an error is detected, but **check\_risk** initialises check with a value of 1, and returns 0 if an error is detected to reset the checking.

<pre style="font-family: monospace; font-size: 0.8em; margin: 0;"> 30 31 </pre>	<pre style="font-family: monospace; font-size: 0.8em; margin: 0;"> 30     enum Check_enum check_stars_enum = dont_check; 31     enum Check_enum check_risk_enum = do_check; </pre>
---	--

**LAB 8: 8x09a – Refactor R-M-T (continued)**

12/05/2022 – Thursday (lab), week 9

**Implementation – refactor these functions so they return an enumerated type value instead of a hardcoded magic number of 0 or 1.**

Old Source Code	Changed Source Code
<pre> <b>check_stars</b> 61 int check_stars(int stars) 62 { 63     int check = 1; 64 65     if (stars == 0) 66     { 67         check = 0; 68     } 69 70     return check; 71 }</pre>	<pre> <b>check_stars</b> with <b>enum</b> type return value 85 int check_stars(int stars, <b>enum Check_enum</b> check_stars_enum) 86 { 87     if (stars == 0) 88     { 89         check_stars_enum = do_check; 90     } 91 92     return check_stars_enum; 93 }</pre>
<pre> <b>check_risk</b> 88 int check_risk(int stars, int risk) 89 { 90     const int LIMIT = 100; 91     int check = 0; 92 93     if (risk &gt; stars) 94     { 95         printf("That's more \"stars\" than you have! (%d stars)\n", stars); 96         check = 1; 97     } 98     if (risk &gt; LIMIT) 99     { 100         printf("That's more \"stars\" than the round limit! "); 101         printf("(%d stars)\n", LIMIT); 102         check = 1; 103     } 104     if (risk &lt; 0) 105     { 106         printf("You can't risk that number of \"Stars\"!!!!\n"); 107         check = 1; 108     } 109 110     return check; 111 }</pre>	<pre> <b>check_risk</b> with <b>enum</b> type return value 95 int check_risk(int stars, int risk, <b>enum Check_enum</b> check_risk_enum) 96 { 97     const int LIMIT = 100; 98 99     if (risk &gt; stars) 100     { 101         printf("That's more \"stars\" than you have! (%d stars)\n", stars); 102         check_risk_enum = dont_check; 103     } 104     if (risk &gt; LIMIT) 105     { 106         printf("That's more \"stars\" than the round limit! "); 107         printf("(%d stars)\n", LIMIT); 108         check_risk_enum = dont_check; 109     } 110     if (risk &lt; 0) 111     { 112         printf("You can't risk that number of \"Stars\"!!!!\n"); 113         check_risk_enum = dont_check; 114     } 115 116 }</pre>

**Lessons learned**

I initially used a single **enum** variable for both **check\_stars** and **check\_risk** functions, which resulted in a problem for **check\_risk** because if the **enum** value in **main** is set to 1 and if the **check\_risk** function sets it to 1, it makes the function redundant because the check return value is unchanged!

**LAB 9: 9x01a – Intro to Pointers**

12/05/2022 – Thursday (lab), week 9

**Program aim – complete tasks in order**

Create a program that uses pointers. Follow the given exercise instructions in order...

**Program Design**

- Call `printf` with the output message: "Week 9: Intro to Pointers\n"
- Declare an integer variable named `my_age`.
- Assign a literal to the `my_age` variable which represents your actual age.
- Declare a pointer to an integer, called `my_pointer`.
- Assign the address of `my_age` to the pointer variable `my_pointer`.
- Using `printf`:
  - o Call `printf` once to display the contents of `my_age`.
  - o Call `printf` once to display the contents of `my_pointer`.
  - o Call `printf` once to display the value pointed to by `my_pointer`.
- Call `printf` again with the output message: "Indirection test!\n"
- Change the integer value pointed to by the pointer `my_pointer` to -1 (do this by dereferencing the pointer, and then assigning -1 to the value that is pointed to).
- Using `printf`:
  - o Call `printf` once to display the contents of `my_age`.
  - o Call `printf` once to display the contents of `my_pointer`.
  - o Call `printf` once to display the value pointed to by `my_pointer`

**Implementation**

Source Code	Output/s / Testing
<pre> 1 #include &lt;stdio.h&gt; 2 3 int main(void) 4 { 5     printf("Week 9: Intro to Pointers\n"); 6 7     int my_age = 0; 8     my_age = 18; 9 10    // declare pointer 11    int* my_pointer; 12 13    // assign value to pointer 14    my_pointer = &amp;my_age; 15 16    printf("my_age holds the value %d\n", my_age); 17    printf("my_pointer holds the value %d\n", *my_pointer); 18    printf("my_pointer points to the value %p\n", my_pointer); 19    printf("Indirection test!\n"); 20 21    *my_pointer = -1; // reassign pointer value 22 23    printf("my_age holds the value %d\n", my_age); 24    printf("my_pointer holds the value %d\n", *my_pointer); 25    printf("my_pointer points to the value %p\n", my_pointer); 26 27    return 0; 28 }</pre>	<p>First program output</p> <p>Week 9: Intro to Pointers  <code>my_age</code> holds the value 18  <code>my_pointer</code> holds the value 18  <code>my_pointer</code> points to the value 0135FA38    Indirection test!  <code>my_age</code> holds the value -1  <code>my_pointer</code> holds the value -1  <code>my_pointer</code> points to the value 0135FA38</p> <p>Second program output</p> <p>Week 9: Intro to Pointers  <code>my_age</code> holds the value 18  <code>my_pointer</code> holds the value 18  <code>my_pointer</code> points to the value 0073FDA4    Indirection test!  <code>my_age</code> holds the value -1  <code>my_pointer</code> holds the value -1  <code>my_pointer</code> points to the value 0073FDA4</p> <p>The address that <code>my_pointer</code> points to has changed on the second run of the program!</p>

**Lessons learned**

I learnt how to use pointers for the first time! I also learnt addresses change every time the program runs.

## LAB 9: 9x02a – float Exchange

12/05/2022 – Thursday (lab), week 9

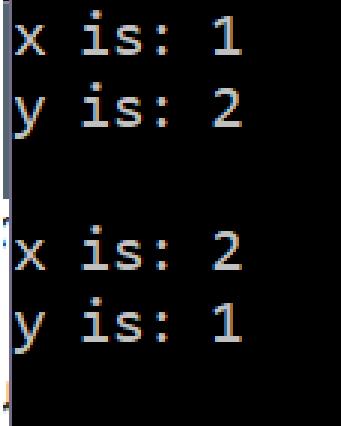
### Program aim

Use pointers and indirection to swap the values of 2 variables

### Step-by-step plan

1. Write a prototype for a function named exchange which takes two pointers to real numbers, **px** and **py**, as parameters, and does not return a value.
2. Write the function definition for the **exchange** function. The function must use indirection to swap the data **px** and **py** point to.
3. In the **main** function, declare two real number variables, **x** and **y**. Set **x** and **y** to appropriate test values, then print out the contents of the **x** and **y** variables. Next call exchange with the addresses of **x** and **y**. After the call, print out the contents of the **x** and **y** variables again.

### Implementation

Source Code	Output/s / Testing
<pre> 1  #include &lt;stdio.h&gt; 2 3  void exchange(int *px, int *py); 4 5  int main(void) 6  { 7      int x = 1; 8      int y = 2; 9 10     printf("x is: %d \n", x); 11     printf("y is: %d \n\n", y); 12 13     // use &amp; to get address 14     exchange(&amp;x, &amp;y); 15 16     printf("x is: %d \n", x); 17     printf("y is: %d \n", y); 18 19     return 0; 20 } 21 22 void exchange(int *px, int *py) 23 { 24     int temp = *px; 25     *px = *py; 26     *py = temp; 27 }</pre>	 <pre> x is: 1 y is: 2  x is: 2 y is: 1 </pre>

### Lessons learned

- I learnt how to use indirection to swap the values of 2 variables: which is assigning value #1 to a temporary variable, assigning value #1 to value #2, and assigning value #2 to the temporary variable.

## LAB 9: 9x03a – Multi-Dice Roll

12/05/2022 – Thursday (lab), week 9

### Program aim

Roll two randomly generated 6-sided die and note their addresses...

### Step-by-step plan

1. Define in **main** two local integer variables, called **dice1** and **dice2**.
2. Call **roll\_dice** from the **main** function, passing the address of the two local variables from the **main** function into the **roll\_dice** function.
3. Declare a third local variable in main, called **total\_roll**, and assign the result of the **roll\_dice** call to it.
4. Add **printf** calls to the program to achieve the output shown in the exercise...
5. All printed lines prefixed with **main** are generated by the **main** function, likewise the **roll\_dice** prefixed lines come from the **roll\_dice** function

### Implementation

#### Source Code

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  int roll_dice(int *address1, int *address2);
6
7  int main(void)
8  {
9      printf("main: Week9: Pass by Reference, with Pointers\n");
10     printf("main: Starting main function:\n");
11
12     int dice1 = 0;
13     int dice2 = 0;
14
15     printf("main: variable dice1 holds the value: %d\n", dice1);
16     printf("main: variable dice1 stored at: %p\n", &dice1);
17
18     printf("main: variable dice2 holds the value: %d\n", dice2);
19     printf("main: variable dice2 stored at: %p\n", &dice2);
20
21     printf("main: calling: roll_dice(%p, %p);\n", &dice1, &dice2);
22     int total_roll = roll_dice(&dice1, &dice2);
23
24     printf("main: variable dice1 holds the value: %d\n", dice1);
25     printf("main: variable dice1 stored at: %p\n", &dice1);
26
27     printf("main: variable dice2 holds the value: %d\n", dice2);
28     printf("main: variable dice2 stored at: %p\n", &dice2);
29
30     printf("main: variable total_roll holds the value: %d\n", total_roll);
31
32     printf("main: Returning zero to the operating system...\n");
33     return 0;
34 }
35
36 int roll_dice(int *address1, int *address2)
37 {
38     printf("roll_dice: Starting roll_dice function!\n");
39     printf("roll_dice: variable address1 holds the value: %p\n", address1);
40     printf("roll_dice: variable address2 holds the value: %p\n", address2);
41
42     srand(time(0));
43
44     printf("roll_dice: ROLLING TWO DICE!\n");
45
46     printf("roll_dice: Assigning first dice to caller's memory...\n");
47     *address1 = rand() % 6 + 1;
48
49     printf("roll_dice: Assigning second dice to caller's memory...\n");
50     *address2 = rand() % 6 + 1;
51
52     printf("roll_dice: Returning sum of two dice...\n");
53     return *address1 + *address2;
54 }
```

#### Output/s / Testing

```

main: Week9: Pass by Reference, with Pointers
main: Starting main function:
main: variable dice1 holds the value: 0
main: variable dice1 stored at: 008FF8F0
main: variable dice2 holds the value: 0
main: variable dice2 stored at: 008FF8E4
main: calling: roll_dice(008FF8F0, 008FF8E4);
roll_dice: Starting roll_dice function!
roll_dice: variable address1 holds the value: 008FF8F0
roll_dice: variable address2 holds the value: 008FF8E4
roll_dice: ROLLING TWO DICE!
roll_dice: Assigning first dice to caller's memory...
roll_dice: Assigning second dice to caller's memory...
roll_dice: Returning sum of two dice...
main: variable dice1 holds the value: 6
main: variable dice1 stored at: 008FF8F0
main: variable dice2 holds the value: 3
main: variable dice2 stored at: 008FF8E4
main: variable total_roll holds the value: 9
main: Returning zero to the operating system...
```

### Lessons learned

6. I learnt how to return the sum of two pointer values (using dereferencing) from a function to **main**

## LAB 9: 9x04a – Vec3 struct

12/05/2022 – Thursday (lab), week 9

### Program aim

Use **structs** and functions to compute the “dot product” of two 3-element vectors

### Step-by-step plan

1. Declare a structure named **Vec3**. Declare three members inside this structure, each of type float, named **x**, **y** and **z**.
2. In the main function, declare two **Vec3** structure variables, one named **vec\_a**, and the other named **vec\_b**.
3. Set the members of the two structures based on the given vector values from the exercise
4. Define function: **dot\_product** that computes the “dot product” of two 3-element vectors using the formula:  
 $v1 = < x_1, y_1, z_1 >$  and  $v2 = < x_2, y_2, z_2 >$   
 $\text{dot product} = v1 \cdot v2 = (x_1 \times x_2) + (y_1 \times y_2) + (z_1 \times z_2)$
5. In **main**, call **dot\_product**, passing in **vec\_a** and **vec\_b** as arguments.
6. In **main**, print out the result returned from the **dot\_product** call as a **float**

### Implementation

Source Code	Output/s / Testing
<pre> 1  #include &lt;stdio.h&gt; 2 3  struct Vec3 4  { 5      float x; 6      float y; 7      float z; 8  }; 9 10 float dot_product(struct Vec3 v1, struct Vec3 v2); 11 12 int main(void) 13 { 14     struct Vec3 vec_a; 15 16     vec_a.x = 1.5f; 17     vec_a.y = 2.5f; 18     vec_a.z = 3.5f; 19 20     struct Vec3 vec_b; 21 22     vec_b.x = 4.0f; 23     vec_b.y = 5.0f; 24     vec_b.z = 6.0f; 25 26     float result = dot_product(vec_a, vec_b); 27 28     printf("%f", result); 29 30     return 0; 31 } 32 33 float dot_product(struct Vec3 v1, struct Vec3 v2) 34 { 35     return ((v1.x*v2.x) + (v1.y*v2.y) + (v1.z*v2.z)); 36 }</pre>	 39.500000

### Lessons learned

- I learnt how to use a **struct** with a function to perform a computation!

## LAB 9: 9x05a - Vec3 Array

12/05/2022 – Thursday (lab), week 9

### Program aim

Use arrays and functions to compute the “dot product” of two 3-element vectors

### Step-by-step plan

1. In the **main** function, declare two float arrays of dimension 3, one named **vec\_a**, and the other named **vec\_b**.
2. Set the elements of the two arrays based on the given vector values from the exercise
3. Define function: **dot\_product** that computes the “dot product” of two 3-element vectors using the formula:  
 $v1 = < x1, y1, z1 >$  and  $v2 = < x2, y2, z2 >$   
 $\text{dot product} = v1 \cdot v2 = (x1 \times x2) + (y1 \times y2) + (z1 \times z2)$
4. In **main**, call **dot\_product**, passing in **vec\_a** and **vec\_b** as arguments.
5. In **main**, print out the result returned from the **dot\_product** call as a **float**

### Implementation

Source Code	Output/s / Testing
<pre> 1  #include &lt;stdio.h&gt; 2 3  float dot_product(float v1[3], float v2[3]); 4 5  int main(void) 6  { 7      float vec_a[4]; 8 9      vec_a[0] = 1.5f; 10     vec_a[1] = 2.5f; 11     vec_a[3] = 3.5f; 12 13     float vec_b[4]; 14 15     vec_b[0] = 4.0f; 16     vec_b[1] = 5.0f; 17     vec_b[3] = 6.0f; 18 19     float result = dot_product(vec_a, vec_b); 20 21     printf("%f", result); 22 23     return 0; 24 } 25 26 float dot_product(float v1[3], float v2[3]) 27 { 28     return ((v1[0]*v2[0]) + (v1[1]*v2[1]) + (v1[3]*v2[3])); 29 }</pre>	 39.500000

### Lessons learned

I find it easier to use **struct** when making these types of computations because when I am writing the math formula, it is easier to identify which number is which that I want to do a math operation on.

**LAB 9: 9x06a – Modularise Sort**

12/05/2022 – Thursday (lab), week 9

**Program aim**

Modularise the program from the given source code by creating new functions. Make it so the sorting algorithm works for all array sizes.

**Source Code given by exercise**

```

1 int main(void)
2 {
3     int data[7] = { 9, 2, 7, 1, 8, 4, 5 };
4     int temp = 0;
5     int j = 0;
6
7     for (int i = 1; i < 7; ++i)
8     {
9         temp = data[i];
10        j = i - 1;
11
12        while (temp < data[j] && j >= 0)
13        {
14            data[j + 1] = data[j];
15            j = j - 1;
16        }
17
18        data[j + 1] = temp;
19    }
20
21    return 0;
22 }
```

**Step-by-step plan**

1. Create a function named `sort_array`, which returns nothing, takes an `int` array via pointer, and the `size` of the array (`int`).
2. Call `sort_array` from `main`, passing in the local `data` array.
3. Add another function, named `print_array`, which returns nothing, and takes `int` array via pointer, and the `size` of the array.
4. In `print_array`, iterate though the array and print out the elements, comma separated, followed by a newline at the end.
5. Call `print_array` with the data array once before calling `sort_array`, and once again after calling `sort_array`.
6. Add at least three other `int` array definitions to main, initialise each array with a list of unsorted numbers.
7. from `main` pass each array into `print_array`, followed by `sort_array`, and finally `print_array` again.

**Implementation****Source Code (lines 1-45)**

```

1 #include <stdio.h>
2
3 void sort_array(int* data, int size);
4 void print_array(int* data, int size);
5
6 int main(void)
7 {
8     int data[7] = { 9, 2, 7, 1, 8, 4, 5 };
9     // 3 other array definitions
10    int data_a[5] = { 100, 34, 53, 5, 42 };
11    int data_b[10] = { 5, 9, 7, 2, 4, 1, 7, 5, 9, 10 };
12    int data_c[3] = { -7, -5, -9 };
13
14    printf("data: Unsorted array\n");
15    print_array(data, 7);
16    sort_array(data, 7);
17    printf("data: Sorted array\n");
18    print_array(data, 7);
19
20    printf("\n");
21
22    printf("data_a: Unsorted array\n");
23    print_array(data_a, 5);
24    sort_array(data_a, 5);
25    printf("data_a: Sorted array\n");
26    print_array(data_a, 5);
27
28    printf("\n");
29
30    printf("data_b: Unsorted array\n");
31    print_array(data_b, 10);
32    sort_array(data_b, 10);
33    printf("data_b: Sorted array\n");
34    print_array(data_b, 10);
35
36    printf("\n");
37
38    printf("data_c: Unsorted array\n");
39    print_array(data_c, 3);
40    sort_array(data_c, 3);
41    printf("data_c: Sorted array\n");
42    print_array(data_c, 3);
43
44    return 0;
45 }
```

**Source Code (lines 47-78)**

```

47 void sort_array(int* data, int size)
48 {
49     int temp = 0;
50     int j = 0;
51
52     for (int i = 1; i < size; ++i)
53     {
54         temp = data[i];
55         j = i - 1;
56
57         while (temp < data[j] && j >= 0)
58         {
59             data[j + 1] = data[j];
60             j = j - 1;
61
62         }
63
64         data[j + 1] = temp;
65     }
66
67 void print_array(int* data, int size)
68 {
69     for (int i = 0; i < size; i++)
70     {
71         printf("%d", data[i]);
72         if (i < (size - 1))
73         {
74             printf(", ");
75         }
76     }
77 }
78 }
```

## LAB 9: 9x06a – Modularise Sort (Continued)

12/05/2022 – Thursday (lab), week 9

### Output/s

```
data: Unsorted array  
9, 2, 7, 1, 8, 4, 5  
data: Sorted array  
1, 2, 4, 5, 7, 8, 9  
  
data_a: Unsorted array  
100, 34, 53, 5, 42  
data_a: Sorted array  
5, 34, 42, 53, 100  
  
data_b: Unsorted array  
5, 9, 7, 2, 4, 1, 7, 5, 9, 10  
data_b: Sorted array  
1, 2, 4, 5, 5, 7, 7, 9, 9, 10  
  
data_c: Unsorted array  
-7, -5, -9  
data_c: Sorted array  
-9, -7, -5
```

### Lessons learned

- I learnt how to create a modular program using functions that can take the size of an array as a parameter to allow an array of any size to be sorted/printed.

## Week 9 lecture 027 Notes- Dynamic Memory, The Heap, Memory Leaks, Recursive Composition

13/05/2022- Friday, week 9

Notes Retrieved from Steffan Hooper (2022)

**Dynamic memory:** `malloc` and `free` are functions used to allocate and deallocate memory. Utilise dynamic memory management. `malloc()` gives us memory (allocate) and `free()` gives the memory back (deallocate)- memory leaks occur if `free` isn't used.

We need to hard code memory size: `int data[10];` Remember: arrays require a constant for its size (error C2057). However, we can allocate memory at runtime using pointers to work around this.

**Stack frames** store every function call (local variables, parameters, return values).

**Heap** stores dynamic allocations at runtime (how much memory program needs while running).

Blocks of bytes allocated to `malloc` must be de-allocated using `free` after the program doesn't need this memory- otherwise a memory leak will happen.

Call `malloc` function using `void*`:

```
#include <stdlib.h>  
void malloc(size_t size);  
void free(void* ptr);
```

Week 9 lecture 027 Notes- Dynamic Memory, The Heap, Memory Leaks, Recursive Composition

(continued)

13/05/2022- Friday, week 9

Notes Retrieved from Steffan Hooper (2022)

```
void* p_allocation = malloc(sizeof(int)); - void pointer can make int variable read as  
an int
```

```
Allocate memory: p_heap_array = malloc(sizeof(int) * 5);
```

If `malloc` returns a null pointer, that means there is not enough ram to compute things...

```
if (p_i != 0) -> check if address is valid (address not 0)  
{  
    *p_i = 17; -> safely dereference address if pointer is not null  
}  
else ... (handle the error)
```

Arrays are treated like pointers: declaring `int ages[]` is the same as `int* ages...`

**Dynamic memory allocation (dynamic array size):** instead of allocating fixed sized arrays at compile time, you can allocate the space to store a pointer, then allocate the array at runtime using `malloc()`. This allows the user to enter any number of items memory can allow.

**DYNAMIC ARRAY SIZE:**

```
scanf("%d", size); -> user inputs array  
int* p_array = malloc(sizeof(int) * size); -> allocate user-inputted size to the array  
...  
free(p_array); -> free memory after using it!
```

#include <crtdbg.h> - allows user to detect memory leaks. Only returns detection result when the program ends (`main's return 0;`). Have memory leak detection on all the time when working with allocating memory!

```
Call: _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
```

Structures can be allocated on the heap using `malloc()`.

Heap structures may be returned (which also needs to be freed).

Recursive composition: when a part of something is made of itself.

Data structure: binary tree

# WEEK 10

## Week 10 lecture 028 Notes- **typedef**, Testing, VS Memory Patterns, Warnings, P1 Colour Console

15/05/2022- Monday, week 10

Notes Retrieved from Steffan Hooper (2022)

**typedef keyword:** creates a Type alias / “synonym”. This allows programmers to write less...  
**typedef struct tag\_point -> tag\_point** needs a underscore/different identifier than “Point”  
{  
    int x;  
    int y;  
} **Point;**  
...  
**Point top\_left;** ...

**Type identifier.** Typedef makes it so we don’t have to type “struct” when calling the struct.

**typedef** can also be used with non-structure types, such as **enum**.

**Testing:** be sure to test with a variety of inputs, instead of limiting myself to a few inputs.

6. Peer review: looking at other’s code and pointing out problems (not running program)
7. Walkthrough: criticising a program as it is running- critique functionality
8. Validation: testing of modules and if functionality behaves as expected. Can be automated.  
(Feedback mode from early exercises is an example of this)

**stubs:** a temporary piece of code that exists to allow the program to compile. This can be used to: return dummy data, not do anything, or trigger an **assert** to notify the programmer of incomplete functionality. COMMENT WHEN YOU ARE DOING STUBS. // **TODO: Stub**

**Instrumentation tracing:** Tracing: a debug log traces activity during the program’s execution, including errors and warnings. It can also measure the performance: time taken to execute a module; memory used... tracing can be outputted to the screen, or to a file.

**assert()** can be used to stop/continue a program, such as checking for a null pointer and stopping program if it is. **assert** is usually used with **malloc** to ensure memory allocation is succeeding.

**DEBUG MEMORY PATTERNS (right click > hexadecimal display):**

uninitialized variable memory: 0xffffffff

“clean” malloc’d variable memory: 0xCDCDCDCD

Freed (deallocated) variable memory: 0xFFFFFFFF

Null pointer prevents dead memory / bad memory.

Treat compiler warnings as errors. Compiler settings has warning level tolerance. Raising warning levels helps to detect potential problems with source code and leads to better quality code. To change warning level: right click project > properties > general > warning level.

**p1colour.h** and **p1colour.lib**, “P1 Colour Console” library : sets text colour and moves cursor in console. install the .h and .lib files and extract them to project directory to use these functions:  
**void move\_cursor\_to(int column, int row);**  
**void set\_text\_colour(int foreground, int background);**  
use **#include "p1colour.h"**

**Week 10 lecture 029 Notes- Function Pointers, Callbacks, QSort, BSearch**

**18/05/2022- Wednesday, week 10**

Notes Retrieved from Steffan Hooper (2022)

Function pointers: to return address of a function, simply use its name WITH NO CALL BRACKETS: **function\_name** or name with ampersand **&function\_name** using **%p** to print it. I can even print **main**'s address.

To view assembly code of program: trigger breakpoint > right click text editor > go to disassembly

Storing function addresses using pointer:

**void\* function\_pointer = (void \*)function\_name;** USES VOID POINTER: STORES ADDRESS NOT CONTENT OF FUNCTION

Calling functions via pointer:

DECLARE FUNCTION POINTER: **void(\*function\_pointer)(void) = function\_name;**  
CALL FUNCTION POINTER: **(\*function\_pointer)(); OR function\_pointer();**

Cast operator general form: **(return type (\*) (argument types))function pointer**

Declare function pointers in one line: **(\* (void (\*) (void)function\_pointer)();**

OR: **((void (\*) (void)function\_pointer)();**

**typedef** can be used with Function pointers to make syntax easier to understand.

DECLARE TYPE GLOBALLY: **typedef void(\*type\_function\_pointer)(void);**

DECLARE IN MAIN: **type\_function\_pointer function\_pointer = function\_name;**

CALL FUNCTION: **function\_pointer();**

Callbacks: used when program needs to keep working until interrupted (parallel task, user input).

**stdlib.h qsort**: **qsort(array to be sorted, array size, sizeof(array), comparative function)** the comparative function is a function pointer passed for callback. **qsort** can be used to sort c-strings.

**stdlib.h bsearch**: binary search must use an array in ASCENDING ORDER. Returns a pointer to an element in an array if found, otherwise a null pointer.

Functional Abstraction: we can turn for loops into a function using functional abstraction.

## **Feedback on Midsemester Exam**

**19/05/2022 – Thursday (lab), week 10**

- **const** variables typically start with capital
- Chars (c-strings) don't need assignment with curly brackets...
- **A simpler for question 6:**  
The Depth – row index – 1  
Hashtag = row index + 1
- Use nested loops, height, then nested inside height is the body (right angle triangle for loop, spaces for loop, second right angle triangle for loop)

### **Note for creating my own program:**

disable these warnings: properties -> C/C++ -> Advanced -> disable specific warnings

**4668 ; 4710 ; 4711 ; 5045 ; 4820 ; 4242 ; 4244**

**LAB 10: 10x01a – Using 2D Arrays**

19/05/2022 – Thursday (lab), week 10

**Program aim**

Complete the given instructions in the TODO statements

**TODO plan**

```

1 #define _CRT_SECURE_NO_WARNINGS
2 #include <stdio.h>
3
4 int main(void)
5 {
6     // TODO: use nested loop to create 2D array values
7
8     // TODO: use nested loop to print 2D array
9
10    return 0;
11 }
```

**Error/debugging**

```

8 // TODO: use nested loop to create 2D array values
9
10 int count = 1;
11
12 while (count <= 25)
13 {
14     for (int x = 0; x < 5; x++)
15     {
16         for (int y = 0; y < 5; y++)
17         {
18             square_array[x][y] = count;
19         }
20     }
21     count++;
22 }
```

```

25 25 25 25 25
25 25 25 25 25
25 25 25 25 25
25 25 25 25 25
25 25 25 25 25
```

I was confused as to why the output was only the last number in the count. I tested and found that the printing was correct (by inputting the values individually with `scanf` instead of assigning with loop), so it was the assignment loop's `count` with the problem.

The solution I found was to place `count++`; in the `y for` loop instead of the end of the `count while` loop.

**Implementation****Source Code****Output/s / Testing**

```

1 #define _CRT_SECURE_NO_WARNINGS
2 #include <stdio.h>
3
4 int main(void)
5 {
6     int square_array[5][5];
7
8     // TODO: use nested loop to create 2D array values
9
10    int count = 1;
11
12    while (count <= 25)
13    {
14        for (int x = 0; x < 5; x++)
15        {
16            for (int y = 0; y < 5; y++)
17            {
18                square_array[x][y] = count;
19            }
20        }
21        count++;
22    }
23
24    // TODO: use nested loop to print 2D array
25
26    for (int x = 0; x < 5; ++x)
27    {
28        for (int y = 0; y < 5; ++y)
29        {
30            printf("%3d", square_array[x][y]);
31        }
32        printf("\n");
33    }
34
35    return 0;
36 }
```

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

**Lessons learned**

I think the reason as to why it only printed '25' instead of the count from 1-25 was that the count loop was executed first, counting 1 to 25 and finishing, then moving on to the `x` and `y` assignment loops where count was an unchanged value of 25, and it was the only value assigned to `square_array`.

**LAB 10: 10x02a – Using 2D char Array**

19/05/2022 – Thursday (lab), week 10

**Program aim**

Complete the given source code using a 2D character array to count the number of digits in the C-String passed into the `count_digits` function.

**Implementation****Given Source Code by Exercise**

```

1 #define _CRT_SECURE_NO_WARNINGS
2 #include <stdio.h>
3
4 int count_digits(char* text);
5
6 int main(void)
7 {
8     char* test[] =
9     {
10         "H3ll0",           // 2 digits
11         "Qu3sti0n 3...", // 3 digits
12         "0ld Pr4ctic4l T3st", // 4 digits
13         "is",             // 0 digits
14         "e4sy!"           // 1 digit
15     };
16
17     for (int k = 0; k < 5; ++k)
18     {
19         printf("%s : ", test[k]);
20         printf("%d digits\n", count_digits(test[k]));
21     }
22
23     return 0;
24 }
25
26 int count_digits(char* text)
27 {
28     // TODO: Insert your code here...
29     // TODO: Temporary stub return...
30     return 0;
31 }
```

**New Source Code**

```

4     int count_digits(char* text);
5
6     int main(void)
7     {
8         char* test[] =
9         {
10             "H3ll0",           // 2 digits
11             "Qu3sti0n 3...", // 3 digits
12             "0ld Pr4ctic4l T3st", // 4 digits
13             "is",             // 0 digits
14             "e4sy!"           // 1 digit
15         };
16
17         for (int k = 0; k < 5; ++k)
18         {
19             printf("%s : ", test[k]);
20             printf("%d digits\n", count_digits(test[k]));
21         }
22
23         return 0;
24     }
25
26     int count_digits(char* text)
27     {
28         int digits = 0;
29
30         // TODO: Insert your code here...
31         for (int i = 0; text[i] != '\0'; i++)
32         {
33             if (text[i] == '0' || text[i] == '1' || text[i] == '2' || text[i] == '3' ||
34                 text[i] == '4' || text[i] == '5' || text[i] == '6' || text[i] == '7' ||
35                 text[i] == '8' || text[i] == '9')
36             {
37                 digits++;
38             }
39         }
40
41         return digits;
42     }

```

**Lessons learned**

- I learnt how to use a 2D character array for the first time!
- Note: 2D character arrays (`char*`) are arrays that store multiple c-strings as elements, as opposed to character arrays (`char`) that store a single c-string, with each single character being an element.

**LAB 10: 10x03a – Detecting mains Args**

19/05/2022 – Thursday (lab), week 10

**Program aim**

Write a program that processes the arguments passed into **main** from the command line. The program must print out the number of arguments passed into the program when executed, followed by each argument on a new line, for example: **argv[1] hello!**

**Implementation****Source Code****Output/s / Testing**

```

1 #define _CRT_SECURE_NO_WARNINGS
2 #include <stdio.h>
3
4 // ARGUMENTS DEFINED
5 // argc: count of number of command line arguments passes to program
6 // argv: stores address of a pointer to a char. each element in argv is c-string
7
8 int main(int argc, char* argv[])
9 {
10     printf("Arguments at execution: %d\n", argc);
11
12     for (int i = 0; i < argc; ++i)
13     {
14         printf("argv[%d]: %s", i, argv[i]);
15         printf("\n");
16     }
17
18     return 0;
19 }
```

```

argv[1]: Testing!
argv[2]: hello
argv[3]: my
argv[4]: name
argv[5]: is
argv[6]: Fran

```

**Lessons learned**

- The arguments must be passed through the properties settings of the solution
- I learnt how to write a program that processes the arguments passed into main from the command line for the first time

**Week 10 lecture 030 Notes- The Pre-Processor, Assert, Build Targets, Function Comments**

**18/05/2022- Wednesday, week 10**

Notes Retrieved from Steffan Hooper (2022)

**The pre-processor**

**#include** (is a directive) allows external code to be included in the program. Ex.

**stdio.h** is a file containing function prototypes for **printf**, **scanf**, **sprintf**...

**Header files** are files that contain declarations. These are shared across multiple source files (.c)

**#define** (is a directive) creates a macro, token-string substitution (text replacement)

**Intermediate files** contain the output of the pre-processor stage. Can be a .i plain text file. Right click project > properties > pre-processor > pre-process to file. Contains function prototypes from libraries used in **#include** plus source code.

**Macro:** **#define** creates a pre-processor Macro.

**Function-like Macro:** **#define RADTODEG(x) ((x) \* 57.29578f)**

**Calling Macro:** **float degrees = RADTODEG(radians);**

9. Macros are UPPERCASE, functions are not.

10. **Replacement text**

11. Always put brackets around variables being operated with in Macros

ALWAYS INCLUDE **#define \_CRT\_SECURE\_NO\_WARNINGS** TO ALLOW USE OF SCANF.  
Can also be done in properties settings (but don't do both).

**Conditional Compilation:** choosing parts of a program to compile depending on what Macro tokens are defined. **#ifdef** tests if a Macro is defined. End with **#endif**.

Define macro 1

Define macro 2

...

**#ifdef** macro 1

  execute action

**#endif**

**#ifdef** macro 2

  Execute action

**#endif**

**Built-in pre-processor directives:** **\_FILE\_**, **\_LINE\_**, **\_DATE\_**, **\_TIME\_** ...

Tokens can be stringified and concatenated...

**Week 10 lecture 030 Notes- The Pre-Processor, Assert, Build Targets, Function Comments**

**(continued)**

**18/05/2022- Wednesday, week 10**

Notes Retrieved from Steffan Hooper (2022)

Assert: `#include <assert.h>` checks if program makes sense (sanity check). If it does, evaluates to true and program keeps running. If it evaluates to false, triggers breakpoint (debug error window appears).

Ex. Check if `x` is 10:

```
int x = 10; ...
assert(x == 10); -> if true, program keeps running
```

Assert is used in:

Pre-conditions: something that must be always true prior to code

Post-conditions: something that must always be true after some code execution

Assert can be used with Pointers (check if pointer is not null)

Asserts only check things, not modify things! Don't call functions inside asserts.

Build targets: debug vs release. **asserts** are removed from the Release build, so none of the sanity checks and accidental side effects will be included in the Release build

Coding standards: Function comments: Comment the name of the function, what it does, inputs, returns, pre and post conditions

# WEEK 11

## Week 11 lecture 031 Notes- Bitwise Operators

23/05/2022- Monday, week 11

Notes Retrieved from Steffan Hooper (2022)

Bitwise operators: used to access individual bits of a variable. (Not same as logical operators)

Bitwise NOT: ~

```
int x = 26;  
int y = ~x; <- y stores -27
```

Bit flipping: swap state of a bit from true to false vice versa. Bit flipping can change values ('a' to 'A')

Bitwise AND: &

```
int x = 26;  
int y = 15;  
int z = x & y; <- z stores 10
```

Bitwise OR: |

```
int x = 26;  
int y = 15;  
int z = x | y; <- z stores 31
```

Bitwise XOR: ^

```
int x = 26;  
int y = 15;  
int z = x ^ y; <- z stores 21
```

declaring unsigned char: `unsigned char byte = '\0';`

Bit shifting: shift bits in memory left/right

Shift left: <<

```
int x = 10;  
int y = x << 1; <- y stores 20
```

Shift right: >>

```
int x = 10;  
int y = x >> 1; <- y stores 5
```

RGB colours are between values 0 to 255. Ex. 0xff stores  $255_{10}$

REMEMBER PADDING:

`printf("%08x\n", xrgb);` -> "08" gives an 8-length output, pads the left with 0's

`xrgb = xrgb | (green << 8);` -> shifting bits prevents override of `xrgb`

Compound operators (assignment): &=, |=, ^=, <<=, >>=

Bit flags: pack multiple 'ideas' into a single variable.

```
int alive = 1;  
int tired = 2;  
  
int student_status = alive | tired; -> unique value generated as a result of alive OR tired  
Bit masking: detects which bits are set in a variable  
if (ta_status & thirsty) // NON ZERO RESULT IS TRUE  
{  
    // then TA is thirsty  
}
```

## Week 11 lecture 031 Notes- Bitwise Operators (continued)

23/05/2022- Monday, week 11

Notes Retrieved from Steffan Hooper (2022)

Passing bit flags into functions: `void print_status(int status) <- pass enumeration of bit flags into function (as powers of 2 / unique values)`

Decode in `print_status` function (if statements), encode in `main`:  
`print_status(ALIVE | TIRED | HAPPY);`

Memory leak detection (`crtdbg.h`) `_CrtSetDbgFlag` parameter used **bit flags**:  
`_CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);`

Storing floats: IEEE-754 single-precision floating-point. Floats are 32-bits, 4 bytes large.

Bit 31: Sign bit (1 is negative, 0 is positive)

Bit 30-23: exponent bits (convert to base-10 and subtract 127 to get exponent)

Bit 22-0: mantissa bits (each bit represents  $\frac{1}{2}$ ,  $\frac{1}{4}$ ,  $1/8$ ,  $1/16$ ,  $1/32$  ...) each bit is summed

Compute real number value: `sign * (2 ^ exponent) * mantissa`

**LAB 8: 8x10a – Rock-Paper-Scissors**

23/05/2022 – Monday, week 11

**Program aim**

Create a rock-paper-scissors game. Provide the option to replay or quit. Keep a tally of rounds, number of wins and losses, and number of ties. Display these details when the program ends.

**Implementation****main & enum & structs & prototypes  
Source Code lines 5-45**

```

5  enum Rock_paper_scissors
6  {
7      NOTHING,
8      ROCK,
9      PAPER,
10     SCISSORS
11 };
12
13 struct Game_details
14 {
15     int rounds_played;
16     int human_wins;
17     int computer_wins;
18     int ties;
19 };
20
21 enum Rock_paper_scissors assign_user_choice(char choice);
22 enum Rock_paper_scissors gen_comp_choice(void);
23 void print_choice(enum Rock_paper_scissors choice);
24 void print_game_details(struct Game_details game);
25
26 int main(void)
27 {
28     srand(time(0));
29
30     char char_input = '\0';
31     char play_again = '\0';
32     enum Rock_paper_scissors choice = NOTHING;
33     enum Rock_paper_scissors comp_choice = NOTHING;
34     struct Game_details current_game;
35
36     current_game.rounds_played = 0;
37     current_game.human_wins = 0;
38     current_game.computer_wins = 0;
39     current_game.ties = 0;
40
41     printf("=====\\n");
42     printf("|| ROCK PAPER SCISSORS ||\\n");
43     printf("=====\\n");
44
45     printf("\\n");

```

**main Source Code lines 5-45**

```

47     do
48     {
49         printf("Rock, paper, or scissors (r/p/s)? ");
50         scanf(" %c", &char_input);
51
52         choice = assign_user_choice(char_input);
53         printf("Human player ");
54         print_choice(choice);
55
56         printf("\\n");
57
58         comp_choice = gen_comp_choice();
59
60         printf("Computer ");
61         print_choice(comp_choice);
62
63         printf("\\n");
64
65         // TODO: make into a function
66
67         if (choice == comp_choice)
68         {
69             printf("Tie!");
70             current_game.ties++;
71         }
72         else if (choice == ROCK && comp_choice == PAPER ||
73                  choice == PAPER && comp_choice == SCISSORS ||
74                  choice == SCISSORS && comp_choice == ROCK)
75         {
76             printf("Computer Wins!");
77             current_game.computer_wins++;
78         }
79         else if (comp_choice == ROCK && choice == PAPER ||
80                  comp_choice == PAPER && choice == SCISSORS ||
81                  comp_choice == SCISSORS && choice == ROCK)
82         {
83             printf("You Win!");
84             current_game.human_wins++;
85         }
86         current_game.rounds_played++;
87
88         printf("\\n\\n");
89
90         printf("Do you want to play again (y/n)? ");
91         scanf(" %c", &play_again);
92
93         printf("\\n");
94     } while (play_again != 'n');
95
96     print_game_details(current_game);
97
98
99     return 0;
100 }
```

**LAB 8: 8x10a – Rock-Paper-Scissors**

23/05/2022 – Monday, week 11

assign_user_choice & gen_comp_choice Source code lines 102-126	print_choice & print_game_details Source code lines 128-154
<pre> 102 enum Rock_paper_scissors assign_user_choice(char input) 103 { 104     int choice = NOTHING; 105 106     if (input == 'r'    input == 'R') 107     { 108         choice = ROCK; 109     } 110     else if (input == 'p'    input == 'P') 111     { 112         choice = PAPER; 113     } 114     else if (input == 's'    input == 'S') 115     { 116         choice = SCISSORS; 117     } 118 119     return choice; 120 } 121 122 enum Rock_paper_scissors gen_comp_choice(void) 123 { 124     // generate random number between 1-3 125     return rand() % 3 + 1; 126 }</pre>	<pre> 128 void print_choice(enum Rock_paper_scissors choice) 129 { 130     if (choice == ROCK) 131     { 132         printf("chose rock!"); 133     } 134     else if (choice == PAPER) 135     { 136         printf("chose paper!"); 137     } 138     else if (choice == SCISSORS) 139     { 140         printf("chose scissors!"); 141     } 142 } 143 144 void print_game_details(struct Game_Details game) 145 { 146     printf("=====\n"); 147     printf("   Game Details   \n"); 148     printf("=====\n\n"); 149 150     printf("Rounds played: %d\n", game.rounds_played); 151     printf("Human wins: %d\n", game.human_wins); 152     printf("Computer wins: %d\n", game.computer_wins); 153     printf("Ties: %d", game.ties); 154 }</pre>
<b>Output/s &amp; testing</b>	
<p>Normal game (valid inputs)</p> <pre> =====    ROCK PAPER SCISSORS    =====  Rock, paper, or scissors (r/p/s)? r Human player chose rock! Computer chose paper! Computer Wins!  Do you want to play again (y/n)? y Rock, paper, or scissors (r/p/s)? p Human player chose paper! Computer chose paper! Tie!  Do you want to play again (y/n)? y Rock, paper, or scissors (r/p/s)? s Human player chose scissors! Computer chose paper! You Win!  Do you want to play again (y/n)? n =====    Game Details    =====  Rounds played: 3 Human wins: 1 Computer wins: 1 Ties: 1</pre>	<p>Invalid input (not r/p/s and not y/n)</p> <pre> =====    ROCK PAPER SCISSORS    =====  Rock, paper, or scissors (r/p/s)? a Human player Computer chose paper!  Do you want to play again (y/n)? 2 Rock, paper, or scissors (r/p/s)? 52 Human player Computer chose rock!  Do you want to play again (y/n)? Rock, paper, or scissors (r/p/s)? n Human player Computer chose paper!  Do you want to play again (y/n)? n =====    Game Details    =====  Rounds played: 3 Human wins: 0 Computer wins: 0 Ties: 0</pre> <p style="text-align: right;">TODO: ask for input again if it is invalid</p>

**Lessons learned & debugging/errors**

- (I tried making the part: `//TODO: make into a function` into a function) Passing a struct as a parameter and updating a struct variable within the parameters will result in values that don't update (testing outputted all of the struct values as 0...).
- Returning a struct variable using a function results in values that don't update as well; it only updates for the current round and not all rounds combined!

Can I turn this TODO into a working function using my new knowledge of pointers?

## Week 11 lecture 032 Notes- Header Files, Static Libraries, P1Colour, Input Polling

25/05/2022- Wednesday, week 11

Notes Retrieved from Steffan Hooper (2022)

.c files generate a .obj file upon compilation, the linker links all .obj files into an executable.

Header files: .h header files contain the prototypes for each .c file. Ex. prototypes are created in a **myfunctions.h** file (header file), and "**myfunctions.h**" is **#included** in a .c file where it defines the function prototypes created in the header file. THEN the defined functions in the .c file is included in **source.c** (main file) to run the function.

To do a multi-file C project: use .cpp file template for C source code definitions, change .cpp file extension to .c.

Static libraries: .lib files allows sharing of functionality across programs- contains compiled function code. They do not contain a main function (no entry point).

Create .lib: right click project > properties > general > configuration type > static library

API (application programming interface): set of functions that allow the creation of a program with access to features or data of an operating system, application, or other service.

#INCLUDE DOES NOT WORK WITH LIB FILES

To use lib, go linker > input > additional dependencies > edit and type the name of the .lib there.

Non-standard console I/O (Windows only): **\_kbhit** and **\_getch** from **conio.h** allow input in real time. \_underscore means the functions are compiler specific. Not buffered like **scanf**, meaning input is passed directly into program.

**\_kbhit** detects whether key is pressed, ex: **while(!\_kbhit()) ...**

**\_getch** retrieves the input, ex: **input = \_getch(); ...**

To detect input of specific keys (like arrow keys), use an **enum** instead of magic numbers. **\_getch** must be called twice to detect what arrow key was pressed. (1. Check if it is an arrow key: 224, 2. check which arrow key: 72 is up arrow key).

Use **move\_cursor\_to(x, y);** (from provided API) to move the player as arrow keys are detected!

LAB 9: 9x07a – Count Categories

26/05/2022– Thursday, week 11

## Program aim

Complete the TODO in the given source code, and the instructions...

## Given Source Code

```

1 #define _CRT_SECURE_NO_WARNINGS
2 #include <stdio.h>
3
4 void count_categories(char* p_cstring, int* p_upper_count,
5   int* p_lower_count, int* p_digit_count);
6
7 Eint main(void)
8 {
9   char buffer1[] = "Hello Programming 1 Students";
10  char buffer2[] = "Learn to program using arrays and pointers!";
11
12  int upper_count = 0;
13  int lower_count = 0;
14  int digit_count = 0;
15
16  count_categories(buffer1, &upper_count, &lower_count, &digit_count);
17
18  printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer1,
19    upper_count, lower_count, digit_count);
20
21  count_categories(buffer2, &upper_count, &lower_count, &digit_count);
22
23  printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer2,
24    upper_count, lower_count, digit_count);
25
26  return 0;
27 }
```

```

28 void count_categories(char* p_cstring, int* p_upper_count,
29   int* p_lower_count, int* p_digit_count)
30 {
31   // TODO: Insert your code here...
32 }
```

## Implementation

## Instructions

Define the function **count\_categories** which returns three pieces of information, based upon the **p\_cstring** passed into the function. Count the number of uppercase, lowercase and digits in the **p\_cstring**, and return this information via the **p\_upper\_count**, **p\_lower\_count** and **p\_digit\_count** pointer parameters.

## Changed Source Code

```

25 Evoid count_categories(char* p_cstring, int* p_upper_count, int* p_lower_count, int* p_digit_count)
26 {
27   // TODO: Insert your code here...
28   int digit_count = 0;
29   int upper_count = 0;
30   int lower_count = 0;
31
32   for (int i = 0; p_cstring[i] != '\0'; i++)
33   {
34     if (p_cstring[i] >= 'A' && p_cstring[i] <= 'Z')
35     {
36       upper_count++;
37     }
38     else if (p_cstring[i] >= 'a' && p_cstring[i] <= 'z')
39     {
40       lower_count++;
41     }
42     digit_count++;
43   }
44
45   *p_upper_count = upper_count;
46   *p_lower_count = lower_count;
47   *p_digit_count = digit_count;
48 }
```

Add another two array declarations and initialisations locally in the main function. Call **count\_categories** with the newly added arrays and print the results.

## New arrays

```

10 char buffer3[] = "Hel0oOo0, my name is FrAaAaAaAn!";
11 char buffer4[] = "BOOTS and CATS and BOOTS and CATS";
```

## Call function &amp; print results

```

25 count_categories(buffer3, &upper_count, &lower_count, &digit_count);
26
27 printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer3, upper_count, lower_count, digit_count);
28
29 count_categories(buffer4, &upper_count, &lower_count, &digit_count);
30
31 printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer4, upper_count, lower_count, digit_count);
```

## Output/s

```

[Hello Programming 1 Students] upper: 3, lower: 21, digit: 28
[Learn to program using arrays and pointers!] upper: 1, lower: 35, digit: 43
[Hel0oOo0, my name is FrAaAaAaAn!] upper: 9, lower: 17, digit: 32
[BOOTS and CATS and BOOTS and CATS] upper: 18, lower: 9, digit: 33
```

## LAB 9: 9x07a – Count Categories (continued)

26/05/2022– Thursday, week 11

### Errors encountered

```
25 void count_categories(char* p_cstring, int* p_upper_count, int* p_lower_count, int* p_digit_count)
26 {
27     // TODO: Insert your code here...
28     for (int i = 0; p_cstring[i] != '\0'; i++)
29     {
30         if (p_cstring[i] >= 'A' && p_cstring[i] <= 'Z')
31         {
32             p_upper_count++;
33         }
34         else if (p_cstring[i] >= 'a' && p_cstring[i] <= 'z')
35         {
36             p_lower_count++;
37         }
38         p_digit_count = i;
39     }
40 }
```

the output (when `p_digit_count` is removed to prevent the error message), because the values are out of scope.

I tried assigning a values directly into a pointer variable, however this does not work as it returns the error: **warning C4047: '=': 'int' differs in levels of indirection from 'int \*'**. This is because I cannot assign to it within the loop as this is an out of scope error. This is also why the upper and lower count values are 0 in

### Lessons learned

- Remember to assign values to variables/pointers OUTSIDE loops and in the main line of a function to avoid going out of scope. Use (temporary) global variables to assign to local variables.

**LAB 11: 11x01a – Function Pointers**

26/05/2022 – Thursday (lab), week 11

**Program aim**

Using function pointers: complete the TODO in the given source code, and the instructions...

**Given Source Code**

```

1 #include <stdio.h>
2
3 // TODO: 1) Add the typedef here:
4
5 // Prototypes:
6 void repeat(int times);
7 void test(void);
8
9 // The main function definition:
10 int main(void)
11 {
12     // TODO: 4) Modify this call to pass in test:
13     repeat(10);
14
15     // TODO: 6) Add another call to your void function:
16
17     return 0;
18 }
```

```

20 // TODO: 2) Modify the parameter list of the repeat function to add
21 // in the function pointer parameter for the function to be
22 // called repeatedly. Also, do not forget to update the
23 // function declaration prototype on line 6 to match...
24
25 void repeat(int times)
26 {
27     for (int k = 0; k < times; ++k)
28     {
29         // TODO: 3) Add the call to the function pointer here:
30     }
31
32 void test(void)
33 {
34     printf("Test!\n");
35 }
36
37 // TODO: 5) Add your own void function, and prototype:
```

**Implementation****Instructions****Changed source code**

To complete the program, add a typedef for a function pointer where the function returns nothing, and takes in no parameters (//TODO: 1).

```

4 // TODO: 1) Add the typedef here:
5 typedef void(*type_void_fp)(void);
```

Modify the repeat function to add a second parameter of the function pointer typedef, which is the address of the function to be repeatedly called times number of times upon calling repeat (//TODO: 2 and //TODO: 3).

```

26 // TODO: 2) Modify the parameter list of the repeat function to add
27 // in the function pointer parameter for the function to be
28 // called repeatedly. Also, do not forget to update the
29 // function declaration prototype on line 6 to match...
30
31 void repeat(int times, type_void_fp test_fp)
32 {
33     for (int k = 0; k < times; ++k)
34     {
35         // TODO: 3) Add the call to the function pointer here:
36         test_fp();
37     }
38 }
```

Next, modify the call to repeat in the main function to pass in the address of the test function (//TODO: 4).

```

17 // TODO: 4) Modify this call to pass in test:
18 repeat(10, test);
```

add your own void function that can be used with the repeat function (//TODO: 5). In the main function, add a call to repeat using your new function as an argument (//TODO: 6).

```

44 // TODO: 5) Add your own void function, and prototype:
45
46 void test2(void)
47 {
48     printf("Horizontal ");
49 }
50
51 // TODO: 6) Add another call to your void function:
52 repeat(5, test2);
```

**Output/s**

```

Test!
Test!
Test!
Test!
Test!
Test!
Test!
Test!
Test!
horizontal horizontal horizontal horizontal horizontal
```

**Lessons learned**

- I learnt how to implement function pointers using **typedef**!

**LAB 11: 11x02a – QSort Integers**

26/05/2022 – Thursday (lab), week 11

**Program aim**

Ask the user how many integers they want to sort, then allow them to input the integer values. Use the heap to store the array of integers. Once all integers have been input by the user, sort the array using `qsort`.

**Planning**

I used TODO statements to guide the programming process, as shown in the source code.

**Implementation**

Source Code	Output/s / Testing
<pre> 3   #include &lt;stdlib.h&gt; 4 5   // sort function used with qsort 6   int sort_function(const void* a, const void* b) 7   { 8       return (*(int*)a - *(int*)b); 9   } 10 11  int main(void) 12  { 13      int length = 0; 14      int int_input = 0; 15      int* p_heap_sort_array = 0; 16 17      // TODO: ask user for length 18      printf("How many? "); 19      scanf("%d", &amp;length); 20 21      // TODO: malloc memory according to user inputted length 22      p_heap_sort_array = malloc(sizeof(int) * length); 23 24      // TODO: take user input and put into array 25      for (int i = 0; i &lt; length; i++) 26      { 27          printf("[%d]? ", i); 28          scanf("%d", &amp;int_input); 29 30          p_heap_sort_array[i] = int_input; 31      } 32 33      qsort(p_heap_sort_array, length, sizeof(int), sort_function); 34 35      // TODO: print sorted array 36      printf("\nSorted: \n\n"); 37      for (int i = 0; i &lt; length; i++) 38      { 39          printf("[%d] is %d \n", i, p_heap_sort_array[i]); 40      } 41 42      return 0; 43  }</pre>	<p>How many? 6</p> <p>[0]? 0  [1]? 8  [2]? 6  [3]? 4  [4]? 2  [5]? 4</p> <p>Sorted:</p> <p>[0] is 0  [1] is 2  [2] is 4  [3] is 4  [4] is 6  [5] is 8</p> <p>How many? 10</p> <p>[0]? -2  [1]? -5  [2]? -10  [3]? 4  [4]? 6  [5]? 2  [6]? 9  [7]? 10  [8]? 11  [9]? 5</p> <p>Sorted:</p> <p>[0] is -10  [1] is -5  [2] is -2  [3] is 2  [4] is 4  [5] is 5  [6] is 6  [7] is 9  [8] is 10  [9] is 11</p> <p>How many? 3</p> <p>[0]? 4  [1]? 4</p> <p>Sorted:</p> <p>[0] is 2  [1] is 4  [2] is 4</p>

**Error/debugging**

I forgot to free the `malloc`'d memory at the end of the program!

```

42          // FREE MEMORY
43          free(p_heap_sort_array);
44          p_heap_sort_array = 0;
45
46      }
47  }
```

**Lessons learned**

- I learnt how to implement `malloc()`, `qsort()`, and `free()` for the first time!
- Remember to free memory after using `malloc()` to prevent memory leaks! (and its good practice)
- I still need to understand how the code in `sort_function` works...

**LAB 11: 11x03a – Function Pointers**

26/05/2022 – Thursday (lab), week 11

**Program aim**

Generate an array of 25 integers randomly generated between 1-100 inclusive. Sort these using `qsort`, print the array elements by index, element address, and contents. Ask user for a search key and use `bsearch` to check and locate the search key in memory.

**Errors/debugging**

I get the following error when using random:

```
bsearch integers\source.c(14): warning C4242: 'function': conversion from 'time_t' to 'unsigned int', possible loss of data
```

To prevent this, I needed to include 4242 and 4244 in “disable specific warnings” to use random without a warning.

```
warning C4477: 'printf' : format string '%p' requires an argument of type 'void *', but variadic argument 2 has type 'int'
```

I got error C4477 because I forgot to use an `&` when printing an address...

**Implementation****Source Code**

```

1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  // function used with qsort & bsearch
7  int int_compare(const void* a, const void* b)
8  {
9      return (*(int*)a - *(int*)b);
10 }
11
12 int main(void)
13 {
14     srand(time(0)); // seed
15
16     int int_array[25];
17     int search_key = 0;
18     int* item = 0;
19
20     for (int i = 0; i < 25; i++)
21     {
22         int_array[i] = rand() % 100 + 1;
23     }
24
25     qsort(int_array, 25, sizeof(int), int_compare);
26
27     for (int i = 0; i < 25; i++)
28     {
29         printf("[%2d] at %p is %d\n", i, &int_array[i], int_array[i]);
30     }
31
32     printf("\nSearch key? ");
33     scanf("%d", &search_key);
34
35     item = bsearch(&search_key, int_array, 25, sizeof(int), int_compare);
36
37     if (item)
38     {
39         printf("%d found at %p\n", search_key, item);
40     }
41     else
42     {
43         printf("Not found!\n");
44     }
45
46     return 0;
47 }
```

**Output/s / Testing**

## Find existing key

```
[ 0] at 00EFFD4C is 1
[ 1] at 00EFFD50 is 7
[ 2] at 00EFFD54 is 12
[ 3] at 00EFFD58 is 28
[ 4] at 00EFFD5C is 28
[ 5] at 00EFFD60 is 30
[ 6] at 00EFFD64 is 33
[ 7] at 00EFFD68 is 34
[ 8] at 00EFFD6C is 39
[ 9] at 00EFFD70 is 52
[10] at 00EFFD74 is 55
[11] at 00EFFD78 is 56
[12] at 00EFFD7C is 57
[13] at 00EFFD80 is 62
[14] at 00EFFD84 is 64
[15] at 00EFFD88 is 68
[16] at 00EFFD8C is 69
[17] at 00EFFD90 is 74
[18] at 00EFFD94 is 78
[19] at 00EFFD98 is 84
[20] at 00EFFD9C is 86
[21] at 00EFFDA0 is 89
[22] at 00EFFDA4 is 91
[23] at 00EFFDA8 is 95
[24] at 00EFFDAC is 100
```

```
Search key? 62
62 found at 00EFFD80
```

## find non-existent key

```
[ 0] at 00EFF6BC is 3
[ 1] at 00EFF6C0 is 5
[ 2] at 00EFF6C4 is 8
[ 3] at 00EFF6C8 is 10
[ 4] at 00EFF6CC is 13
[ 5] at 00EFF6D0 is 21
[ 6] at 00EFF6D4 is 27
[ 7] at 00EFF6D8 is 30
[ 8] at 00EFF6DC is 34
[ 9] at 00EFF6E0 is 34
[10] at 00EFF6E4 is 35
[11] at 00EFF6E8 is 39
[12] at 00EFF6EC is 52
[13] at 00EFF6F0 is 55
[14] at 00EFF6F4 is 56
[15] at 00EFF6F8 is 60
[16] at 00EFF6FC is 75
[17] at 00EFF700 is 79
[18] at 00EFF704 is 82
[19] at 00EFF708 is 84
[20] at 00EFF70C is 88
[21] at 00EFF710 is 88
[22] at 00EFF714 is 92
[23] at 00EFF718 is 92
[24] at 00EFF71C is 94
```

```
Search key? 9000
Not found!
```

**Lessons learned**

- Include 4242 & 4244 in “disable specific warnings” to use the random number generator without warnings
- Print addresses using `&` before the name of whatever you’re printing.

## Week 11 lecture 033 Notes- Windows programming, MSDN, Win32, MsgBox

27/05/2022- Friday, week 11

Notes Retrieved from Steffan Hooper (2022)

### **Creating a windows application**

Windows handles: windows operating system uses “handles” (components of windows).

```
HANDLE console_window = GetStdHandle(STD_OUTPUT_HANDLE);  
A Windows.h type, stores handle of a window.  
Is a Windows.h function, returns handle (input, output, error)
```

Creating windows application: new project > windows desktop wizard > windows application (.exe) > empty project, no SDL check, no precompiled header...

### Creating a windows application:

1. start with `#include <Windows.h>` - Windows Platform SDK header file
2. create an entrypoint for windows application:

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)  
  
12. function that implements code required to run an EXE, is Windows entrypoint  
13. calling convention for system functions  
14. handle to an instance, base address of module in memory  
15. pointer to null-terminated string of 8-bit windows chars (ANSI).  
16. Command-line arguments  
17. Allows OS to specify how the window shows when launched
```

3. Messagebox example:

```
MessageBox(NULL, L"Hello COMP500/ENSE501", L"Hello!", MB_OK);  
  
18. Displays dialog/message box. Returns an integer to indicate which button user clicked  
19. Null pointer value (0)  
20. Wide-char string literal. REMEMBER L"..." AT THE START  
21. Message box style.
```

Microsoft development network (MSDN)- programmer documentation: highlight something you don't know and press f1 to open MSDN documentation on the web browser. Provides info and example code for a certain function etc.

Use `OutputDebugString()` to print to the output tab in VS, ex:

`OutputDebugString(L"Yes clicked!\n");` -> will output “Yes clicked!” to output tab.

`wchar_t & wsprintf`: wide char type. (REMEMBER WIDE CHAR IS USED IN MESSAGEBOX)

## LAB 9: 9x08a – Morse Code

27/05/2022 – Friday, week 11

## Program aim

Complete the given source code's TODO comments.

## Given Source Code

```
1 // Student ID:  
2  
3 #include <stdio.h>  
4 #include <string.h>  
5  
6 void analyse(char* code, int* p_dots, int* p_dashes)  
7 {  
8     // TODO: 1) Insert code here...  
9 }  
10
```

```
11 int main(void)
12 {
13     char* morse[] =
14     {
15         "... --- ...", // SOS
16         "- - - - . . . .", // MORSE
17         ". - - - - . . . .", // CODE
18         "- - - - - . . . . - - - - -", // COMP500
19         "- - - - - . . . . - - - - -", // ENSE501
20     };
21
22     // TODO: 2) Insert code here...
23
24     return 0;
25 }
```

## Implementation

## Instructions

At // TODO: 1) implement the analyse function.  
This function must take in a Morse code C-String as  
the first parameter, and return, via reference, the  
number of dashes and dots in the Morse code C-string.  
The analyse function must not have any printing or  
scanning side effects!

## Changed source code

```
1 // Student ID: 21145382
2
3 #include <stdio.h>
4 #include <string.h>
5
6 void analyse(char* code, int* p_dots, int* p_dashes)
7 {
8     // TODO: 1) Insert code here...
9
10    int dots = 0;
11    int dashes = 0;
12
13    for (int i = 0; code[i] != '\0'; i++)
14    {
15        if (code[i] == '.')
16        {
17            dots++;
18        }
19        else if (code[i] == '-')
20        {
21            dashes++;
22        }
23        // Return via reference
24        *p_dots = dots;
25        *p_dashes = dashes;
26    }
27}
```

Add code at `// TODO: 2)` to call and test your analyse implementation using the Morse code C-String array declared in the main function. Utilise console printing to display your test results.

(Instead of copying and pasting the print of each morse code individually, I used a for loop to call **analyse** and print each morse code c-string and their dot and dash count) →

## LAB 9: 9x08a – Morse Code (continued)

27/05/2022 – Friday, week 11

### Output/s

```
Morse code: ... --- ...
Dots: 6
Dashes: 3

Morse code: -- --- .-. .... .
Dots: 6
Dashes: 6

Morse code: -.-. --- -...
Dots: 5
Dashes: 6

Morse code: -.-. --- -- .--. .... -----
Dots: 9
Dashes: 19

Morse code: . -. .... . .... ----- ,-----
Dots: 12
Dashes: 10
```

### Lessons learned

- I learnt how to use a c-string array. Remember, a c-string array is initialised as a pointer (**char\***).
- C-string arrays are null-terminated, so I can use loop until '\0' is detected to loop through it fully without using a magic number

LAB 9: 9x09a – Triangle to File

27/05/2022 – Friday, week 11

## Program aim

Ask the user for the height of a triangle, and write out an ASCII art triangle to a file named `triangle.txt`

## Error/debugging

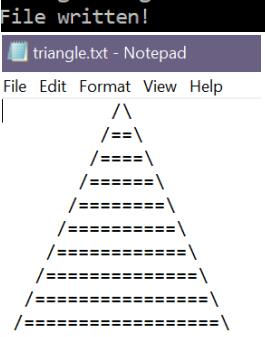
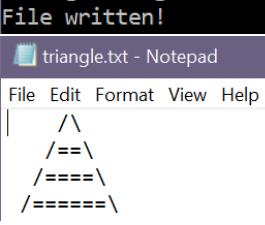
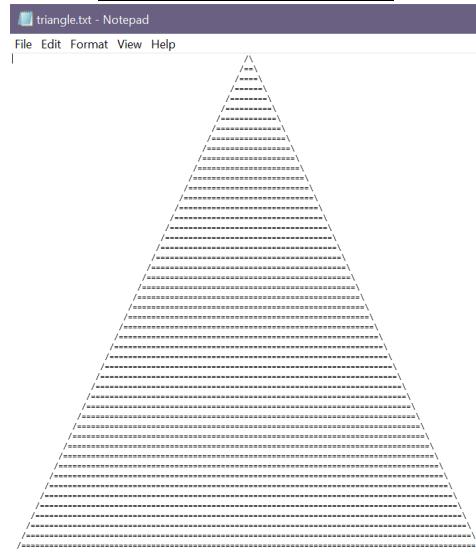
I got the **C4133** warning and **C2198** error because I used `fsprintf` to write each triangle character to the text file instead of `fputc`. `fsprintf` lines of text need to end with newline, but this would break the triangle shape.

```
Warning C4133: 'function': incompatible types - from 'char [2]' to 'FILE *const '
error C2198: 'fprintf': too few arguments for call
```

I also got warnings **C4047** & **C4024** because I used "" double quotes instead of '' single quotes, which is what is supposed to be used for a char.

```
warning C4047: 'function': 'int' differs in levels of indirection from 'char [2]'
warning C4024: 'fputc': different types for formal and actual parameter 1
```

## Implementation

Source Code	Output/s / Testing
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      int t_height = 0; 6 7      printf("Triangle height? "); 8      scanf("%d", &amp;t_height); 9 10     FILE* p_file = fopen("triangle.txt", "w"); 11 12     for (int i = 0; i &lt; t_height; i++) 13     { 14         // print spaces 15         for (int j = 0; j &lt; (t_height - i); j++) 16         { 17             // Write single char 18             fputc(' ', p_file); 19         } 20 21         fputc('/', p_file); 22 23         // print '=' 24         for (int k = 0; k &lt; (i * 2); k++) 25         { 26             fputc('=', p_file); 27         } 28 29         fputc('\\', p_file); 30         fputc('\n', p_file); 31     } 32 33     fclose(p_file); 34 35     printf("File written!"); 36 37     return 0; 38 }</pre>	<div style="display: flex; justify-content: space-around;"> <div style="text-align: center;"> <p>Triangle height? 10 File written!</p>  </div> <div style="text-align: center;"> <p>Triangle height? 4 File written!</p>  </div> </div> <div style="text-align: center;"> <p>Triangle height? 50 File written!</p>  </div>

## Lessons learned

- I learnt how to print to a text file for the first time!
- Use `sprintf` to print lines of text (double quotes), use `fputc` to print single characters (single quotes)

LAB 10: 9x10a – Text File Input

27/05/2022 – Friday, week 11

**Program aim**

Read in a text file one character at a time. At the start of the program, ask the user for the name of the file to read from. As the text file is read in, count the number of lines in the text file. Print out the contents of the text file, with each line numbered on the left-hand margin. Use the sample.txt file for an output.

**Error/debugging**

```

19 // check if end of file has been reached
20 while (feof(p_file) == 0)
21 {
22     int result = fgetc(p_file); // get single char
23
24     printf("%c", result); // print each char
25
26     if (result == '\n') // if char is a newline
27     {
28         printf("%d: ", line); // print line number
29     }
30     line++; // add to the line number
31 }
```

← when adding the line incrementation to the end of the while loop...

The line numbering seemed to largely multiply →

I found the reason was the lines were increasing for every single character instead of every newline, so I moved the line incrementation to **if (result == '\n')** (newline check).

```

Filename? sample.txt
1: #include <stdio.h>
19:
20: int main(void)
35: {
37:     int example[3][3];
60:
61:     // Fill the array with user input:
100:    for (int x = 0; x < 3; ++x)
132:    {
138:        for (int y = 0; y < 3; ++y)
174:        {
184:            printf("Set %d, %d to? ", x, y);
229:
230:            scanf("%d", &example[x][y]);
271:        }
281:    }
287:
288:    printf("\n");

```

**Implementation****Source Code****Output/s / Testing**

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     char filename[255];
6     int line = 1;
7
8     printf("Filename? ");
9     scanf("%s", &filename);
10    printf("\n");
11
12    FILE* p_file = 0;
13    p_file = fopen(filename, "r");
14
15    if (p_file)
16    {
17        printf("%d: ", line);
18
19        // check if end of file has been reached
20        while (feof(p_file) == 0)
21        {
22            int result = fgetc(p_file); // get single char
23
24            printf("%c", result); // print each char
25
26            if (result == '\n') // if char is a newline
27            {
28                line++; // add to the line number
29                printf("%d: ", line); // print line number
30            }
31        }
32
33        fclose(p_file);
34        p_file = 0;
35    }
36
37    return 0;
38 }
```

```

Filename? sample.txt
1: #include <stdio.h>
2:
3: int main(void)
4: {
5:     int example[3][3];
6:
7:     // Fill the array with user input:
8:     for (int x = 0; x < 3; ++x)
9:     {
10:         for (int y = 0; y < 3; ++y)
11:         {
12:             printf("Set %d, %d to? ", x, y);
13:
14:             scanf("%d", &example[x][y]);
15:         }
16:     }
17:
18:     printf("\n");
19:
20:     // Print out the array:
21:     for (int x = 0; x < 3; ++x)
22:     {
23:         for (int y = 0; y < 3; ++y)
24:         {
25:             printf("%d ", example[x][y]);
26:         }
27:
28:         printf("\n");
29:     }
30:
31:     printf("\n");
32:
33:     return 0;
34: }
```

**Lessons learned**

- For the first time, I learnt how to read from a text file and output to the contents to the console using the functions **feof()** (end of file check) and **fgetc()** (get character from file)
- Don't increment the line count every time a single char is printed, increment the line count within a line check like so: **if (character == '\n') ... line\_count++**

**LAB 10: 10x04a – Simple Casting**

28/05/2022 – Saturday, week 11

**Program aim**

Fix the given source code using type casting

**Implementation****Given Source Code**

```

1 #define _CRT_SECURE_NO_WARNINGS
2 #include <stdio.h>
3
4 int main(void)
5 {
6     int left;
7     int right;
8     float division;
9
10    printf("> ");
11    scanf("%d", &left);
12    printf("> ");
13    scanf("%d", &right);
14
15    // TODO: Fix the warning caused by the line below:
16    division = left / right;
17
18    printf("%d / %d is %f\n", left, right, division);
19
20    return 0;
21 }
```

**Changed source code**

```

1 #define _CRT_SECURE_NO_WARNINGS
2 #include <stdio.h>
3
4 int main(void)
5 {
6     int left;
7     int right;
8     float division;
9
10    printf("> ");
11    scanf("%d", &left);
12    printf("> ");
13    scanf("%d", &right);
14
15    // TODO: Fix the warning caused by the line below:
16    division = (float)left / right;
17
18    printf("%d / %d is %f\n", left, right, division);
19
20    return 0;
21 }
```

**Output/s**

Warning C4244: '=': conversion from 'int' to 'float', possible loss of data

> 5	> 10
> 2	> 2
5 / 2 is 2.500000	10 / 2 is 5.000000
> 32	> 7
> 3	> 3
32 / 3 is 10.666667	7 / 3 is 2.333333

**Lessons learned**

- I have already used type casting in some past exercises to work around dividing integer variables

LAB 10: 10x05a - Small malloc

28/05/2022 – Saturday, week 11

## Program aim

Allocate space to an `int` array using `malloc`.

## plan

1. Use a single `malloc` call to allocate space for an array of three `int` values
2. Assign the values: **150, 275, 400** to the positions `[0], [1], [2]` in the array
3. display the contents of the array
4. deallocate the heap allocation using `free`

## Error/debugging

```
warning C4013: 'malloc' undefined; assuming extern returning int
warning C4013: 'free' undefined; assuming extern returning int
warning C4047: 'initializing': 'int *' differs in levels of indirection from 'int'
```

Microsoft Visual C++ Runtime Library (Not Responding)

use the `malloc` and `free` functions.

Debug Error!

Program: ...IO B\Exercises\Lab 10 VSS\x64\Debug\10x05a - Small malloc.exe

HEAP CORRUPTION DETECTED: after Normal block (#74) at 0x0000026FC0C3ED20.  
 CRT detected that the application wrote to memory after end of heap buffer.

(Press Retry to debug the application)

← the debug error occurred because I accidentally allocated an integer out of scope of the allocated memory (index of 3 instead of 2)

```
// Assign values to positions in array
malloc_array[0] = 150;
malloc_array[1] = 275;
malloc_array[3] = 400;
```

## Implementation

## Source Code

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int main(void)
6  {
7      // allocate space to array of 3 in values on the heap
8      int* malloc_array = malloc(sizeof(int) * 3);
9
10     // Assign values to positions in array
11     malloc_array[0] = 150;
12     malloc_array[1] = 275;
13     malloc_array[2] = 400;
14
15     // display contents of array
16     printf("First element: %d\n", malloc_array[0]);
17     printf("Second element: %d\n", malloc_array[1]);
18     printf("Third element: %d\n", malloc_array[2]);
19
20     // Free memory
21     free(malloc_array);
22     malloc_array = 0;
23
24     return 0;
25 }
```

## Output/s / Testing

```
First element: 150
Second element: 275
Third element: 400
```

## Lessons learned

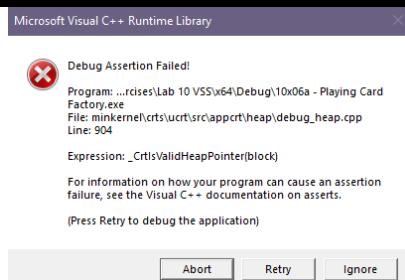
- include `<stdlib.h>` to use `malloc()` and `free()`
- Avoid assigning values out of scope of the `malloc`'d memory

**LAB 10: 10x06a - Playing Card Factory**

28/05/2022 – Saturday, week 11

**Program aim**

Create a random card generator from a deck of cards. Randomly generate 5 cards (suit and deck) using structs and pointers, memory allocation and freeing, functions, random number generator, and use assert to check if members are within specified ranges.

**Error/debugging**

← I received the following error because I attempted to free the allocated memory in `card_factory()` like this:

```
free(cards);
```

But this would just free the array itself, which isn't `malloc`'d...

It's the elements of cards that are `malloc`'d, so I freed the memory for each struct array element of cards using a `for` loop (which worked).

I also made the mistake of seeding the rng (random number generator) in the function `card_factory()`, which resulted in all 5 structs in the cards array to have the exact same values (not updating). To fix this, I moved the rng seed to `main`.

**Implementation**

Source Code	Output/s / Testing																				
<pre> 1 #define _CRT_SECURE_NO_WARNINGS 2 #include &lt;stdio.h&gt; 3 #include &lt;stdlib.h&gt; 4 #include &lt;time.h&gt; 5 #include &lt;assert.h&gt; 6 7 struct Playing_card 8 { 9     int rank; 10    int suit; 11 }; 12 13 struct Playing_card* card_factory(void); 14 void print_playing_card(struct Playing_card* cards); 15 16 int main(void) 17 { 18     srand(time(0)); // seed rng 19 20     struct Playing_card* cards[5]; 21 22     for (int i = 0; i &lt; 5; i++) 23     { 24         cards[i] = card_factory(); 25         print_playing_card(cards[i]); 26         printf("\n"); 27     } 28 29     for (int i = 0; i &lt; 5; i++) 30     { 31         free(cards[i]); 32     } 33 34     return 0; 35 }  36 struct Playing_card* card_factory(void) 37 { 38     // allocate new Playing_Card structure on the heap 39     struct Playing_card* p_new_card = malloc(sizeof(struct Playing_card)); 40     assert(p_new_card); 41 42     p_new_card-&gt;rank = rand() % 13 + 1; 43     // use assert to ensure rank and suit members are set within the specified ranges 44     assert(p_new_card-&gt;rank &gt;= 1 &amp;&amp; p_new_card-&gt;rank &lt;= 13); 45 46     p_new_card-&gt;suit = rand() % 3; 47     assert(p_new_card-&gt;suit &gt;= 0 &amp;&amp; p_new_card-&gt;suit &lt;= 3); 48 49     return p_new_card; 50 } 51 52 53 void print_playing_card(struct Playing_card* card) 54 { 55     char* card_suits[] = { "Diamonds", "Hearts", "Clubs", "Spades" }; 56     char* card_ranks[] = { " ", "Ace", "two", "three", "four", "five", "six", "seven", 57                           "eight", "nine", "ten", "Jack", "Queen", "King" }; 58 59     printf("Printing playing card:\n"); 60     printf("Rank: %s\n", card_ranks[card-&gt;rank]); 61     printf("Suit: %s\n", card_suits[card-&gt;suit]); 62 }</pre>	<p>Playing cards are different each program run because of rng</p> <table border="1"> <tbody> <tr> <td>Printing playing card: Rank: three Suit: Clubs</td><td>Printing playing card: Rank: ten Suit: Diamonds</td></tr> <tr> <td>Printing playing card: Rank: eight Suit: Diamonds</td><td>Printing playing card: Rank: three Suit: Clubs</td></tr> <tr> <td>Printing playing card: Rank: Queen Suit: Clubs</td><td>Printing playing card: Rank: eight Suit: Hearts</td></tr> <tr> <td>Printing playing card: Rank: seven Suit: Hearts</td><td>Printing playing card: Rank: ten Suit: Diamonds</td></tr> <tr> <td>Printing playing card: Rank: five Suit: Hearts</td><td>Printing playing card: Rank: King Suit: Diamonds</td></tr> <tr> <td>Printing playing card: Rank: eight Suit: Hearts</td><td>Printing playing card: Rank: Jack Suit: Diamonds</td></tr> <tr> <td>Printing playing card: Rank: eight Suit: Hearts</td><td>Printing playing card: Rank: Jack Suit: Diamonds</td></tr> <tr> <td>Printing playing card: Rank: six Suit: Diamonds</td><td>Printing playing card: Rank: three Suit: Clubs</td></tr> <tr> <td>Printing playing card: Rank: three Suit: Hearts</td><td>Printing playing card: Rank: Jack Suit: Clubs</td></tr> <tr> <td>Printing playing card: Rank: three Suit: Diamonds</td><td>Printing playing card: Rank: Queen Suit: Hearts</td></tr> </tbody> </table>	Printing playing card: Rank: three Suit: Clubs	Printing playing card: Rank: ten Suit: Diamonds	Printing playing card: Rank: eight Suit: Diamonds	Printing playing card: Rank: three Suit: Clubs	Printing playing card: Rank: Queen Suit: Clubs	Printing playing card: Rank: eight Suit: Hearts	Printing playing card: Rank: seven Suit: Hearts	Printing playing card: Rank: ten Suit: Diamonds	Printing playing card: Rank: five Suit: Hearts	Printing playing card: Rank: King Suit: Diamonds	Printing playing card: Rank: eight Suit: Hearts	Printing playing card: Rank: Jack Suit: Diamonds	Printing playing card: Rank: eight Suit: Hearts	Printing playing card: Rank: Jack Suit: Diamonds	Printing playing card: Rank: six Suit: Diamonds	Printing playing card: Rank: three Suit: Clubs	Printing playing card: Rank: three Suit: Hearts	Printing playing card: Rank: Jack Suit: Clubs	Printing playing card: Rank: three Suit: Diamonds	Printing playing card: Rank: Queen Suit: Hearts
Printing playing card: Rank: three Suit: Clubs	Printing playing card: Rank: ten Suit: Diamonds																				
Printing playing card: Rank: eight Suit: Diamonds	Printing playing card: Rank: three Suit: Clubs																				
Printing playing card: Rank: Queen Suit: Clubs	Printing playing card: Rank: eight Suit: Hearts																				
Printing playing card: Rank: seven Suit: Hearts	Printing playing card: Rank: ten Suit: Diamonds																				
Printing playing card: Rank: five Suit: Hearts	Printing playing card: Rank: King Suit: Diamonds																				
Printing playing card: Rank: eight Suit: Hearts	Printing playing card: Rank: Jack Suit: Diamonds																				
Printing playing card: Rank: eight Suit: Hearts	Printing playing card: Rank: Jack Suit: Diamonds																				
Printing playing card: Rank: six Suit: Diamonds	Printing playing card: Rank: three Suit: Clubs																				
Printing playing card: Rank: three Suit: Hearts	Printing playing card: Rank: Jack Suit: Clubs																				
Printing playing card: Rank: three Suit: Diamonds	Printing playing card: Rank: Queen Suit: Hearts																				

**Lessons learned**

- I learnt how to use `assert()` for the first time! (used to ensure values are valid). Use `assert.h` to access it!
- Seed the random number generator in main so that rng values can update
- I should ensure I am freeing something that was actually `malloc`'d

**LAB 10: 10x07a - Detect and Fix Leaks**

28/05/2022 – Saturday, week 11

**Program aim**

Fix the given source code by detecting the memory leaks.

**Given Source Code**

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int* create_array(int size);
5  void fill_array(int* p_array, int size);
6
7  int main(void)
8  {
9      int* first_array = create_array(5);
10     fill_array(first_array, 5);
11
12     int* second_array = create_array(3);
13     fill_array(second_array, 3);
14
15     int* third_array = create_array(3);
16     fill_array(third_array, 3);
17
18     return 0;
19 }
20
21 int* create_array(int size)
22 {
23     int* p_data = malloc(size * sizeof(int));
24
25     return p_data;
26 }
27
28 void fill_array(int* p_array, int size)
29 {
30     for (int k = 0; k < size; ++k)
31     {
32         printf("Enter a value: ");
33         scanf("%d", &(p_array[k]));
34     }
35 }
```

**Implementation****"Instructions" (and own comments)**

"Modify the program, using `crtdbg.h`, such that you can detect the memory leak in the Output window of Visual Studio."

(Added `#include <crtdbg.h>` and called `_CrtSetDbgFlag`, with bit flag parameters) →  
Also added a print for the addresses of all 3 `malloc`'d data in the program to check if they match up to the memory leak detection

(The addresses printed out in the console match the detected memory leaks in the debug output) →

"Once detected, fix the memory leaks. Ensure the program no longer leaks!"

(freed the memory for each array) →

(program exits without memory leaks detected) →

**Changed source code**

```

4  #include <crtdbg.h>
5
6  int* create_array(int size);
7  void fill_array(int* p_array, int size);
8
9  int main(void)
10 {
11     _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
12
13     int* first_array = create_array(5);
14     fill_array(first_array, 5);
15
16     int* second_array = create_array(3);
17     fill_array(second_array, 3);
18
19     int* third_array = create_array(3);
20     fill_array(third_array, 3);
21
22     printf("Malloc gave me: %p, %p, %p", first_array, second_array, third_array);
23 }
```

**Console output (added print to show addresses)**

```
Malloc gave me: 0000025403B8F9C0, 0000025403B8F6F0, 0000025403B8F8D0
```

**Debug output:**

```

Detected memory leaks!
Dumping objects ->
{81} normal block at 0x0000025403B8F8D0, 12 bytes long.
Data: <           > 09 00 00 00 0A 00 00 00 0B 00 00 00
{80} normal block at 0x0000025403B8F6F0, 12 bytes long.
Data: <           > 06 00 00 00 07 00 00 00 08 00 00 00
{77} normal block at 0x0000025403B8F9C0, 20 bytes long.
Data: <           > 01 00 00 00 02 00 00 00 03 00 00 00 04 00 00 00
Object dump complete.]
```

```

24     free(first_array);
25     first_array = 0;
26
27     free(second_array);
28     second_array = 0;
29
30     free(third_array);
31     third_array = 0;
```

**Debug output:**

```
The thread 0x1b84 has exited with code 0 (0x0).
The thread 0x1828 has exited with code 0 (0x0).
The program '[9028] 10x07a - Detect and Fix Leaks.exe' has exited with code 0 (0x0).
```

**Lessons learned**

- I learnt how to use `crtdbg.h` to detect, identify, and fix memory leaks using the debugger.
- Use this line at the beginning of `main` to detect memory leaks:  
`CrtSetDbgFlag( _CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF );`

**LAB 10: 10x08a - Recursive Movie**

28/05/2022 – Saturday, week 11

**Program aim**

Movie's Name:	Year of Release:	Rotten Tomatoes:
Jurassic Park	1993	93%
The Lost World: Jurassic Park	1997	51%
Jurassic Park III	2001	50%
Jurassic World	2015	72%

Store the following data as **struct** members, create an additional member representing whether the movie has a sequel or not (should be a pointer to a **struct Movie**), and print them using a function.

**Error/debugging**

```
for (int i = 0; i < total_movies; ++i)
{
    sprintf(jp_movie[i]->name, "%s", m_names[i]);
    jp_movie[i]->release_year = y_releases[i];
    jp_movie[i]->rotten_tomatoes = rt_scores[i];
```

← this syntax doesn't seem to work. [I realised this is due to the '->' member access operator not using the index \[\] operator.](#)

I fixed this by changing the syntax to, for example:

```
(jp_movie+i)->release_year = y_releases[i];
```

**Implementation****Source Code**

```
5  struct Movie
6  {
7      char name[30];
8      int release_year;
9      int rotten_tomatoes;
10     struct Movie* is_sequel;
11 };
12
13 void print_movie(struct Movie* jp_movie);
14
15 int main(void)
16 {
17     struct Movie* jp_movie = 0;
18     int total_movies = 4;
19
20     jp_movie = malloc(sizeof(struct Movie) * total_movies);
21
22     char* m_names[] = { "Jurassic park", "The Lost World: Jurassic Park", "Jurassic Park III", "Jurassic World" };
23     int y_releases[] = { 1993, 1997, 2001, 2015 };
24     int rt_scores[] = { 93, 51, 50, 72 };
25
26     for (int i = 0; i < total_movies; ++i)
27     {
28         sprintf((jp_movie+i)->name, "%s", m_names[i]);
29         (jp_movie+i)->release_year = y_releases[i];
30         (jp_movie+i)->rotten_tomatoes = rt_scores[i];
31
32         if (i < (total_movies-1))
33         {
34             (jp_movie + i)->is_sequel = &jp_movie[i + 1];
35         }
36         else
37         {
38             (jp_movie + i)->is_sequel = 0; // null
39         }
40     }
41
42     for (int i = 0; i < total_movies; i++)
43     {
44         print_movie(&jp_movie[i]);
45         printf("\n\n");
46     }
47
48     free(jp_movie);
49
50     return 0;
51 }
52
53 void print_movie(struct Movie* jp_movie)
54 {
55     printf("Name: %s\n", jp_movie->name);
56     printf("Year of Release: %d\n", jp_movie->release_year);
57     printf("Rotten Tomatoes: %d%\n", jp_movie->rotten_tomatoes);
58
59     if (jp_movie->is_sequel != 0)
60     {
61         printf("%s's sequel was...", jp_movie->name);
62     }
63     else
64     {
65         printf("%s has no sequel, yet! ", jp_movie->name);
66     }
67 }
```

**Output/s / Testing**

```
Name: Jurassic park
Year of Release: 1993
Rotten Tomatoes: 93%
Jurassic park's sequel was...

Name: The Lost World: Jurassic Park
Year of Release: 1997
Rotten Tomatoes: 51%
The Lost World: Jurassic Park's sequel was...

Name: Jurassic Park III
Year of Release: 2001
Rotten Tomatoes: 50%
Jurassic Park III's sequel was...

Name: Jurassic World
Year of Release: 2015
Rotten Tomatoes: 72%
Jurassic World has no sequel, yet!
```

**Lessons learned**

- The syntax of index use for member access operators (->) doesn't use [] syntax, ex:  
`(struct_array_name+i)->member_name = something;`

**LAB 10: 10x09a - `typedef enum`**

28/05/2022 – Saturday, week 11

**Program aim**Given the following source code, edit it so it uses a `typedef` with the `enum` declaration.**Implementation**

Given Source Code	Changed source code
<pre> 4  enum Suit 5  { 6      CLUB, 7      SPADE, 8      DIAMOND, 9      HEART 10 } 11 12 void print_card_type(enum Suit card_type); 13 14 int main(void) 15 { 16     enum Suit example_card = SPADE; 17     printf("example card is a... "); 18     print_card_type(example_card); 19 20     enum Suit another_card = HEART; 21     printf("another card is a... "); 22     print_card_type(another_card); 23 24     return 0; 25 } 26 27 void print_card_type(enum Suit card_type) 28 { 29     if (card_type == CLUB) 30     { 31         printf("Club\n"); 32     } 33     else if (card_type == SPADE) 34     { 35         printf("Spade\n"); 36     } 37     else if (card_type == DIAMOND) 38     { 39         printf("Diamond\n"); 40     } 41     else if (card_type == HEART) 42     { 43         printf("Heart\n"); 44     } 45 }</pre>	<pre> 4  typedef enum tag_Suit 5  { 6      CLUB, 7      SPADE, 8      DIAMOND, 9      HEART 10 } Suit; 11 12 void print_card_type(enum Suit card_type); 13 14 int main(void) 15 { 16     Suit example_card = SPADE; 17     printf("example card is a... "); 18     print_card_type(example_card); 19 20     Suit another_card = HEART; 21     printf("another card is a... "); 22     print_card_type(another_card); 23 24     return 0; 25 } 26 27 void print_card_type(enum Suit card_type) 28 { 29     if (card_type == CLUB) 30     { 31         printf("Club\n"); 32     } 33     else if (card_type == SPADE) 34     { 35         printf("Spade\n"); 36     } 37     else if (card_type == DIAMOND) 38     { 39         printf("Diamond\n"); 40     } 41     else if (card_type == HEART) 42     { 43         printf("Heart\n"); 44     } 45 }</pre>

**Output/s**

Both source code has the same output.

```

example card is a... Spade
another card is a... Heart

```

**Lessons learned**

- I learnt how to use `typedef` with an `enum` so that I wouldn't have to write '`enum`' when calling the `Suit` enumeration- I only have to write '`Suit` (enum variable name) = (enum member)'

## LAB 10: 10x10a - Coloured Log Cabin

**28/05/2022 – Saturday, week 11**

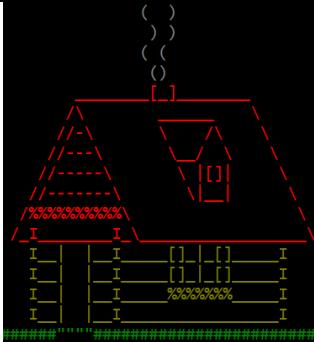
## Program aim

Given the following source code, modularise the printing of the ASCII art log cabin and use `p1colour.h` and `p1colour.lib` functionality to colour its parts.

## Implementation

Given Source Code	Changed source code
<pre> 1  #include &lt;stdio.h&gt; 2 3  #include "picolour.h" 4 5  void print_smoke(void); 6  void print_roof(void); 7  void print_walls(void); 8  void print_ground(void); 9 10 11 int main(void) 12 { 13     printf("      (\n"); 14     printf("      )\n"); 15     printf("      (\n"); 16     printf("      ()\n"); 17     printf("      [ ]\n"); 18     printf("      /\\ \\\n"); 19     printf("      //\\ \\ \\\n"); 20     printf("      //---\\ \\ \\\n"); 21     printf("      //----\\ \\ \\\n"); 22     printf("      //-----\\ \\ \\\n"); 23     printf("      /%%%%%%%%%%%%%\n"); 24     printf("      / _ _ _ _ _ \\ \\\n"); 25     printf("      I _     I [ ] [ ] I\n"); 26     printf("      I _     I [ ] [ ] I\n"); 27     printf("      I _     I %%%%%%%%%% I\n"); 28     printf("      I _     I I\n"); 29     printf("#####\\ \"\"\" #####\n"); 30 31     return 0; 32 }</pre>	<pre> 3  #include "picolour.h" 4 5  void print_smoke(void); 6  void print_roof(void); 7  void print_walls(void); 8  void print_ground(void); 9 10 11 int main(void) 12 { 13     print_smoke(); 14     print_roof(); 15     print_walls(); 16     print_ground(); 17 18     return 0; 19 } 20 21 void print_smoke(void) 22 { 23     set_text_colour(DARKGRAY, BLACK); 24 25     printf("      (\n"); 26     printf("      )\n"); 27     printf("      (\n"); 28     printf("      ()\n"); 29 } 30 31 void print_roof(void) 32 { 33     set_text_colour(LIGHTRED, BLACK); 34 35     printf("      [ ]\n"); 36     printf("      /\\ \\\n"); 37     printf("      //\\ \\ \\\n"); 38     printf("      //---\\ \\ \\\n"); 39     printf("      //----\\ \\ \\\n"); 40     printf("      //-----\\ \\ \\\n"); 41     printf("      /%%%%%%%%%%%%%\n"); 42     printf("      / _ _ _ _ _ \\ \\\n"); 43 } 44 45 void print_walls(void) 46 { 47     set_text_colour(BROWN, BLACK); 48 49     printf("      I _     I [ ] [ ] I\n"); 50     printf("      I _     I [ ] [ ] I\n"); 51     printf("      I _     I %%%%%%%%%% I\n"); 52     printf("      I _     I I\n"); 53 } 54 55 void print_ground(void) 56 { 57     set_text_colour(GREEN, BLACK); 58 59     printf("#####\\ \"\"\" #####\n"); 60 }</pre>

## Output/s



### **Lessons learned**

- I learnt how to use the given header file: `plcolour.h` to create coloured console text!

# **WEEK 12**

## Week 12 lecture 034 Notes- Windows API

30/05/2022- Monday, week 12

Notes Retrieved from Steffan Hooper (2022)

Windows software development kit (SDK) – documentation and tools for developing windows software

`wchar_t` stores wide characters. Use: `wchar_t wstr[]`; and `wchar_t* p_wstr;`  
`sizeof(wchar_t)` is 2 bytes.

To create string wide c-string: `L"Hello"` ...

Disable warnings `4668, 4255, C4255...`

Warning `C4100`: unreferenced formal parameter.

With `Windows.h`, use `UNREFERENCED_PARAMETER` to prevent warning `C4100`. For example:  
`UNREFERENCED_PARAMETER(hInstance);`

`GetSystemMetrics()` – gets screen size, returns an int.

`SM_CXSCREEN` is the width, `SM_CYSCREEN` is the height. (Pass these as arguments)

`wsprintf()` is the wide character version of `sprintf()`.

`BOOL LockWorkStation(void)`; - locks workstation display, protecting it from unauthorised use. Returns `BOOL`, which is `TRUE` (`int` value of 1) or `FALSE` (`int` value of 0).

`DWORD GetLastError(void)`; - returns double-byte/unsigned int (`DWORD`). Retrieves the caller's *last-error* code value (which is what is set after a windows system function fails).

`BOOL GetComputerName(LPTSTR lpBuffer, LPDWORD lpnSize)`; - gets computer name.  
`BOOL GetUserName(LPTSTR lpBuffer, LPDWORD lpnSize)`; - get username.

22. Pointer to buffer receiving computer name/cluster virtual server name/user logon name.

23. Input: specifies size of buffer in TCHARs. Output: number of TCHARs copied to destination buffer.

24. Function succeeds = nonzero value. Fail = 0.

`UNLEN` accessed using `lcons.h`. `[UNLEN + 1]` holds max length of username.

**More functions:**

`DWORD GetCurrentDirectory(DWORD nBufferLength, LPTSTR lpBuffer)`; - get current directory

`BOOL SetCurrentDirectory(LPCTSTR lpPathName)`; - set current directory

`void ZeroMemory(PVOID Destination, SIZE_T Length)`; - fills block of memory with zeros.

`SYSTEMTIME` struct and `GetLocalTime()`; function: get local time...

Important functions for creating a windows application.

Entry point for GUI: `int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR params, int nCmdShow);`

Receives messages for window: `LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);`

## Week 12 lecture 034 Notes- Windows API (Continued)

30/05/2022- Monday, week 12

Notes Retrieved from Steffan Hooper (2022)

Register own type of window (first things done in **WinMain** function):

1. Populate struct: **WNDCLASSEX**
2. Call **RegisterClassEx()** function

Then

3. call **CreateWindowEx()** to create window...

After successful window creation,

4. **BOOL ShowWindow(HWND hWnd, int nCmdShow)** – show window
5. **BOOL UpdateWindow(HWND hWnd)** – update window

Then

6. **MSG** – struct containing message info
7. **GetMessage()** – retrieves message

**PERSONAL PROJECT #1 – Random ASCII emoticon generator**  
**(To be continued)**  
**30/05/2022- Monday, week 12**

**Program aim**

Create a program that randomly generates ASCII emoticons. The user can keep generating random emoticons until they want it to stop.

**Plan**

- Create 2 char arrays, one for different eye char symbols, and one for different mouth char symbols.
- Randomly generate a number within the range of each array size (0 to array size) to generate a random ASCII character selection each.
- Loop while the user hasn't entered 0 (to quit)
- Print the randomly generated emoticon in the format: bracket-eye-mouth-eye-bracket. Ex: (-\_-)
- Ask user if they want to generate another emoticon (0 = no, 1 = yes)
- Quit program if user enters 0.

**Implementation**

**Source Code**

```

1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3  #include <time.h>
4  #include <stdlib.h>
5
6  int main(void)
7  {
8      srand(time(0));
9
10     int generate_emoji = 1;
11     char* ascii_eyes[] = {"-", "O", "~", "U", "X", "*", "^", "@", "+"};
12     char* ascii_mouths[] = {"_", ".", "w", "o"};
13
14     while (generate_emoji == 1)
15     {
16         int rand_eye = rand() % 8;
17         int rand_mouth = rand() % 3;
18
19         printf("(%%c%c)", *ascii_eyes[rand_eye], *ascii_mouths[rand_mouth], *ascii_eyes[rand_eye]);
20         printf("\n");
21
22         printf("Generate another emoticon? ");
23         scanf("%d", &generate_emoji);
24     }
25
26     printf("\nGoodbye!");
27
28     return 0;
29 }
30
31 // TODO: modularise program
32 // TODO: no magic number array size, make variable for array size
33 // TODO: use enums
34 // TODO: add input validation

```

**Output/s / Testing**

```

(*_*)
Generate another emoticon? 1
(^_^)
Generate another emoticon? 1
(*w*)
Generate another emoticon? 1
(x_x)
Generate another emoticon? 1
(-w-)
Generate another emoticon? 1
(O.O)
Generate another emoticon? 1
(u.u)
Generate another emoticon? 1
(~w~)
Generate another emoticon? 1
(^w^)
Generate another emoticon? 1
(~_~)
Generate another emoticon? 1
(xwx)
Generate another emoticon? 1
(~_~)
Generate another emoticon? 0
Goodbye!

```

**Errors/debugging**

**Complete the TODOs in the source code to make the code more modular and robust.**

Invalid input will infinitely generate a random emoticon, along with the prompt (I need to clear the buffer):

```

(^w^)
Generate another emoticon? no
(@@)
Generate another emoticon? (-w-)
Generate another emoticon? (*w*)
Generate another emoticon? (u.u)
Generate another emoticon? (@@)
Generate another emoticon? (xwx)
Generate another emoticon? (OoO)
Generate another emoticon? (~_~)

```

Int input that isn't 1 or 0 quits the program, which is fine

```

(*_*)
Generate another emoticon? 2
Goodbye!

```

**LAB 11: 11x04a – Debug vs Release**

30/05/2022- Monday, week 12

**Program aim**

With the given source code, build it using the debug build target and the release debug target. Note the differences.

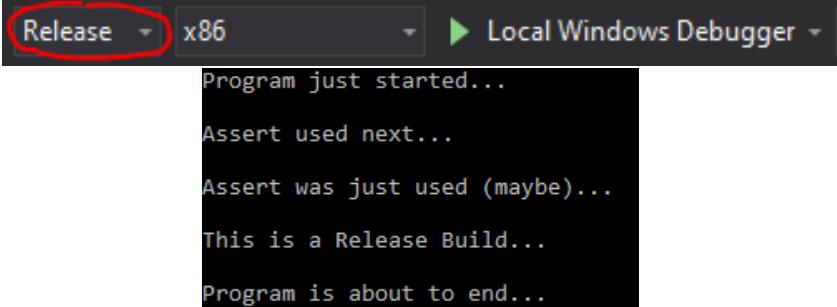
**Given Source Code**

```

1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3  #include <assert.h>
4
5  int side_effect_example(void);
6
7  int main(void)
8  {
9      printf("Program just started...\n\n");
10     printf("Assert used next...\n\n");
11     assert(side_effect_example());
12     printf("Assert was just used (maybe)...\n\n");
13
14     #ifdef _DEBUG
15         printf("This is a Debug Build...\n\n");
16     #else
17         printf("This is a Release Build...\n\n");
18     #endif // _DEBUG
19
20     printf("Program is about to end...\n\n");
21
22     return 0;
23 }
24
25 int side_effect_example(void)
26 {
27     printf("This function will not be ");
28     printf("called when the program is ");
29     printf("compiled in Release mode.\n\n");
30
31     return 1;
32 }
33

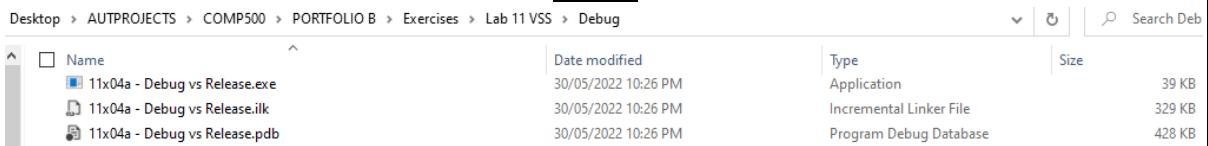
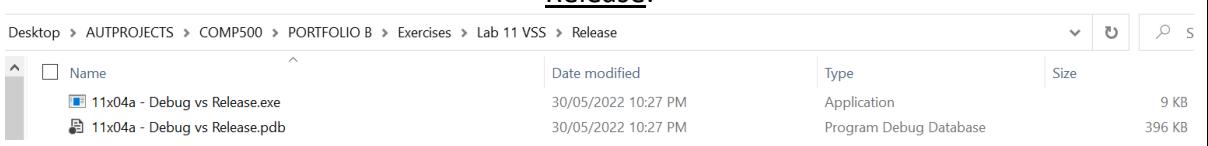
```

**Implementation**

Instructions	Changed source code / reflection
“Build the source code using the Debug build target. Run the resulting program and note the output.”	 <pre> Program just started... Assert used next... This function will not be called when the program is compiled in Release mode. Assert was just used (maybe)... This is a Debug Build... Program is about to end... </pre>
“Build the source code using the Release build target. Run the resulting program and note the output.”	 <pre> Program just started... Assert used next... Assert was just used (maybe)... This is a Release Build... Program is about to end... </pre>
“What changes occurred between the two build targets?”	<ul style="list-style-type: none"> <li>- The <code>side_effect_example</code> function’s print statements in the debug build target is not included in the release build target due to <code>assert()</code> preventing it from doing so.</li> <li>- The debug build prints “This is a debug build” and the release build prints “this is a release build” instead. This happens because the <code>#ifdef</code> and <code>#else</code> preprocessor directives allows parts of the program to selectively run depending on the build type.</li> </ul>

**LAB 11: 11x04a – Debug vs Release (Continued)**

30/05/2022- Monday, week 12

Implementation	
Instructions	Changed source code / reflection
“Find the resulting executable files from each build target on disk.”	<p style="text-align: center;"><u>Debug:</u></p>  <p style="text-align: center;"><u>Release:</u></p> 
“How were the resulting executables different?”	<ul style="list-style-type: none"> <li>- The debug executable size is 39 KB, and the release executable size is 9 KB.</li> <li>- Debug executable sizes are usually larger in size than release executable file sizes. This may be due to the additional side effects that exist in debug files for the purpose of debugging taking more space.</li> </ul>
Lessons learned	
<ul style="list-style-type: none"> <li>- <b>assert()</b> can be used to prevent side-effect functions in debug builds from running in release builds.</li> <li>- the <b>#ifdef</b> and <b>#else</b> preprocessor directives allows parts of the program to selectively run depending on the build type.</li> <li>- Debug executable file sizes are typically larger than release executable file sizes.</li> </ul>	

**FIXING MY OWN CODE FROM THE MIDSEM EXAM – Valley Triangle**

30/05/2022- Monday, week 12

**Program aim**

Create a triangle valley ASCII shape with a user-chosen height. Use a function to print the triangle valley. (Note, I avoided looking at the answers in the feedback page to rely on memory and learning)

**Plan**

Create a nested loop starting with the incrementation of the height. Nest the following loops within the height loop with the following logic:

- **Left right-angle triangle shape:** print line of #'s while it is less than the height incrementation + 1. The plus one enables a '#' to be printed on the first line.
- **Spaces (prints upside-down isometric triangle):** print double spaces while they are less than the height – height incrementation – 1. The minus one prevents the double space from being printed on the last line. (Needs to be closed like: "\")
- **Right right-angle triangle shape:** the same as left triangle shape's loop logic.

**Implementation****Original Source code**

```

2  #include <stdio.h>
3
4  void print_valley(int depth);
5  int main(void)
6  {
7      int depth = 0;
8      printf("Valley depth? ");
9      scanf("%d", &depth);
10     print_valley(depth);
11     return 0;
12 }
13 void print_valley(int depth)
14 {
15     // height
16     for (int i = 0; i < depth; i++)
17     {
18         // left triangle
19         for (int j = 0; j <= i; j++)
20         {
21             printf("#");
22         }
23         printf("\\");
24         // spaces
25         for (int k = depth + 2; k >= (i * 2); --k)
26         {
27             printf(" ");
28         }
29         // right triangle
30         printf("/");
31         for (int l = 0; l <= i; l++)
32         {
33             printf("#");
34         }
35
36         // print newline
37         printf("\n");
38     }
39 }
```

**Fixed source code**

```

2  #include <stdio.h>
3
4  void print_valley(int depth);
5  int main(void)
6  {
7      int depth = 0;
8      printf("Valley depth? ");
9      scanf("%d", &depth);
10     print_valley(depth);
11     return 0;
12 }
13 void print_valley(int depth)
14 {
15     // height
16     for (int i = 0; i < depth; i++)
17     {
18         // left triangle
19         for (int j = 0; j < (i + 1); j++)
20         {
21             printf("#");
22         }
23         printf("\\");
24         // spaces
25         for (int k = 0; k < (depth - i - 1); k++)
26         {
27             printf(" ");
28         }
29         printf("/");
30         // right triangle
31         for (int j = 0; j < (i + 1); j++)
32         {
33             printf("#");
34         }
35         printf("\n");
36     }
37 }
```

**Output/s**

A depth of 5 is normal, the output is messed up for other values.

Valley depth? 5      Valley depth? 7  
 #\        /#  
 ##\        /##  
 ###\        /###  
 ####\        /####  
 #####\        /#####  
 #####\ /#####  
 #####\ /#####

Valley depth? 3  
 #\        /#  
 ##\        /##  
 ###\        /###  
 ####\        /####  
 #####\ /#####  
 #####\ /#####

Fixed (all depth values work!)  
 Valley depth? 5      Valley depth? 7      Valley depth? 3  
 #\        /#  
 ##\        /##  
 ###\        /###  
 ####\        /####  
 #####\ /#####  
 #####\ /#####  
 #####\ /#####  
 #####\ /#####  
 #####\ /#####  
 #####\ /#####

**Lessons learned**

- My code's output failed because I overcomplicated the logic! Take a step back and evaluate how to do things.
- I need to revise on using loops to print ASCII shapes more.
- Planning is important, even if it is time consuming ...

## 11x05a – Commenting

31/05/2022- Tuesday, week 12

### Program aim

From the given source code, add function comments to each function in a specific style.

### Given Source Code

```

4   float compute_bill(float starter, float main, float dessert);
5   void print_bill(float starter, float main, float dessert);
6
7   int main(void)
8   {
9       float starter_price = 0.0f;
10      float main_price = 0.0f;
11      float dessert_price = 0.0f;
12
13      printf("Starter price? ");
14      scanf("%f", &starter_price);
15
16      printf("Main price? ");
17      scanf("%f", &main_price);
18
19      printf("Dessert price? ");
20      scanf("%f", &dessert_price);
21
22      printf("\n");
23
24      print_bill(starter_price, main_price, dessert_price);
25
26      return 0;
27 }

```

```

29  float compute_bill(float starter, float main, float dessert)
30  {
31      return starter + main + dessert;
32  }
33
34  void print_bill(float starter, float main, float dessert)
35  {
36      printf("Starter: $%.2f\n", starter);
37      printf("Main:    $%.2f\n", main);
38      printf("Dessert: $%.2f\n", dessert);
39      printf("-----\n");
40      printf("Total:   ");
41      printf("$%.2f\n", compute_bill(starter, main, dessert));
42 }

```

### Implementation

#### Instructions & program output

"Add function comments to each function in the following style:"

```

/*
 * Function: compute_factorial
 * -----
 * Returns the factorial of the value input
 *
 * param: the number to compute the factorial of
 *
 * returns: the factorial of input
 */

```

#### Program output/s:

Starter price? 5.5	Starter price? 1000
Main price? 10.75	Main price? 2000
Dessert price? 7.6	Dessert price? 1500
Starter: \$ 5.50 Main:   \$ 10.75 Dessert: \$ 7.60 ----- Total:   \$ 23.85	
Starter: \$1000.00 Main:   \$2000.00 Dessert: \$1500.00 ----- Total:   \$4500.00	

#### Added Function Comments

##### compute\_bill

###### Source code

```

29  /**
30   * Function: compute_bill
31   * -----
32   * Returns the total bill of the starter, main, and dessert
33
34   * starter: price of starter as a real value
35   * main: price of main as a real value
36   * dessert: price of dessert as a real value
37
38   * returns: the sum of starter, main, and dessert
39  */
40  float compute_bill(float starter, float main, float dessert)
41  {
42      return starter + main + dessert;
43 }

```

###### hover pop-up menu

float starter

\* Function: compute\_bill  
 \* Returns the total bill of the starter, main, and dessert  
 \* starter: price of starter as a real value  
 \* main: price of main as a real value  
 \* dessert: price of dessert as a real value  
 \* returns: the sum of starter, main, and dessert

##### print\_bill

###### Source code

```

45 /**
46  * Function: print_bill
47  * -----
48  * Prints the price for starter, main, dessert, and total bill
49
50  * starter: price of starter as a real value
51  * main: price of main as a real value
52  * dessert: price of dessert as a real value
53
54  * returns: no return value
55  */
56 void print_bill(float starter, float main, float dessert)
57 {
58     printf("Starter: $%.2f\n", starter);
59     printf("Main:    $%.2f\n", main);
60     printf("Dessert: $%.2f\n", dessert);
61     printf("-----\n");
62     printf("Total:   ");
63     printf("$%.2f\n", compute_bill(starter, main, dessert));
64 }

```

###### hover pop-up menu

void print\_bill(float starter, float main, float dessert)

\* Function: print\_bill  
 \* Prints the price for starter, main, dessert, and total bill  
 \* starter: price of starter as a real value  
 \* main: price of main as a real value  
 \* dessert: price of dessert as a real value  
 \* returns: no return value

### Lessons learned

- I learnt how to create function comments using the multi-line comment technique `/**` along with the conventional method of writing function comments that include the function purpose, parameters, and returns.

11x06a - Read in Binary Arrays

31/05/2022- Tuesday, week 12

## Program aim

Write a program that can read in a binary file (floatdata.bin)

## Plan

1. Read in the first four bytes of `floatdata.bin` into an `int` to find how many floating-point values are in the file.
2. Next, read in the entire array of floating-point values (use a dynamic heap array to allocate and store this data, based on how many floating-point values are stored in the binary file).
3. Finally, compute the average of the float values and then print out the average.

## Errors/debugging

```

22 // read all of the floats in the binary file
23 for (int i = 0; i < num[0]; i++)
24 {
25     fread(read_floats, sizeof(float), 1, p_file);
26 }

```

when accidentally setting the argument in `fread` (representing element count) to 1 instead of the total number of floats to be read (`num[0]`), it outputs the following... →

```

Num float values: 100
[0]: -13.870781
[1]: 0.000000
[2]: 0.000000
[3]: 0.000000

```

## Implementation

## Source Code

```

3 #include <stdlib.h>
4
5 int main(void)
6 {
7     int num[4];
8     float average = 0.0f;
9     float sum = 0.0f;
10    float* read_floats = 0;
11
12    FILE* p_file = 0;
13
14    p_file = fopen("floatdata.bin", "rb");
15
16    // read first 4 bytes of file to find 'num' of floats
17    fread(num, sizeof(int), 1, p_file);
18
19    // dynamically allocate to array of floats based on 'num'
20    read_floats = malloc(sizeof(float) * num[0]);
21
22    // read all of the floats in the binary file
23    for (int i = 0; i < num[0]; i++)
24    {
25        fread(read_floats, sizeof(float), num[0], p_file);
26    }
27
28    fclose(p_file);
29
30    printf("Num float values: %d \n", num[0]);
31
32    for (int i = 0; i < num[0]; i++)
33    {
34        printf("[%d]: %f \n", i, read_floats[i]);
35    }
36
37    // compute average
38
39    for (int i = 0; i < num[0]; i++)
40    {
41        sum += read_floats[i];
42    }
43
44    average = sum / num[0];
45
46    printf("Sum: %f \n", sum);
47    printf("Average: %f", average);
48
49    return 0;
50

```

## Output/s / Testing

Total number of float values based on the first 4 bytes of floatdata.bin:

**Num float values: 100**

Output of printing each element in `read_floats` to check if every element is a valid float value

[0]: 91.760597	[95]: 68.174141
[1]: 40.363304	[96]: -46.152809
[2]: 53.259911	[97]: 96.669006
[3]: 54.510044	[98]: 3.844202
[4]: 78.981018	[99]: -13.870781
...	

Output of printing the sum (to check if it actually summed), and the printing of the average.

**Sum: 137.502991**  
**Average: 1.375030**

The average value matches to the value in the exercise



## Lessons learned

- I learnt how to read from a binary file for the first time!
- Remember to use “rb” to read binary files, and “wb” to write to binary files
- These are the parameters for `fread`: `fread(void *ptr, size_t size, size_t nmemb, FILE *stream)`

Pointer to a block of memory to be read into, size in bytes of each element to be read, number of elements, file pointer

**LAB 11: 11x07a - Write out Binary Arrays**

31/05/2022- Tuesday, week 12

**Program aim**

Write a program that writes 100 randomly generated ints between -100 and 100 inclusive to a binary file named **randoms.bin**

**Plan**

1. Using a for loop, generate 100 random numbers between -100 and 100 inclusive.
2. As it loops, write each random number to a binary file named **randoms.bin**
3. To check if it worked, view the file in a hex editor and use a for loop to print each value from the file to the console

**Errors/debugging**

```
[0]: 0 ← I only got 0's for the output of reading randoms.bin because I accidentally read from the file while it was opened in "wb" writing binary mode, so the program couldn't save any values...
```

```
[1]: 0
[2]: 0
[3]: 0
[4]: 0
```

To fix this, I reopened the file in "**rb**" reading binary mode and read the file's values from there.

**Implementation****Source Code**

```

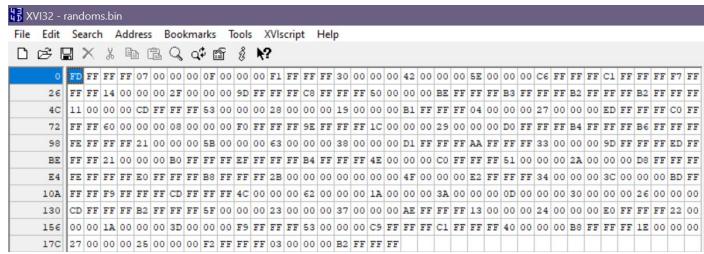
3  #include <stdlib.h>
4  #include <time.h>
5
6  int main(void)
7  {
8      srand(time(0));
9
10     int rand_num = 0;
11     int read_nums[100];
12     FILE* p_file = fopen("randoms.bin", "wb");
13
14     p_file = fopen("randoms.bin", "rb");
15
16     // 1. Using a for loop, generate 100 random numbers between -100 and 100 inclusive.
17     for (int i = 0; i < 100; i++)
18     {
19         //           MAX - MIN + MIN + 1
20         rand_num = rand() % (100 + 100) - 100 + 1;
21         // 2. As it loops, write each random number to a binary file named randoms.bin
22         fwrite(&rand_num, sizeof(int), 1, p_file);
23     }
24
25     fclose(p_file);
26
27     // Read the 100 int values from the file
28     p_file = fopen("randoms.bin", "rb");
29     fread(&read_nums, sizeof(int), 100, p_file);
30     fclose(p_file);
31
32     // 3. Use a for loop to print each value from the file to the console
33     for (int i = 0; i < 100; i++)
34     {
35         printf("%d: %d\n", i, read_nums[i]);
36     }
37
38     return 0;
39 }
```

**Output/s / Testing**

To test if the 100 values were saved to the file, I printed all 100 values read from **randoms.bin**.

[0]: -3	[95]: 39
[1]: 7	[96]: 37
[2]: 15	[97]: -14
[3]: -15	[98]: 3
[4]: 48	[99]: -78
...	

The hex editor shows that 100 values are stored in the file

**Lessons learned**

- Remember to only perform reading or writing operations when you have opened the file in the correct mode!

LAB 11: 11x08a - Count 1s and 0s

31/05/2022- Tuesday, week 12

## Program aim

Complete the given incomplete source code.

## Given Source Code

```

1 #define _CRT_SECURE_NO_WARNINGS
2 #include <stdio.h>
3
4 int count_1s(unsigned int value);
5 int count_0s(unsigned int value);
6
7 int main(void)
8 {
9     unsigned int input = 0;
10    int looping = 1;
11
12    while (looping)
13    {
14        printf("> ");
15        scanf("%d", &input);
16    }
17
18    int num_zeroes = count_0s(input);
19    int num_ones = count_1s(input);
20
21    printf("%u contains ", input);
22    printf("%u zeroes and ", num_zeroes);
23    printf("%u ones\n\n", num_ones);
24
25    return 0;
26
27
28 // TODO: Define count_1s function:
29 // TODO: Define count_0s function:
30
31

```

## Implementation

## Instructions

"Using sequence, selection, repetition and bitwise operators, define the **count\_1s** function such that it will count how many bits are set to 1 within the value parameter. The function must then return the count."

## MORE INFO

**bit\_to\_check** checks each bit of the value using the `<<` bit shift operator. When the if statement detects that the 1 in the **bit\_to\_check** binary matches up to a 1 in **value**, the count of 1's is increased. →

## Changed source code

```

29 // TODO: Define count_1s function:
30 int count_1s(unsigned int value)
31 {
32     int num_bytes = sizeof(int);
33     int num_bits = num_bytes * 8;
34     int count = 0;
35
36     for (int i = num_bits - 1; i >= 0; --i)
37     {
38         int bit_to_check = (1 << i);
39
40         if (value & bit_to_check)
41         {
42             count++;
43         }
44     }
45
46     return count;
47

```

"Using sequence, selection, repetition and bitwise operators, define the **count\_0s** function such that it will count how many bits are set to 0 within the value parameter. The function must then return the count."

## MORE INFO

The for loop uses the same logic as **count\_1's**, but the IF statement uses IF NOT (! logical operator) to count the 0's. →

```

49 // TODO: Define count_0s function:
50 int count_0s(unsigned int value)
51 {
52     int num_bytes = sizeof(int);
53     int num_bits = num_bytes * 8;
54     int count = 0;
55
56     for (int i = num_bits - 1; i >= 0; --i)
57     {
58         int bit_to_check = (1 << i);
59
60         if (!(value & bit_to_check))
61         {
62             count++;
63         }
64     }
65
66     return count;
67

```

## LAB 11: 11x08a - Count 1s and 0s

**31/05/2022- Tuesday, week 12**

## Output/s

(Example outputs shown in the exercise match up to my program's output)

## Lessons learned

- It seems like **4294967295** is the max value for unsigned integers, so values above this are only read at **4294967295**.
  - I learnt how to use bitwise operators (such as **&**, and **<<** left bit shift) for the first time! I still have yet to understand bitwise operations fully, however.

**LAB 11: 11x09a - Encrypted ASCII**

31/05/2022- Tuesday, week 12

**Program aim**

Decrypt the secret ASCII code stored in **encrypted.bin** by bit-flipping each byte in the file. Print the resulting ASCII value.

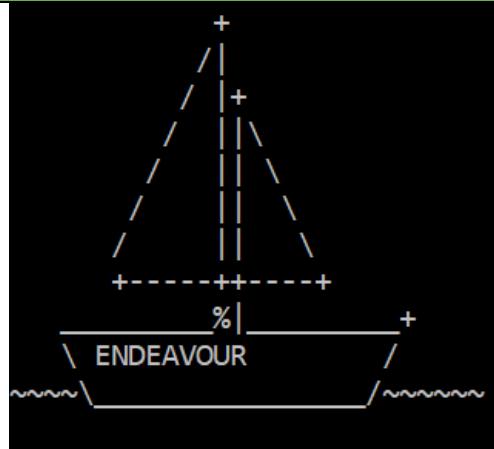
**Plan**

1. Create 2 variables: the read character and the bit-flipped character.
2. Open **encrypted.bin** in read-binary mode.
3. Loop while a single character read (**fgetc**) from **encrypted.bin** is NOT **EOF** (end of file)
4. Assign the bit flipped character as character using the bitwise NOT operator (~)
5. Print the bit-flipped character
6. End loop, close the file.

**Implementation****Source Code**

```

1 #define _CRT_SECURE_NO_WARNINGS
2 #include <stdio.h>
3
4 int main(void)
5 {
6     char character = '\0';
7     char bflipped_char = '\0';
8
9     FILE* p_file = fopen("encrypted.bin", "rb");
10
11    while ((character = fgetc(p_file)) != EOF)
12    {
13        bflipped_char = ~character;
14        printf("%c", bflipped_char);
15    }
16
17    fclose(p_file);
18
19    return 0;
20 }
```

**Output/s / Testing****Lessons learned**

- I learnt how to decrypt a binary file using bitwise operators by decrypting and printing each character so that it outputs an ASCII image!

**LAB 11: 11x10a - Save and Sort**

31/05/2022- Tuesday, week 12

**Program aim**

Complete the given source code. The program should save 5 integers from the user as an array, and sort it (using **qsort**). Sorted and unsorted arrays must be saved to text files: **unsorted.txt** and **sorted.txt**.

**Given Source Code**

```

1 #define _CRT_SECURE_NO_WARNINGS
2 #include <stdio.h>
3 [ #include <stdlib.h>
4
5 int compare(const void* a, const void* b);

```

**Errors/debugging**

Please enter 5 integers:

```

Input 1: 6
Input 2: 3
Input 3: 5
Input 4: 8
Input 5: 3

```

Array input was: { 6, 3, 5, 8, 3 }

← When running the program, I found it only stopped at the point where it's supposed to write to the file. I found this happened because I forgot to put the file name (stream) in the parameters of **fprintf()**, causing the file writing to fail!

```

fprintf("{ %d, %d, %d, %d, %d }",
        int_input[0], int_input[1], int_input[2], int_input[3], int_input[4]);

```

**Implementation****Instructions**

"Ask the user to input five integers. The program must use an array to store these values."

**Added source code**

```

10 int main(void)
11 {
12     int int_input[5];
13
14     // Ask user to input 5 integers, store to array.
15     printf("Please enter 5 integers:");
16     printf("\n\n");
17
18     for (int i = 0; i < 5; i++)
19     {
20         printf("Input %d: ", (i+1));
21         scanf("%d", &int_input[i]);
22     }

```

"Print out the contents of each array element and then save the array to a text file named **unsorted.txt**."

```

24
25     printf("\n");
26     printf("Array input was: { %d, %d, %d, %d, %d }",
27            int_input[0], int_input[1], int_input[2], int_input[3], int_input[4]);
28     printf("\n");
29
30     // Save unsorted array to unsorted.txt
31     FILE* p_file1 = fopen("unsorted.txt", "w");
32
33     fprintf(p_file1, "{ %d, %d, %d, %d, %d }",
34             int_input[0], int_input[1], int_input[2], int_input[3], int_input[4]);
35
36     fclose(p_file1);
37
38     printf("Unsorted array has been saved to the 'unsorted.txt' file.");
39     printf("\n\n");

```

"Sort the array using the **qsort** function from **stdlib.h**, from lowest to highest."

**compare function:**

```

3 #include <stdlib.h>
4
5 int compare(const void* a, const void* b)
6 {
7     return (*(int*)a - *(int*)b);
8 }

```

**compare function used with **qsort**:**

```

40
41     // Call qsort and compare function to sort array.
42     qsort(int_input, 5, sizeof(int), compare);
43
44     printf("Sorted is: { %d, %d, %d, %d, %d }",
45            int_input[0], int_input[1], int_input[2], int_input[3], int_input[4]);
46     printf("\n");

```

**LAB 11: 11x10a - Save and Sort (continued)**

31/05/2022- Tuesday, week 12

**Implementation****Instructions**

"Print out the sorted array and save the array to disk in a file named **sorted.txt**."

**Added source code**

```

49 FILE* p_file2 = fopen("sorted.txt", "w");
50
51 fprintf(p_file2, "{ %d, %d, %d, %d, %d }",
52         int_input[0], int_input[1], int_input[2], int_input[3], int_input[4]);
53
54 fclose(p_file2);
55
56 printf("Sorted array has been saved to the 'sorted.txt' file.");
57
58 return 0;
59

```

**Output/s**

Please enter 5 integers:

Input 1: -5  
 Input 2: -7  
 Input 3: -2  
 Input 4: 5  
 Input 5: 7

Array input was: { -5, -7, -2, 5, 7 }  
 Unsorted array has been saved to the 'unsorted.txt' file.  
 Sorted is: { -7, -5, -2, 5, 7 }  
 Sorted array has been saved to the 'sorted.txt' file.

Please enter 5 integers:

Input 1: 9  
 Input 2: 2  
 Input 3: 4  
 Input 4: 6  
 Input 5: 7

Array input was: { 9, 2, 4, 6, 7 }  
 Unsorted array has been saved to the 'unsorted.txt' file.  
 Sorted is: { 2, 4, 6, 7, 9 }  
 Sorted array has been saved to the 'sorted.txt' file.

Please enter 5 integers:

Input 1: 100  
 Input 2: 124  
 Input 3: 513  
 Input 4: 5315  
 Input 5: 1

Array input was: { 100, 124, 513, 5315, 1 }  
 Unsorted array has been saved to the 'unsorted.txt' file.  
 Sorted is: { 1, 100, 124, 513, 5315 }  
 Sorted array has been saved to the 'sorted.txt' file.

unsorted.txt - Notepad

File Edit Format View Help  
 { -5, -7, -2, 5, 7 }

sorted.txt - Notepad

File Edit Format View Help  
 { -7, -5, -2, 5, 7 }

unsorted.txt - Notepad

File Edit Format View Help  
 { 9, 2, 4, 6, 7 }

sorted.txt - Notepad

File Edit Format View Help  
 { 2, 4, 6, 7, 9 }

unsorted.txt - Notepad

File Edit Format View Help  
 { 100, 124, 513, 5315, 1 }

sorted.txt - Notepad

File Edit Format View Help  
 { 1, 100, 124, 513, 5315 }

**Lessons learned**

- Remember the **fprintf** parameters are: **fprintf(FILE \*stream, const char \*format, ...)**

File stream / name of file pointer, print line

## Week 12 lecture 035 Notes- Events, Dialogs, Windows GDI Drawing

1/06/2022- Wednesday, week 12

Notes Retrieved from Steffan Hooper (2022)

REMEMBER: `#include windows.h` when working with window interface.

**WinProc**: message handler function, allows program to respond to window events- like mouse left/right clicks and keys.

`LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)`

Use `switch(msg)` ... to handle cases for different buttons pressed... Which button is pressed is stored in `wParam`. `hwnd` allows window to be shown, ex: `MessageBox(hwnd ...)`.

Cause window to flash with `FlashWindowEx(address of flash struct variable);`

`WM_CREATE` allows adding controls to a window. Use `case WM_CREATE:`

`CreateWindow()` can add static text, buttons to a window. Use to set ID of buttons, can use `enum` for the IDs so that the IDs are named and easier to work with.

`WM_COMMAND` responds to the click of a button. Use `case WM_COMMAND:`

Button ID is sent in `wParam`, use it to detect which button is pressed.

Use `BS_CHECKBOX` as an argument in `CreateWindow()` to create a checkbox.

Use a `bool (True/False)` with `WM_COMMAND` to check if the checkbox is on/off.

Visually check the checkbox using `CheckDlgButton(...)`.

`static` in front of variable creates persistent data throughout various function calls (not recommended to use)

Retrieve text using `GetWindowText()`. Get length of text before using it with `GetWindowTextLength()`.

Create top-level menus using `CreateMenu()` ; ex: `HMENU hHelpMenu = CreateMenu();`

There are many other functions ...

GDI drawing – can draw to the screen! Use paint message: `WM_PAINT`.

Can randomly populate screen with pixels with loops and setting random values for `x` and `y` in `SetPixel()` .

**PERSONAL PROJECT #1 – Random ASCII emoticon generator (continued)**

1/06/2022- Wednesday, week 12

**Program aim**

Create a program that randomly generates ASCII emoticons. The user can keep generating random emoticons until they want it to stop. **ADD MODULARITY (FUNCTIONS), ENUM FOR NO MAGIC NUMBERS, AND VALIDATION**

**Implementation**

Source Code	Output/s
<pre> 2  #include &lt;stdio.h&gt; 3  #include &lt;time.h&gt; 4  #include &lt;stdlib.h&gt; 5 6  // TODO: use enums 7  enum Check // for no magic numbers 8  { 9      FALSE, 10     TRUE 11 }; 12 13 // TODO: modularise program 14 void print_emoji(char* eyes[], char* mouths[], int rand_eye, int rand_mouth); 15 char get_input(void); // get input &amp; validate 16 17 int main(void) 18 { 19     srand(time(0)); 20 21     int generate_emoji = TRUE; 22     char user_input = '\0'; 23 24     char* ascii_eyes[] = {"-", "O", "~", "U", "X", "+", "^", "@", "+"}; 25     char* ascii_mouths[] = { "-", ".", "W", "o", "&lt;", "3"}; 26 27     while (generate_emoji == TRUE) 28     { 29         int rand_eye = rand() % sizeof(rand_eye); // dynamic length using sizeof 30         int rand_mouth = rand() % sizeof(rand_mouth); 31 32         print_emoji(ascii_eyes, ascii_mouths, rand_eye, rand_mouth); 33 34         printf("\n"); 35 36         user_input = get_input(); 37 38         if (user_input == 'y') 39         { 40             generate_emoji = TRUE; 41         } 42         else if (user_input == 'n') 43         { 44             generate_emoji = FALSE; 45         } 46     } 47 48     printf("\nGoodbye!"); 49 50     return 0; 51 } 52 53 void print_emoji(char* eyes[], char* mouths[], int rand_eye, int rand_mouth) 54 { 55     printf("%c%c%c", *eyes[rand_eye], *mouths[rand_mouth], *eyes[rand_eye]); 56 } 57 58 // TODO: add input validation 59 char get_input(void) 60 { 61     char user_input = '\0'; 62 63     while (user_input != 'y' &amp;&amp; user_input != 'n') 64     { 65         printf("Generate another emoticon (y/n)? "); 66         scanf(" %c", &amp;user_input); 67 68         while (getchar() != '\n'); 69 70         if (user_input != 'y' &amp;&amp; user_input != 'n') 71         { 72             printf("Invalid input! \n"); 73         } 74     } 75 76     return user_input; 77 }</pre>	<p>Normal output</p> <p>(u.u) Generate another emoticon (y/n)? y (~o~) Generate another emoticon (y/n)? y (OwO) Generate another emoticon (y/n)? y (~_~) Generate another emoticon (y/n)? y (u_u) Generate another emoticon (y/n)? y (~_~) Generate another emoticon (y/n)? y (uwu) Generate another emoticon (y/n)? y (uou) Generate another emoticon (y/n)? y (OwO) Generate another emoticon (y/n)? y (~_~) Generate another emoticon (y/n)? y (~w~) Generate another emoticon (y/n)? n  Goodbye!</p>
	<p>Output for invalid input/s</p> <p>(-o-) Generate another emoticon (y/n)? a Invalid input! Generate another emoticon (y/n)? kjaskdj Invalid input! Generate another emoticon (y/n)? 1 Invalid input! Generate another emoticon (y/n)? 210839210 Invalid input! Generate another emoticon (y/n)? 5.5 Invalid input! Generate another emoticon (y/n)? @@@*(&amp;\$ Invalid input! Generate another emoticon (y/n)? y (OoO) Generate another emoticon (y/n)? n  Goodbye!</p>

## **PERSONAL PROJECT #1 – Random ASCII emoticon generator (continued)**

**1/06/2022- Wednesday, week 12**

### **Lessons learned**

- `getchar()` is a function that gets an unsigned character from stdin (standard input). This is useful for input validation. [https://www.tutorialspoint.com/c\\_standard\\_library/c\\_function\\_getchar.htm](https://www.tutorialspoint.com/c_standard_library/c_function_getchar.htm)
- From the TODOs from the previous iteration of this program, I added functions for increased modularity, an enumeration for avoiding using magic numbers for checking `generate_emoji`, and input validation.
- I changed the input to a char ('y'/'n') to make the program more 'user friendly'.
- I learnt that using `sizeof()` for the char c-string arrays will provide their size/length. This is helpful for making the random number generation dynamic, because I can add/subtract characters for the eyes/mouth arrays without changing the random number generator's range.

## 12x01a - Coloured Sail Boat

2/06/2022- Thursday (lab), week 12

### Program aim

From the given source code, modularise the printing of the ASCII art and colour the ASCII art using the **p1colour** library and header file.

### Implementation

#### Given Source Code

```

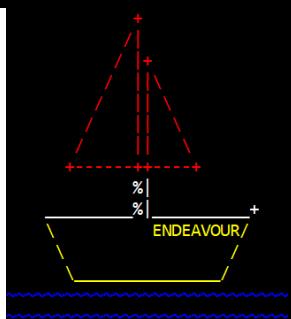
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  #include "p1colour.h"
5
6  void print_sails(void);
7  void print_lower_mast_and_deck(void);
8  void print_hull(void);
9  void print_water(void);
10
11 int main(void)
12 {
13     printf("      +\n");
14     printf("      /|\n");
15     printf("      / |+\n");
16     printf("      / ||\\|\n");
17     printf("      / || \\|\n");
18     printf("      / ||  \\|\n");
19     printf("      / ||   \\|\n");
20     printf("      +-----+\n");
21     printf("          %%|\n");
22     printf("          %%|_____+\n");
23     printf("      \\"          ENDEAVOUR/\n");
24     printf("      \\"          /\n");
25     printf("      \\"          /\n");
26     printf("~~~~~\n");
27     printf("~~~~~\n");
28
29     return 0;
30 }
```

#### Changed source code

```

1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  #include "p1colour.h"
5
6  void print_sails(void);
7  void print_lower_mast_and_deck(void);
8  void print_hull(void);
9  void print_water(void);
10
11 int main(void)
12 {
13     print_sails();
14     print_lower_mast_and_deck();
15     print_hull();
16     print_water();
17
18     return 0;
19 }
20
21 void print_sails(void)
22 {
23     set_text_colour(LIGHTRED, BLACK);
24
25     printf("      +\n");
26     printf("      /|\n");
27     printf("      / |+\n");
28     printf("      / ||\\|\n");
29     printf("      / || \\|\n");
30     printf("      / ||  \\|\n");
31     printf("      / ||   \\|\n");
32     printf("      +-----+\n");
33 }
34
35 void print_lower_mast_and_deck(void)
36 {
37     set_text_colour(WHITE, BLACK);
38
39     printf("          %%|\n");
40     printf("          %%|_____+\n");
41 }
42
43 void print_hull(void)
44 {
45     set_text_colour(YELLOW, BLACK);
46
47     printf("      \\"          ENDEAVOUR/\n");
48     printf("      \\"          /\n");
49     printf("      \\"          /\n");
50 }
51
52 void print_water(void)
53 {
54     set_text_colour(LIGHTBLUE, BLACK);
55
56     printf("~~~~~\n");
57     printf("~~~~~\n");
58 }
```

#### Output/s



#### Lessons learnt

25. I used the **p1colour.h** and **p1colour.lib** once again to colour ASCII art text.

## 12x02a - Classic Character

2/06/2022- Thursday (lab), week 12

### Program aim

```
classic.txt - Notepad
File Edit Format View Help
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15
15, 15, 15, 15, 4, 4, 4, 4, 4, 4, 15, 15, 15, 15, 15, 15
15, 15, 15, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 15
15, 15, 15, 6, 6, 6, 14, 14, 0, 14, 15, 15, 15, 15, 15
15, 15, 6, 14, 6, 14, 14, 14, 0, 14, 14, 14, 15, 15, 15
15, 15, 6, 14, 6, 14, 14, 14, 0, 14, 14, 14, 14, 15
15, 15, 15, 15, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 15, 15
15, 15, 15, 4, 4, 1, 4, 4, 4, 1, 4, 4, 4, 4, 15, 15, 15
15, 4, 4, 4, 4, 1, 4, 1, 1, 1, 4, 4, 4, 4, 4, 4, 15
15, 14, 14, 4, 1, 14, 1, 1, 14, 1, 4, 14, 14, 15
15, 14, 14, 14, 1, 1, 1, 1, 1, 1, 14, 14, 14, 15
15, 14, 14, 1, 1, 1, 1, 1, 1, 1, 1, 14, 14, 15
15, 15, 15, 1, 1, 1, 1, 15, 15, 15, 1, 1, 1, 15, 15
15, 15, 1, 1, 1, 15, 15, 15, 15, 1, 1, 1, 15, 15
15, 0, 0, 0, 0, 15, 15, 15, 15, 0, 0, 0, 0, 15
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15 |
```

Create a text file called **classic.txt** and paste the given numbers into the file.

Then, create a program that reads the numbers, and based on the number, output 2 spaces and colour them according to the number using **p1colour.h** and **p1colour.lib** functionality

← this is the **classic.txt** file I have created for this exercise

### Errors/debugging

I tried getting the integer values using **getc()** ... but this only returns individual char values (and the commas!!), so this ended up turning my integer values into their char values, thus outputting the incorrect colours and messing up the order and printing. To fix this, I instead iterated getting a numbers using **fscanf()** from each row until **EOF** is reached (using **getchar()** to detect this)

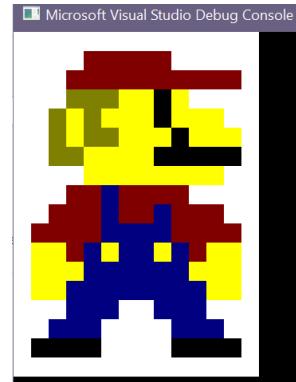
### Implementation

#### Source Code

```
2  #include <stdio.h>
3  #include "p1colour.h"
4
5  int main(void)
6  {
7      int getchar = 0;
8      int row[14];
9
10     FILE* p_file = fopen("classic.txt", "r");
11
12     fscanf(p_file, "%d %d %d",
13             &row[0], &row[1], &row[2], &row[3], &row[4], &row[5], &row[6], &row[7],
14             &row[8], &row[9], &row[10], &row[11], &row[12], &row[13]);
15     getchar = fgetc(p_file); // get char to check if EOF has been reached
16
17     while (getchar != EOF)
18     {
19         for (int i = 0; i < 14; i++)
20         {
21             set_text_colour(BLACK, row[i]); // foreground is black to check the numbers
22             printf(" ");
23         }
24         printf("\n");
25
26         fscanf(p_file, "%d %d %d",
27             &row[0], &row[1], &row[2], &row[3], &row[4], &row[5], &row[6], &row[7],
28             &row[8], &row[9], &row[10], &row[11], &row[12], &row[13]);
29         getchar = fgetc(p_file);
30     }
31
32     fclose(p_file);
33
34     return 0;
35 }
```

#### Output/s / Testing

##### Output with spaces



Output with numbers for checking if it matches **classic.txt**

```
22     printf("%2d", row[i]);
Microsoft Visual Studio Debug Console
15151515151515151515151515
15151515 4 4 4 4 1515151515
151515 4 4 4 4 4 4 4 4 4 15
151515 6 6 6 1414 1415151515
1515 6 14 6 141414 1414141515
1515 6 6 141414 1414141515
1515 6 6 14141414 1414141515
1515151514141414141414141515
151515 4 4 1 4 4 4 4 15151515
1515 4 4 4 1 4 4 4 1 4 4 4 15
15 4 4 4 4 1 1 1 1 4 4 4 4 15
151414 4 4 14 1 14 1 14141415
15141414 1 1 1 1 1 1 1 1 14141415
151414 1 1 1 1 1 1 1 1 1 141415
151515 1 1 1 151515 1 1 1 151515
1515 1 1 1 15151515 1 1 1 151515
151515151515151515151515151515
```

### Lessons learned

When reading numbers that are meant to be used as an integer, do not use **getc()** as it will read the numbers from the text file as a char instead! Use **fscanf()** instead, for example.

12x03a - Key Detection

2/06/2022- Thursday (lab), week 12

## Program aim

Write a program that detects which key is pressed, outputting the inputted key as text to the user. Use `conio.h` and the `_getch` function. Base my implementation on the given pseudocode algorithm.

## Implementation

## Given Pseudocode (DESIGN)

```

INT loop = 1

WHILE (loop)
    INT key_pressed = _getch();

    IF key_pressed == 224
        INT arrow_key = _getch();

        IF arrow_key == 72
            PRINT "Key: Up Arrow"
        ELSE IF arrow_key == 80
            PRINT "Key: Down Arrow"
        ELSE IF arrow_key == 75
            PRINT "Key: Left Arrow"
        ELSE IF arrow_key == 77
            PRINT "Key: Right Arrow"
        ENDIF

        PRINT newline
    ELSE
        PRINT "Key: "
        PRINT key_pressed
        PRINT " (ASCII: )"
        PRINT key_pressed
        PRINT newline
    ENDIF
ENDWHILE

```

## Source Code

```

3 | #include <conio.h>
4 |
5 | int main(void)
6 | {
7 |     int loop = 1;
8 |
9 |     while (loop)
10|     {
11|         int key_pressed = _getch();
12|
13|         if (key_pressed == 224)
14|         {
15|             int arrow_key = _getch();
16|
17|             if (arrow_key == 72)
18|             {
19|                 printf("Key: Up Arrow");
20|             }
21|             else if (arrow_key == 80)
22|             {
23|                 printf("Key: Down Arrow");
24|             }
25|             else if (arrow_key == 75)
26|             {
27|                 printf("Key: Left Arrow");
28|             }
29|             else if (arrow_key == 77)
30|             {
31|                 printf("Key: Right Arrow");
32|             }
33|             printf("\n");
34|
35|         }
36|         else
37|         {
38|             printf("Key: ");
39|             printf("%c", key_pressed);
40|             printf(" (ASCII: )");
41|             printf("%d", key_pressed);
42|             printf("\n");
43|         }
44|
45|
46|     }
47|     return 0;
}

```

## Output/s

Key: Up Arrow  
 Key: Down Arrow  
 Key: Left Arrow  
 Key: Right Arrow  
 Key: h (ASCII: )104  
 Key: i (ASCII: )105  
 Key: ! (ASCII: )33  
 Key: ? (ASCII: )63

Key: 1 (ASCII: )49  
 Key: 2 (ASCII: )50  
 Key: 3 (ASCII: )51  
 Key: 4 (ASCII: )52  
 Key: 5 (ASCII: )53  
 Key: 6 (ASCII: )54  
 Key: 7 (ASCII: )55  
 Key: 8 (ASCII: )56  
 Key: 9 (ASCII: )57  
 Key: 0 (ASCII: )48

Key: ! (ASCII: )33  
 Key: ? (ASCII: )63  
 Key: ! (ASCII: )33  
 Key: @ (ASCII: )64  
 Key: # (ASCII: )35  
 Key: \$ (ASCII: )36  
 Key: % (ASCII: )37  
 Key: ^ (ASCII: )94  
 Key: & (ASCII: )38  
 Key: \* (ASCII: )42  
 Key: ( (ASCII: )40  
 Key: ) (ASCII: )41

Key: H (ASCII: )72  
 Key: e (ASCII: )101  
 Key: l (ASCII: )108  
 Key: l (ASCII: )108  
 Key: o (ASCII: )111  
 Key: ! (ASCII: )33  
 Key: 1 (ASCII: )49  
 Key: 2 (ASCII: )50

## Lessons learned

- I learnt how to use `conio.h`'s `_getch` function for the first time to create a program that instantly attains user input without a buffer.
- Remember to call `getch` with an underscore: `_getch()`

## 12x04a - Important Msg Boxes

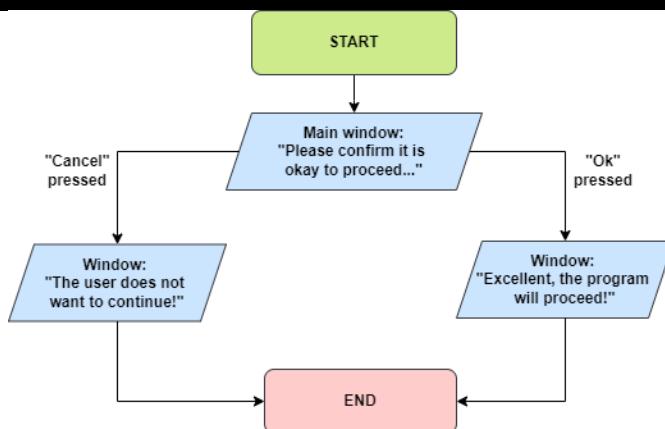
28/05/2022 – Saturday, week 11

### Program aim

Write a program that uses the Windows API function **MessageBox** to query the user to press “ok” or “cancel”.

Output different windows in response to what button the user clicked.

### Flowchart



### Error/debugging

```
warning C4100: 'nCmdShow': unreferenced formal parameter
warning C4100: 'lpCmdLine': unreferenced formal parameter
warning C4100: 'hPrevInstance': unreferenced formal parameter
warning C4100: 'hInstance': unreferenced formal parameter
```

← Warning **C4100:unreferenced formal parameter** is an unavoidable warning with using the **WinMain** function from **Windows.h**.

### Implementation

#### Source Code

```

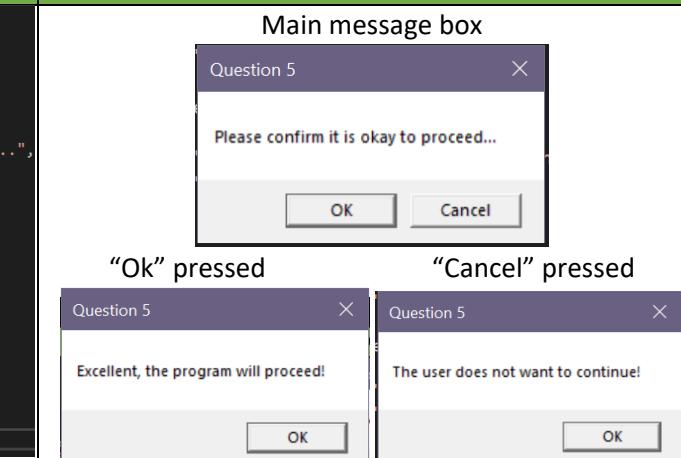
1  #include <Windows.h>
2
3  int WINAPI WinMain(HINSTANCE hInstance,
4      HINSTANCE hPrevInstance,
5      LPSTR lpCmdLine, int nCmdShow)
6  {
7      int result = MessageBox(NULL, L"Please confirm it is okay to proceed...",  

8          L"Question 5", MB_OKCANCEL);
9      if (IDOK == result)
10     {
11         MessageBox(NULL, L"Excellent, the program will proceed!",  

12             L"Question 5", MB_OK);
13     }
14     else if (IDCANCEL == result)
15     {
16         MessageBox(NULL, L"The user does not want to continue!",  

17             L"Question 5", MB_OK);
18     }
19     return 0;
20 }
```

#### Output/s / Testing



### Lessons learned

- Disable “treat warnings as errors” to stop the undefined parameters in **WinMain** from preventing the program to run
- I learnt how to create a window program for the first time, with message boxes!

## 12x04a - Important Msg Boxes (ADDED LOOP)

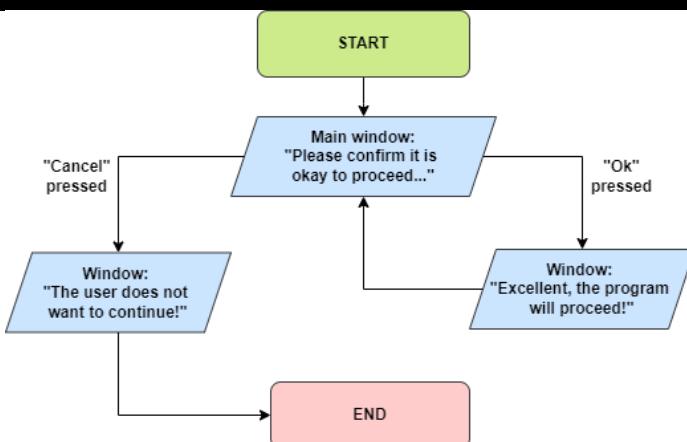
28/05/2022 – Saturday, week 11

I wanted to test how to use windows message boxes with loops:

### Program aim

Write a program that uses the Windows API function **MessageBox** to query the user to press “ok” or “cancel”. Output different windows in response to what button the user clicked. If the user clicks “ok”, loop until “cancel is pressed”

### Flowchart



### Implementation

#### Source Code

```

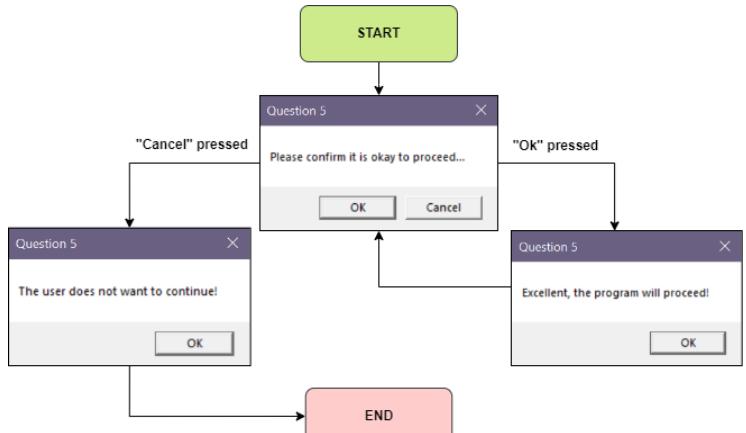
1 #include <Windows.h>
2
3 int WINAPI WinMain(HINSTANCE hInstance,
4                     HINSTANCE hPrevInstance,
5                     LPSTR lpCmdLine, int nCmdShow)
6 {
7     int loop = 1;
8
9     while (loop)
10    {
11        int result = MessageBox(NULL, L"Please confirm it is okay to proceed...",  

12                             L"Question 5", MB_OKCANCEL);
13
14        if (IDOK == result)
15        {
16            MessageBox(NULL, L"Excellent, the program will proceed!",  

17                         L"Question 5", MB_OK);
18        }
19        else if (IDCANCEL == result)
20        {
21            MessageBox(NULL, L"The user does not want to continue!",  

22                         L"Question 5", MB_OK);
23            loop = 0;
24        }
25    }
26
27    return 0;
28 }
  
```

#### Output/s / Testing (as a flowchart)



### Lessons learned

- I learnt how to use **Windows.h** message boxes with loops!

## Week 12 lecture 036 Notes- Summary

3/06/2022- Friday, week 12

Notes Retrieved from Steffan Hooper (2022)

### Revision Ideas:

1. Adhere to the programming standard:
  - Well-named variables, constants (CAPITAL LETTERS), functions, types
  - No global variables, no **goto**
  - Good source code whitespace layout
  - Good commenting practices
  - Variable initialisation
  - Good scoping practices
2. Input and output, sequence, selection, iteration, and functions
  - Function prototypes and definitions
  - Order of operations
  - Logical (**&&**, **||**, **!**) and relational (**==**, **!=**, **>=...**) operators
  - **return** and return type
  - REMEMBER conditional operator: **w:t:f → ? : true : false**
  - **for**, **while**, **do**
  - **switch**, **case**, **default**, **break**
  - **stdlib.h**, **time.h**, **math.h**, **Windows.h**
  - colour using **palcolour.h**
3. Storing and manipulating data with variables and arrays
  - Declaring arrays and variables
  - The stack and heap difference
  - Types short, long, double
  - **[]** array notation vs **\*** pointer notation
  - **strcmp**, **strcpy**, **strcat**, **sprint**, **atoi**, **atof**
4. Storing and manipulating data with structures and file storage
  - struct declaration & members
  - dot **.** operator
  - struct variable instance declaration
  - structures and functions
  - **->** operator (member access, pointers to structures)
  - reading and writing text and binary files
  - **FILE\***, **fopen**, **fclose**, **fprintf**, **fscanf**, **fgetc**, **fread**, **fwrite**
5. Construct and maintain small-scale programs
  - VS enterprise 2017 editor, compiler, linker, output window
  - Testing/preprocessor, debugging/debugger, preprocessor directives and macros
  - Macros: **#include** and **#define**

I will not be creating any files for the final exam; they will be pre-made!

There will be no binary file reading the exam 😊

When thinking of solutions, “keep it simple”

While debugging, I can move the breakpoint pointer to the beginning of the program to keep current data and run the program again at a certain point. (Simulates looping).

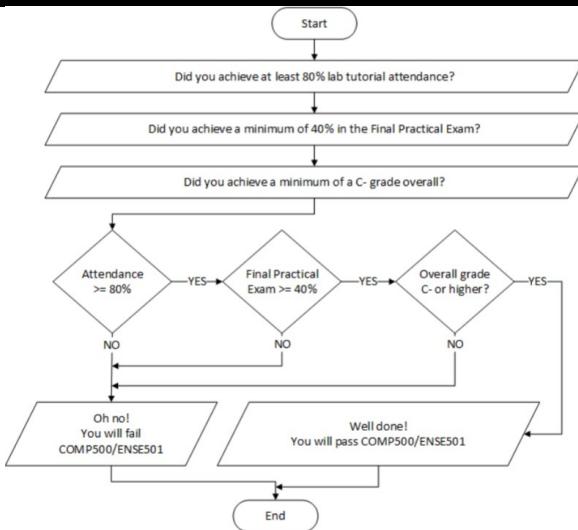
Other programming languages to explore: Python (my first language), C++, C#, and Java

12x05a - Course Result GUI

3/06/2022- Friday, week 12

**Program aim**

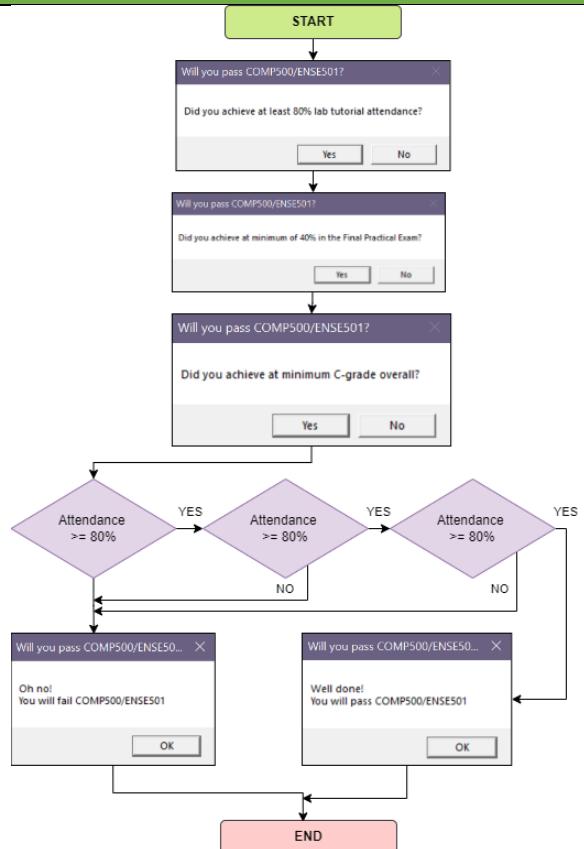
Using the Windows API function **MessageBox**, write a program that queries and informs the user on whether they passed COMP500/ENSE501 through a series of questions- based on the given flowchart

**Given Flowchart****Implementation****Source Code**

```

1  #include <Windows.h>
2
3  int WINAPI WinMain(HINSTANCE hInstance,
4      HINSTANCE hPrevInstance,
5      LPSTR lpCmdLine, int nCmdShow)
6  {
7
8      int attendance = MessageBox(NULL,
9          L"Did you achieve at least 80% lab tutorial attendance?",
10         L"Will you pass COMP500/ENSE501?", MB_YESNO);
11
12     int exam_result = MessageBox(NULL,
13         L"Did you achieve at minimum of 40% in the Final Practical Exam?",
14         L"Will you pass COMP500/ENSE501?", MB_YESNO);
15
16     int min_grade = MessageBox(NULL,
17         L"Did you achieve at minimum C-grade overall?",
18         L"Will you pass COMP500/ENSE501?", MB_YESNO);
19
20     if (IDYES == attendance && IDYES == exam_result && IDYES == min_grade)
21     {
22         MessageBox(NULL,
23             L"Well done! \nYou will pass COMP500/ENSE501",
24             L"Will you pass COMP500/ENSE501?", MB_OK);
25     }
26     else if (IDNO == attendance || IDNO == exam_result || IDNO == min_grade)
27     {
28         MessageBox(NULL,
29             L"Oh no! \nYou will fail COMP500/ENSE501",
30             L"Will you pass COMP500/ENSE501?", MB_OK);
31     }
32
33     return 0;
34 }
```

If the user presses "no" for any inquiry, they will end up with the "fail" messagebox.

**Output/s as a flowchart****Lessons learned**

I learnt how to use the Windows API function **MessageBox** with multiple selection: **if/else** and logical/relational operators

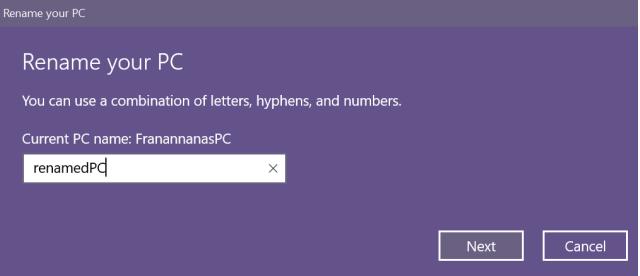
12x06a - User, Machine, Time

3/06/2022- Friday, week 12

## Program aim

Use windows API to output the username, machine name, date, and time to the **console**.

## Implementation

Source Code	Output/s / Testing
<pre> 1  #include &lt;Windows.h&gt; 2  #include &lt;stdio.h&gt; 3  #include &lt;lmcons.h&gt; // UNLEN is defined in lmcoms.h 4 5  int main(void) 6  { 7      SYSTEMTIME systime; 8      GetLocalTime(&amp;systime); 9 10     // DATE 11     wchar_t date[256]; 12     wsprintf(date, L"%d/%d/%d", // wide sprintf 13             systime.wDay, systime.wMonth, systime.wYear); 14 15     // TIME 16     wchar_t time[256]; 17     wsprintf(time, L"%d:%d:%d", 18             systime.wHour, systime.wMinute, systime.wSecond); 19 20     // USERNAME 21     wchar_t username[UNLEN + 1]; 22     DWORD username_size = UNLEN + 1; 23     GetUserName(username, &amp;username_size); 24 25     // COMPUTER NAME 26     wchar_t computername[MAX_COMPUTERNAME_LENGTH + 1]; 27     DWORD compname_size = MAX_COMPUTERNAME_LENGTH + 1; 28     GetComputerName(computername, &amp;compname_size); 29 30     // PRINTING EVERYTHING 31     printf("User name: "); 32     wprintf(username); 33     printf("\n"); 34 35     printf("Machine name: "); 36     wprintf(computername); 37     printf("\n"); 38 39     printf("Date: "); 40     wprintf(date); 41     printf("\n"); 42 43     printf("Time: "); 44     wprintf(time); 45     printf("\n"); 46 47     return 0; 48 }</pre>	<pre>User name: franc Machine name: FRANANNANASPC Date: 3/6/2022 Time: 15:37:2</pre> <p>Output 3 minutes later...</p> <pre>User name: franc Machine name: FRANANNANASPC Date: 3/6/2022 Time: 15:41:13</pre> <p>Time has changed!</p> <p>Renaming PC...</p> 
	<p>New output:</p> <pre>User name: franc Machine name: RENAMEDPC Date: 3/6/2022 Time: 15:53:30</pre>

## Lessons learned

- Use **wprintf** to print wide text (note that format specifiers don't seem to work with it...), use **wchar\_t** to create wide character type variables, and use **wsprintf** to "print into" (equivalent to **sprintf**).
- When printing to the console and not using windows interface, set the application to *Win32 Console*, and use **int main(void)** ; entry point.

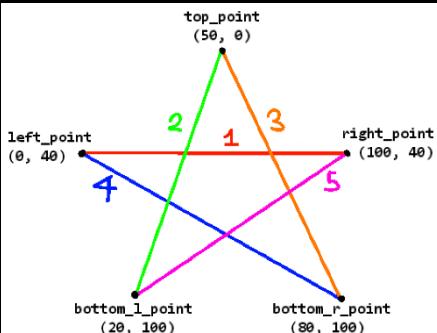
## 12x07b - GDI Stars

3/06/2022- Friday, week 12

### Program aim

Using the Windows API and GDI, create a function named `draw_five_point_star_at` with parameters that control its position and colour. Call the function multiple times to print the stars in a 3 by 5 grid in different colours.

### Plan (How to draw the lines)



← For the function: `draw_five_point_star_at`, this diagram was drawn to help me understand the base coordinates, how to label the lines in the source code, and where each point is placed (`top_point`, `left_point`, etc.)

### Implementation

#### Source Code (I used the given source code: W12 Example Code 034a - Basic Window as a base)

##### Coordinate struct and function prototypes

```

1  #include <Windows.h>
2
3  struct Coords
4  {
5      int x;
6      int y;
7  };
8
9  void draw_five_point_star_at(HDC hdc, int move_x, int move_y, int r, int g, int b);
10 void draw_background(HDC hdc);

```

##### `draw_five_point_star_at` function definition

```

130 void draw_five_point_star_at(HDC hdc, int move_x, int move_y, int r, int g, int b)
131 {
132     struct Coords top_point;
133     struct Coords left_point;
134     struct Coords right_point;
135     struct Coords bottom_l_point;
136     struct Coords bottom_r_point;
137
138     top_point.x = 50 + move_x;
139     top_point.y = 0 + move_y;
140
141     left_point.x = 0 + move_x;
142     left_point.y = 40 + move_y;
143
144     right_point.x = 100 + move_x;
145     right_point.y = 40 + move_y;
146
147     bottom_l_point.x = 20 + move_x;
148     bottom_l_point.y = 100 + move_y;
149
150     bottom_r_point.x = 80 + move_x;
151     bottom_r_point.y = 100 + move_y;
152
153     // Set colour of star
154     HPEN my_pen = CreatePen(PS_SOLID, 1, RGB(r, g, b));
155     OldPen = SelectObject(hdc, my_pen);
156
157     // Draw 5 lines of star
158     MoveToEx(hdc, left_point.x, left_point.y, 0); // line 1 (horizontal line)
159     LineTo(hdc, right_point.x, right_point.y);
160
161     MoveToEx(hdc, top_point.x, top_point.y, 0); // line 2
162     LineTo(hdc, bottom_l_point.x, bottom_l_point.y);
163
164     MoveToEx(hdc, top_point.x, top_point.y, 0); // line 3
165     LineTo(hdc, bottom_r_point.x, bottom_r_point.y);
166
167     MoveToEx(hdc, left_point.x, left_point.y, 0); // line 4
168     LineTo(hdc, bottom_r_point.x, bottom_r_point.y);
169
170     MoveToEx(hdc, right_point.x, right_point.y, 0); // line 5
171     LineTo(hdc, bottom_l_point.x, bottom_l_point.y);
172
173     // delete pen object to prevent memory leak
174     SelectObject(hdc, OldPen);
175     DeleteObject(my_pen);
176 }

```

##### `draw_background` function definition

```

178 void draw_background(HDC hdc)
179 {
180     // grey background
181     HBRUSH my_brush = CreateSolidBrush(RGB(165, 165, 165));
182     HBRUSH old_brush = SelectObject(hdc, my_brush);
183
184     Rectangle(hdc, 0, 0, 700, 440);
185
186     SelectObject(hdc, old_brush);
187     DeleteObject(my_brush);
188 }

```

##### WM\_PAINT case for msg in WndProc (call

##### `draw_five_point_star_at` to draw each star)

```

12 LRESULT CALLBACK WndProc(HWND hwnd, UINT msg,
13     WPARAM wParam, LPARAM lParam)
14 {
15     switch (msg)
16     {
17         case WM_PAINT:
18         {
19             PAINTSTRUCT ps;
20             HDC hdc = BeginPaint(hwnd, &ps);
21
22             draw_background(hdc);
23
24             // RED RGB: (255, 0, 0)
25             // YELLOW RGB: (255, 255, 0)
26             // WHITE RGB: (255, 255, 255)
27             // BLUE RGB: (0, 0, 255)
28             // CYAN RGB: (0, 255, 255)
29             // MAGENTA RGB: (255, 0, 255)
30             // GREEN RGB: (0, 255, 0)
31
32             draw_five_point_star_at(hdc, 40, 20, 255, 0, 0);
33             draw_five_point_star_at(hdc, 160, 20, 0, 0, 0);
34             draw_five_point_star_at(hdc, 280, 20, 255, 255, 0);
35             draw_five_point_star_at(hdc, 400, 20, 255, 255, 0);
36             draw_five_point_star_at(hdc, 520, 20, 0, 0, 255);
37
38             draw_five_point_star_at(hdc, 40, 140, 0, 255, 255);
39             draw_five_point_star_at(hdc, 160, 140, 255, 0, 255);
40             draw_five_point_star_at(hdc, 280, 140, 0, 0, 255);
41             draw_five_point_star_at(hdc, 400, 140, 255, 0, 0);
42             draw_five_point_star_at(hdc, 520, 140, 0, 255, 0);
43
44             draw_five_point_star_at(hdc, 40, 280, 255, 255, 0);
45             draw_five_point_star_at(hdc, 160, 280, 0, 255, 255);
46             draw_five_point_star_at(hdc, 280, 280, 255, 0, 255);
47             draw_five_point_star_at(hdc, 400, 280, 0, 255, 0);
48             draw_five_point_star_at(hdc, 520, 280, 255, 0, 255);
49
50             EndPaint(hwnd, &ps);
51         }
52     break;
53 }

```

## 12x07b - GDI Stars (continued)

3/06/2022- Friday, week 12

### Output/s



### Lessons learned & reflection

- Don't pass `hwnd` as a parameter for `draw_five_point_star_at` for the purpose of putting `PAINTSTRUCT` and `BeginPaint` in that function, as this will prevent the program from drawing more than one star!
- There is probably a better way to print all of the stars, which is iterating through a for loop. However, I cannot assign the colours specific to the required output from the exercise.
- I used a commented guide for the RGB colours, however, I could have created a `struct` with members: `r`, `g`, and `b` and assigned a struct variable to each colour...
- I think there is a way to colour the background without having to draw a rectangle to fill the page, however, I could not find how to do this, nor could I find documentation online.

**12x07b - GDI Stars- FULL SOURCE CODE (1)**

3/06/2022- Friday, week 12

```
#include <Windows.h>

struct Coords
{
    int x;
    int y;
};

void draw_five_point_star_at(HDC hdc, int move_x, int move_y, int r, int g, int b);
void draw_background(HDC hdc);

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg,
    WPARAM wParam, LPARAM lParam)
{
    switch (msg)
    {
        case WM_PAINT:
        {
            PAINTSTRUCT ps;
            HDC hdc = BeginPaint(hwnd, &ps);

            draw_background(hdc);

            // RED RGB: (255, 0, 0)
            // YELLOW RGB: (255, 255, 0)
            // WHITE RGB: (255, 255, 255)
            // BLUE RGB: (0, 0, 255)
            // CYAN RGB: (0, 255, 255)
            // MAGENTA RGB: (255, 0, 255)
            // GREEN RGB: (0, 255, 0)

            draw_five_point_star_at(hdc, 40, 20, 255, 0, 0);
            draw_five_point_star_at(hdc, 160, 20, 0, 0, 0);
            draw_five_point_star_at(hdc, 280, 20, 255, 255, 0);
            draw_five_point_star_at(hdc, 400, 20, 255, 255, 255);
            draw_five_point_star_at(hdc, 520, 20, 0, 0, 255);

            draw_five_point_star_at(hdc, 40, 140, 0, 255, 255);
            draw_five_point_star_at(hdc, 160, 140, 255, 0, 255);
            draw_five_point_star_at(hdc, 280, 140, 0, 0, 255);
            draw_five_point_star_at(hdc, 400, 140, 255, 0, 0);
            draw_five_point_star_at(hdc, 520, 140, 0, 255, 0);

            draw_five_point_star_at(hdc, 40, 280, 255, 255, 255);
            draw_five_point_star_at(hdc, 160, 280, 0, 255, 255);
            draw_five_point_star_at(hdc, 280, 280, 255, 0, 255);
            draw_five_point_star_at(hdc, 400, 280, 0, 255, 0);
            draw_five_point_star_at(hdc, 520, 280, 255, 255, 0);

            EndPaint(hwnd, &ps);
        }
        break;
        case WM_CLOSE:
        {
            DestroyWindow(hwnd);
        }
        break;
        case WM_DESTROY:
        {
            PostQuitMessage(0);
        }
        break;
    }
}

// Continued on page 2
```

**12x07b - GDI Stars- FULL SOURCE CODE (2)**

3/06/2022- Friday, week 12

```

        default:
        {
            return DefWindowProc(hwnd, msg, wParam, lParam);
        }
    }
    return 0;
}

int WINAPI WinMain(HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR params, int nCmdShow)
{
    const wchar_t g_szClassName[] = L"COMP500DEMO";
    MSG message;
    HWND hwnd;
    WNDCLASSEX wc;
    ZeroMemory(&wc, sizeof(wc));

    wc.cbSize = sizeof(WNDCLASSEX);
    wc.style = 0;
    wc.lpfnWndProc = WndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(COLOR_WINDOW);
    wc.lpszMenuName = NULL;
    wc.lpszClassName = g_szClassName;
    wc.hIconSm = LoadIcon(NULL, IDI_APPLICATION);

    if (!RegisterClassEx(&wc))
    {
        MessageBox(0, L"Error Registering Window!",
                  L"Error", MB_OK);

        return 0;
    }

    hwnd = CreateWindowEx(WS_EX_CLIENTEDGE,
                         g_szClassName,
                         L"GDI Stars",
                         WS_OVERLAPPEDWINDOW,
                         CW_USEDEFAULT, CW_USEDEFAULT,
                         680, 460, // WINDOW DIMENSIONS
                         NULL, NULL, hInstance, NULL);

    if (NULL == hwnd)
    {
        MessageBox(0, L"Error Creating Window!", L"Error", MB_OK);

        return 0;
    }
    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);

    while (GetMessage(&message, NULL, 0, 0) > 0)
    {
        TranslateMessage(&message);
        DispatchMessage(&message);
    }

    return message.wParam;
}
// Continued on page 3

```

**12x07b - GDI Stars- FULL SOURCE CODE (3)****3/06/2022- Friday, week 12**

```

void draw_five_point_star_at(HDC hdc, int move_x, int move_y, int r, int g, int b)
{
    struct Coords top_point;
    struct Coords left_point;
    struct Coords right_point;
    struct Coords bottom_l_point;
    struct Coords bottom_r_point;

    top_point.x = 50 + move_x;
    top_point.y = 0 + move_y;

    left_point.x = 0 + move_x;
    left_point.y = 40 + move_y;

    right_point.x = 100 + move_x;
    right_point.y = 40 + move_y;

    bottom_l_point.x = 20 + move_x;
    bottom_l_point.y = 100 + move_y;

    bottom_r_point.x = 80 + move_x;
    bottom_r_point.y = 100 + move_y;

    // Set colour of star
    HPEN my_pen = CreatePen(PS_SOLID, 1, RGB(r, g, b));
    HPEN old_pen = SelectObject(hdc, my_pen);

    // Draw 5 lines of star
    MoveToEx(hdc, left_point.x, left_point.y, 0); // line 1 (horizontal line)
    LineTo(hdc, right_point.x, right_point.y);

    MoveToEx(hdc, top_point.x, top_point.y, 0); // line 2
    LineTo(hdc, bottom_l_point.x, bottom_l_point.y);

    MoveToEx(hdc, top_point.x, top_point.y, 0); // line 3
    LineTo(hdc, bottom_r_point.x, bottom_r_point.y);

    MoveToEx(hdc, left_point.x, left_point.y, 0); // line 4
    LineTo(hdc, bottom_r_point.x, bottom_r_point.y);

    MoveToEx(hdc, right_point.x, right_point.y, 0); // line 5
    LineTo(hdc, bottom_l_point.x, bottom_l_point.y);

    // delete pen object to prevent memory leak
    SelectObject(hdc, my_pen);
    DeleteObject(my_pen);
}

void draw_background(HDC hdc)
{
    // grey background
    HBRUSH my_brush = CreateSolidBrush(RGB(165, 165, 165));
    HBRUSH old_brush = SelectObject(hdc, my_brush);

    Rectangle(hdc, 0, 0, 700, 440);

    SelectObject(hdc, my_brush);
    DeleteObject(my_brush);
}

```

**12x02a - Classic Character IMPROVED**

5/06/2022- Saturday, week 12

## Program aim

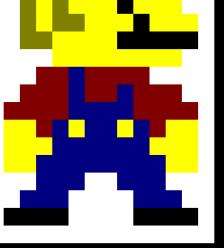
Create a text file called **classic.txt** and paste the given numbers into the file.

Then, create a program that reads the numbers, and based on the number, output 2 spaces and colour them according to the number using **p1colour.h** and **p1colour.lib** functionality

← this is the **classic.txt** file I have created for this exercise

Find a simpler solution to the exercise! And add the ability to print other coloured text files

## Implementation

Source Code	Output/s / Testing
<pre> 1 #define _CRT_SECURE_NO_WARNINGS 2 #include &lt;stdio.h&gt; 3 #include "p1colour.h" 4 5 int main(void) 6 { 7     int getint = 0; // scanned integer 8     char current = '\0'; // get character 9     char file_name[255]; 10 11    // Inquire user for filename 12    printf("File name? "); 13    scanf("%s", &amp;file_name); 14    FILE* p_file = fopen(file_name, "r"); 15 16    // feof works the same as detecting EOF 17    while (!feof(p_file)) 18    { 19        fscanf(p_file, " %d", &amp;getint); 20 21        set_text_colour(getint, getint); 22        printf(" "); 23 24        current = fgetc(p_file); 25 26        if (current != ',') 27        { 28            printf("\n"); 29        } 30    } 31 32    fclose(p_file); 33 34    return 0; 35 }</pre>	<p>File name? classic.txt</p>  <p>(User inputs <b>Character.txt</b>- a txt file provided by Xinyu)</p> <hr/> 

## Lessons learned

Creating a hard-coded array that is sized on the length of each row is not very modular. I can't print another numbered pixel art text file with a different length for each row! As it turns out, the simpler solution is to not use an array, but a single `int` variable that scans in a single integer. The space before `%d` (line 15) allows spaces to ignore being read. If a comma is not detected while reading a row, this means that it is the end of the line, thus a newline is printed. I also added the feature of the user choosing the filename!

## Evaluation: What am I not confident about?

5/06/2022- Sunday, week 12

Do alternative exercises for the following topics:

### Week 5

- Input validation, especially for clearing the buffer in loops.

### Week 7

- Logic for printing ASCII shapes using loops

### Week 9

- Pointers, struct pointers (->), binary file input and output.

### Week 10

- Multi-dimensional arrays, using parameters in main, dynamic memory, using other header files.

### Week 11

- `typedef` and function pointers, macros, `assert`, bitwise operators & bit-shifting & bit flags.

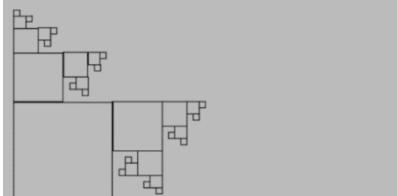
### Week 12

- Creating windows and GDI graphics!

## 12x07c - Box Fractal

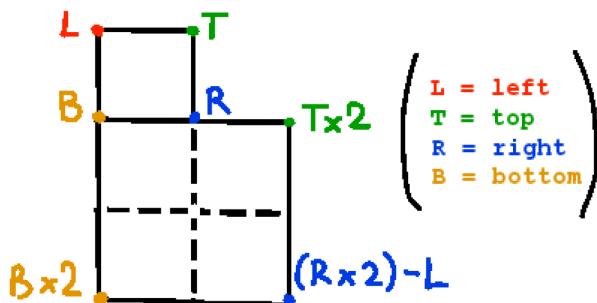
5/06/2022- Sunday, week 12

### Program aim



← Create a box fractal like in this image, using a recursive algorithm

### Plan (scaling the boxes)



← I drew this diagram to find how to scale/move the boxes...

### Implementation

#### Source Code (just the key parts)

#### Output/s / Testing

**draw\_box** function prototype

```
9 | int draw_box(HDC hdc, int left, int top, int right, int bottom, int count);
```

**draw\_box** definition

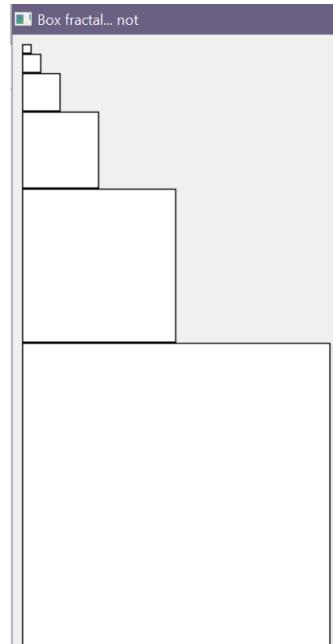
```
114 | int draw_box(HDC hdc, int left, int top, int right, int bottom, int count)
115 | {
116 |     if (count == 0)
117 |     {
118 |         return 0;
119 |     }
120 |     Rectangle(hdc, left, top, right, bottom);
121 |
122 |     draw_box(hdc, left, top * 2, right * 2 - left, bottom * 2, count - 1);
123 | }
```

^ implements recursion through calling itself

**draw\_box** call

```
11 | LRESULT CALLBACK WndProc(HWND hwnd, UINT msg,
12 | {
13 |     WPARAM wParam, LPARAM lParam)
14 | {
15 |     switch (msg)
16 |     {
17 |     case WM_PAINT:
18 |     {
19 |         PAINTSTRUCT ps;
20 |         HDC hdc = BeginPaint(hwnd, &ps);
21 |
22 |         int min_size = 8;
23 |
24 |         draw_box(hdc, min_size, min_size, min_size*2, min_size*2, 6);
25 |
26 |         EndPaint(hwnd, &ps);
27 |     } break;
28 | }
```

I ended up not creating a box fractal, but I just settled on creating a box that doubles in size and repeats vertically using a recursive function...



### Lessons Learned & reflection

- I was reminded about recursion. In this case, it was a function calling itself to double the size of the drawn square while repeating vertically.
- I was unable to finish this exercise by creating a fractal, as I didn't know how to rotate the shapes...

## 7x02b – AI Hobby choice

5/06/2022- Sunday, week 12

### Program aim

Create a simple AI program that helps the user select a hobby based upon the table below:

Location?	Active?	AI recommendation:
Outdoors	Yes	Tennis
Outdoors	No	Bird Watching
Indoors	Yes	Tenpin Bowling
Indoors	No	Stamp Collecting

### Implementation

#### Source Code

```

4 void print_ai_recommends(char location, char active);
5
6 int main(void)
7 {
8     char location = '\0';
9     char active = '\0';
10
11    printf("Location, indoor or outdoor (i/o)? ");
12    scanf(" %c", &location);
13
14    printf("Active hobby, yes or no (y/n)? ");
15    scanf(" %c", &active);
16
17    print_ai_recommends(location, active);
18
19    return 0;
20 }
21
22 void print_ai_recommends(char location, char active)
23 {
24     if (location == 'i' || location == 'I')
25     {
26         if (active == 'y' || active == 'Y')
27         {
28             printf("AI recommends Tenpin Bowling");
29         }
30         else if (active == 'n' || active == 'N')
31         {
32             printf("AI recommends Stamp Collecting ");
33         }
34     }
35     else if (location == 'o' || location == 'O')
36     {
37         if (active == 'y' || active == 'Y')
38         {
39             printf("AI recommends Tennis");
40         }
41         else if (active == 'n' || active == 'N')
42         {
43             printf("AI recommends Bird watching");
44         }
45     }
46     else
47     {
48         printf("Invalid input!");
49     }
50 }
```

#### Output/s / Testing

Location, indoor or outdoor (i/o)? i  
 Active hobby, yes or no (y/n)? y  
 AI recommends Tenpin Bowling

Location, indoor or outdoor (i/o)? I  
 Active hobby, yes or no (y/n)? n  
 AI recommends Stamp Collecting

Location, indoor or outdoor (i/o)? O  
 Active hobby, yes or no (y/n)? Y  
 AI recommends Tennis

Location, indoor or outdoor (i/o)? o  
 Active hobby, yes or no (y/n)? N  
 AI recommends Bird watching

Location, indoor or outdoor (i/o)? 3  
 Active hobby, yes or no (y/n)? 3  
 Invalid input!

Location, indoor or outdoor (i/o)? y  
 Active hobby, yes or no (y/n)? i  
 Invalid input!

### Lessons learned

- Just some basic selection with a function

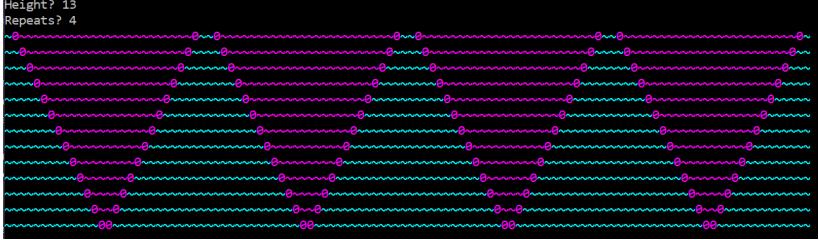
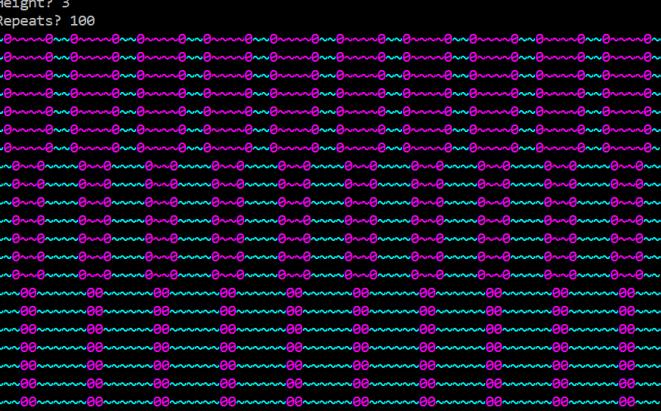
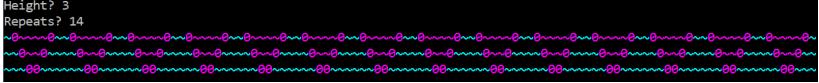
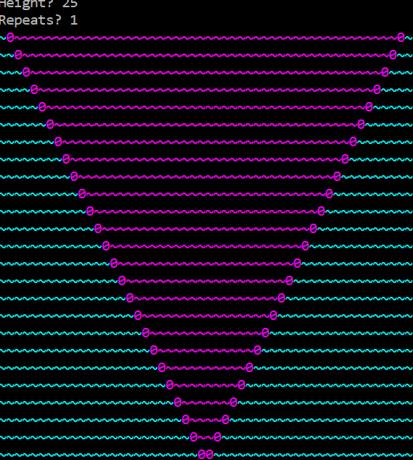
**PERSONAL PROJECT – coloured triangle pattern**

5/06/2022- Sunday, week 12

**Program aim**

Create a program that creates (using `p1colour.h`) a coloured zig-zag triangle pattern using loops and functions for modularity.

**Implementation**

Source Code	Output/s / Testing
<pre> 2  #include &lt;stdio.h&gt; 3  #include "p1colour.h" 4 5  void print_hashes(int count); 6  void print_spaces(int height, int count); 7 8  int main(void) 9  { 10     int height = 0; 11     int repeats = 0; 12 13     printf("Height? "); 14     scanf("%d", &amp;height); 15 16     printf("Repeats? "); 17     scanf("%d", &amp;repeats); 18 19     for (int count = 0; count &lt; height; count++) 20     { 21         for (int r = 0; r &lt; repeats; r++) 22         { 23             print_hashes(count); 24             print_spaces(height, count); 25             print_hashes(count); 26         } 27 28         printf("\n"); 29     } 30 31     return 0; 32 } 33 34 void print_hashes(int count) 35 { 36     set_text_colour(LIGHTCYAN, BLACK); 37 38     for (int h = 0; h &lt;= count; h++) 39     { 40         printf("~"); 41     } 42 } 43 44 void print_spaces(int height, int count) 45 { 46     set_text_colour(LIGHTMAGENTA, BLACK); 47 48     printf("0"); 49     for (int s = 0; s &lt; (height - count - 1); s++) 50     { 51         printf("~~"); 52     } 53     printf("0"); 54 }</pre>	   

**Lessons Learned & discussion**

This program uses the same logic as the “print valley” program. I played around with `p1colour.h` to create different colours in the outputted shape, I used functions to create modularity (with `print_hashes` and `print_spaces`, which are variables I forgot to rename...), and I added the ability to horizontally repeat the shape, which is done through another nested loop before calling the drawing/printing functions. A cool feature to add would be to tile the shape though...

# WEEK 13

## Alternative Exercises Plan (for Revision)

6/06/2022- Monday, week 13

### Lab 7

- 7x04c - Print Half Triangle
- 7x09d - Custom Rocket
- 7x08 – “is” series

### Lab 8

- 8x06b - Comparing Sizes
- 8x07b - Student Records

### Lab 9

- 9x01b - Pointer Knowledge
- 9x05b - Count Odds in Array
- 9x10c - Word Counter

### Lab 10

- 10x01b - Zero and Identity
- 10x03b - main Repeater
- 10x09b - typedef struct

### Lab 11

- 11x02b- QSort Floats
- 11x03d- BSearch argv

### Lab 12

- 12x05e - GUI Dice Roller

## 7x04c – Print half triangle

6/06/2022- Monday, week 13

## Program aim

Create a program that prints a half-triangle. Use the given function prototype to print the shape.

## Pseudocode

```

DECLARE FUNCTION void print_pattern(int total_rows)

START MAIN
    DECLARE height (integer)
    GET height from user input
    CALL print_pattern(height)
END MAIN

START PRINT_PATTERN(INT TOTAL_ROWS)
    FOR i = 0, while i < total_rows, increment i
        FOR j = total_rows, while j > i, decrement j
            PRINT space
        ENDFOR

        PRINT "/"
        FOR k = 0, while k < i, increment k
            PRINT "#"
        ENDFOR

        PRINT "|"
        PRINT newline
    ENDFOR
END PRINT_PATTERN(INT TOTAL_ROWS)

```

## Implementation

## Lessons learned

- I practiced some more ASCII shape printing using for loop logic...

## 7x09d - Custom Rocket

6/06/2022- Monday, week 13

Program aim	
Implementation	
Pseudocode (from exercise)	Source Code
<pre> DECLARE FUNCTION print_nozzle(int width) DECLARE FUNCTION print_body(int width, int height) DECLARE FUNCTION print_logo(int width)  START main     PRINT "Rocket body width (minimum 6)? "     READ body_width     VALIDATE body_width IS 6 or GREATER      PRINT "Rocket body height? "     READ body_height     VALIDATE body_height IS 0 or GREATER      CALL print_nozzle WITH body_width / 2     CALL print_body WITH body_height AND body_width     CALL print_logo WITH body_width     CALL print_body WITH body_height AND body_width     CALL print_nozzle WITH body_width / 2 END main </pre>	<pre> 1  #include &lt;stdio.h&gt; 2 3  void print_nozzle(int width); 4  void print_body(int width, int height); 5  void print_logo(int width); 6 7  int main(void) 8  { 9      int width = 0; 10     int height = 0; 11     int loop = 1; 12 13     while (loop) 14     { 15         printf("Rocket body width (minimum 6)? "); 16         scanf("%d", &amp;width); 17 18         if (width &lt; 6) 19         { 20             printf("Invalid input! Width must be greater than 6!\n"); 21         } 22         else 23         { 24             loop = 0; 25         } 26     } 27     loop = 1; 28     printf("\n"); 29 30     while (loop) 31     { 32         printf("Rocket body height? "); 33         scanf("%d", &amp;height); 34 35         if (height &lt; 0) 36         { 37             printf("Invalid input! Height must be greater than 0!\n"); 38         } 39         else 40         { 41             loop = 0; 42         } 43     } 44     loop = 1; 45     printf("\n"); 46 47     print_nozzle(width / 2); 48     print_body(width, height); 49     print_logo(width); 50     print_body(width, height); 51     print_nozzle(width / 2); 52 53 54     return 0; 55 } </pre>

7x09d - Custom Rocket (continued)

6/06/2022- Monday, week 13

## Implementation

My Pseudocode	Source Code
<pre>// ( r = row, s = space, s2 = space 2)  START print_nozzle(int width)     FOR int r = 0, while r &lt; width, increment r         FOR int s = width, while s &gt; r, decrement s             PRINT space         ENDFOR          PRINT "/"          FOR int s2 = 0, while s2 &lt; r, increment s2             PRINT space         ENDFOR          PRINT "\"         PRINT newline     END print_nozzle(int width)  // ( d = dashes, r = row, s = space)  START print_body(int width, int height)     PRINT "+"     FOR int d = 0, while d &lt; width, increment d         PRINT "-"     ENDFOR     PRINT "+"     PRINT newline      FOR int r = 0, while r &lt; height, increment r         PRINT " "         FOR int s = 0, while s &lt; width, increment s             PRINT space         ENDFOR         PRINT " "         PRINT newline     ENDFOR      PRINT "+"     FOR int d = 0, while d &lt; width, increment d         PRINT "-"     ENDFOR     PRINT "+"     PRINT newline END print_body(int width, int height)</pre>	<pre>57 void print_nozzle(int width) 58 { 59     for (int r = 0; r &lt; width; r++) // rows 60     { 61         for (int s = width; s &gt; r; s--) // spaces 1 62         { 63             printf(" "); 64         } 65         printf("/"); 66         for (int s2 = 0; s2 &lt; r; s2++) 67         { 68             printf(" "); 69         } 70         printf("\\"); 71         printf("\n"); 72     } 73 }  74 void print_body(int width, int height) 75 { 76     printf("+"); // first row 77     for (int d = 0; d &lt; width; d++) 78     { 79         printf("-"); 80     } 81     printf("+"); 82     printf("\n");  83     for (int r = 0; r &lt; height; r++) // rows 84     { 85         printf(" "); 86         for (int s = 0; s &lt; width; s++) // spaces 87         { 88             printf(" "); 89         } 90         printf(" "); 91         printf("\n"); 92     }  93     printf("+"); // last row 94     for (int d = 0; d &lt; width; d++) 95     { 96         printf("-"); 97     } 98     printf("+"); 99     printf("\n"); 100 }</pre>

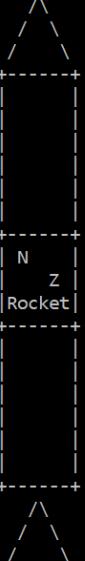
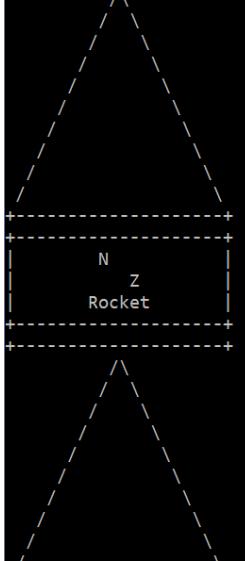
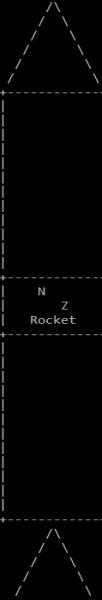
7x09d - Custom Rocket (continued)

6/06/2022- Monday, week 13

## Implementation

My Pseudocode	Source Code
<pre>( r = row, s = space )  START print_logo(int width)     FOR int r = 0, while r &lt; 3, increment r         PRINT " "         FOR int s = 0, while s &lt; ((width-6)/2), increment s             PRINT space     ENDFOR      IF r is 0         PRINT " N " // (1 space, "N", 4 spaces)     ELSE IF r is 1         PRINT "   Z " // (4 spaces, "Z", 1 space)     ELSE IF r is 2         PRINT "Rocket"  // WIDTH IS - 5 INSTEAD OF - 6 SO THAT ODD NUMBER WIDTHS HAVE AN EXTRA SPACE      FOR int s = 0, while s &lt; ((width-5)/2), increment s         PRINT space     ENDFOR      PRINT " "     PRINT newline END print_logo(int width)</pre>	<pre>105 void print_logo(int width) 106 { 107     for (int r = 0; r &lt; 3; r++) 108     { 109         printf(" "); 110         for (int s = 0; s &lt; ((width - 6) / 2); s++) 111         { 112             printf(" "); // spaces before logo 113         } 114         if (r == 0) 115         { 116             printf(" N "); // logo text... 117         } 118         else if (r == 1) 119         { 120             printf("   Z "); 121         } 122         else if (r == 2) 123         { 124             printf("Rocket"); 125         } 126         for (int s = 0; s &lt; ((width - 5) / 2); s++) 127         { 128             printf(" "); // spaces after logo 129         } 130         printf(" "); 131     } 132     printf("\n"); 133 } 134 }</pre>

## Output/s

Rocket body width (minimum 6)? 6 Rocket body height? 6 	Rocket body width (minimum 6)? 20 Rocket body height? 0 	Rocket body width (minimum 6)? 12 Rocket body height? 12 	<b>Input validation:</b> Rocket body width (minimum 6)? 5 Invalid input! Width must be greater than 6! Rocket body width (minimum 6)? 8 Rocket body height? -2 Invalid input! Height must be greater than 0! Rocket body height? 1 
---	---	--	---

## Lessons learned

- I practiced some more ASCII shape printing using loops - as well as input validation and function modularity

**7x08b – Is letter**

6/06/2022- Monday, week 13

**Program aim**

Write a program that checks if the user inputted an alphabetic letter (either capital or lowercase) using a side-effect free function. Return/print 1 if it is a letter, return 0 if it is not a letter.

**Implementation**

Source Code	Output/s / Testing
<pre> 1 #include &lt;stdio.h&gt; 2 3 char is_letter(char get_letter) 4 { 5     int check = 0; 6 7     if (get_letter &gt;= 'a' &amp;&amp; get_letter &lt;= 'z'     8         get_letter &gt;= 'A' &amp;&amp; get_letter &lt;= 'Z') 9     { 10        check = 1; 11    } 12 13    return check; 14 } 15 16 int main(void) 17 { 18     char get_letter = '\0'; 19 20     printf("&gt; "); 21     scanf(" %c", &amp;get_letter); 22 23     printf("%d", is_letter(get_letter)); 24 25     return 0; 26 }</pre>	<p>Not a letter:</p> <pre> &gt; &amp;a87s 0 &gt; 8 0 &gt; ? 0 </pre> <p>Is a letter:</p> <pre> &gt; A 1 &gt; g 1 &gt; Zz 1 </pre>
	<b>Lessons learned</b>

- I learnt how to check for capital & lowercase alphabetic input using a function...

## 7x08c – Is Digit

6/06/2022- Monday, week 13

### Program aim

Write a program that checks if the user inputted a digit (ASCII char digit between 0-9 inclusive) using a side-effect free function. Return/print 1 if it is a digit, return 0 if it is not a digit.

### Implementation

Source Code	Output/s / Testing
<pre> 1  #include &lt;stdio.h&gt; 2 3  char is_digit(char get_digit) 4  { 5      int check = 0; 6 7      if (get_digit &gt;= '0' &amp;&amp; get_digit &lt;= '9') 8      { 9          check = 1; 10     } 11 12     return check; 13 } 14 15 int main(void) 16 { 17     char get_digit = '\0'; 18 19     printf("&gt; "); 20     scanf(" %c", &amp;get_digit); 21 22     printf("%d", is_digit(get_digit)); 23 24     return 0; 25 }</pre>	<p>Is a digit:</p> <pre>&gt; 1 1 &gt; 4 1 &gt; 9 1 &gt; 10 1</pre> <p>Not a digit:</p> <pre>&gt; -5 0 &gt; &amp; 0 &gt; A 0 &gt; Hello! 0</pre>

### Lessons learned

- I learnt how to check for the input of a digit using a function...

7x08d – Is Hex Character

6/06/2022- Monday, week 13

## Program aim

Write a program that checks if the user inputted a hex symbol (ASCII char including 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E, and F) using a side-effect free function. Return/print 1 if it is a hex symbol, return 0 if it is not.

## Implementation

## Source Code

```

1 #include <stdio.h>
2
3 char is_hex_digit(char get_hex)
4 {
5     int check = 0;
6
7     if (get_hex >= '0' && get_hex <= '9' ||
8         get_hex >= 'A' && get_hex <= 'F')
9     {
10         check = 1;
11     }
12
13     return check;
14 }
15
16 int main(void)
17 {
18     char get_hex = '\0';
19
20     printf("> ");
21     scanf(" %c", &get_hex);
22
23     printf("%d", is_hex_digit(get_hex));
24
25     return 0;
26 }
```

## Output/s / Testing

Not a hex digit:

```

> G
0
> -5
0
> Hello
0
> b
0
> ...
> a
0
```

Is a hex digit:

```

> 1
1
> 5
1
> A
1
> F
1
> 9
1
```

## Lessons learned

- I learnt how to check for the input of a hex digit using a function...

**7x08d – Is P-N-Z**

6/06/2022- Monday, week 13

**Program aim**

Write a program that checks if the user inputted either a positive, negative, or zero number- using 3 side-effect free functions that check for each. Return/print 1 if it is a positive/negative/zero number, return 0 if it is not.

**Given Source Code**

```

1  #include <stdio.h>
2
3  int is_positive(int number);
4  int is_negative(int number);
5  int is_zero(int number);
6
7  int main(void)
8  {
9      int input = 0;
10     printf("> ");
11     scanf("%d", &input);
12
13     return 0;
14 }
```

**Implementation****Changed Source Code**

```

3  int is_positive(int number);
4  int is_negative(int number);
5  int is_zero(int number);
6
7  int main(void)
8  {
9      int input = 0;
10     printf("> ");
11     scanf("%d", &input);
12
13     printf("Calling is_positive: %d\n", is_positive(input));
14     printf("Calling is_negative: %d\n", is_negative(input));
15     printf("Calling is_zero: %d\n", is_zero(input));
16
17     return 0;
18 }
19
20 int is_positive(int number)
21 {
22     int check = 0;
23
24     if (number > 0)
25     {
26         check = 1;
27     }
28
29     return check;
30 }
31
32 int is_negative(int number)
33 {
34     int check = 0;
35
36     if (number < 0)
37     {
38         check = 1;
39     }
40
41     return check;
42 }
43
44 int is_zero(int number)
45 {
46     int check = 0;
47
48     if (number == 0)
49     {
50         check = 1;
51     }
52
53     return check;
54 }
```

**Output/s / Testing**

```

> 5
Calling is_positive: 1
Calling is_negative: 0
Calling is_zero: 0

> 100
Calling is_positive: 1
Calling is_negative: 0
Calling is_zero: 0
```

```

> -34
Calling is_positive: 0
Calling is_negative: 1
Calling is_zero: 0
```

```

> -1
Calling is_positive: 0
Calling is_negative: 1
Calling is_zero: 0
```

```

> 0
Calling is_positive: 0
Calling is_negative: 0
Calling is_zero: 1
```

**Lessons learned**

- I learnt how to check if the integer input is positive/negative/zero using functions...

## 8x06b - Comparing Sizes

6/06/2022- Monday, week 13

### Program aim

Declare several **structs** with different numbers of **float** members and use **sizeof** to compare their sizes.

### Step-by-step plan

1. Declare a structure type named **Vector2D**, with two **float** members, named **x** and **y**.
2. Declare a structure type named **Vector3D**, with three **float** members, named **x**, **y** and **z**.
3. Declare a structure type named **Vector4D**, with four **float** members, named **x**, **y**, **z** and **w**.
4. Declare a structure type named **Matrix2D**, with four **float** members named **m11**, **m12**, **m21**, and **m22**.
5. Declare a structure type named **Matrix3D**, with nine **float** members named **m11**, **m12**, **m13**, **m21**, **m22**, **m23**, **m31**, **m32**, and **m33**.
6. Declare a structure type named **Matrix4D**, with sixteen **float** members named **m11**, **m12**, **m13**, **m14**, **m21**, **m22**, **m23**, **m24**, **m31**, **m32**, **m33**, **m34**, **m41**, **m42**, **m43**, and **m44**.
7. In the **main** function print the **sizeof** each structure type (**Vector2D**, **Vector3D**, **Vector4D**, **Matrix2D**, **Matrix3D** and **Matrix4D**) to the console.

### Implementation

#### Source Code (struct declarations)

```

3 // Declaring structures
4
5 struct Vector2D
6 {
7     float x;
8     float y;
9 };
10
11 struct Vector3D
12 {
13     float x;
14     float y;
15     float z;
16 };
17
18 struct Vector4D
19 {
20     float x;
21     float y;
22     float z;
23     float w;
24 };
25
26 struct Matrix2D
27 {
28     float m11;
29     float m12;
30     float m21;
31     float m22;
32 };
33
34 struct Matrix3D
35 {
36     float m11;
37     float m12;
38     float m13;
39     float m21;
40     float m22;
41     float m23;
42     float m31;
43     float m32;
44     float m33;
45 };
46
47 struct Matrix4D
48 {
49     float m11;
50     float m12;
51     float m13;
52     float m14;
53     float m21;
54     float m22;
55     float m23;
56     float m24;
57     float m31;
58     float m32;
59     float m33;
60     float m34;
61     float m41;
62     float m42;
63     float m43;
64     float m44;
65 };

```

#### Source Code (main)

```

67 int main(void)
68 {
69     // Printing sizeof() structures to the console
70
71     printf("sizeof Vector2D is: %d\n", sizeof(struct Vector2D));
72     printf("sizeof Vector3D is: %d\n", sizeof(struct Vector3D));
73     printf("sizeof Vector4D is: %d\n", sizeof(struct Vector4D));
74     printf("sizeof Matrix2D is: %d\n", sizeof(struct Matrix2D));
75     printf("sizeof Matrix3D is: %d\n", sizeof(struct Matrix3D));
76     printf("sizeof Matrix4D is: %d\n", sizeof(struct Matrix4D));
77
78 }
79

```

### OUTPUT

Microsoft Visual Studio Debug Console

```

sizeof Vector2D is: 8
sizeof Vector3D is: 12
sizeof Vector4D is: 16
sizeof Matrix2D is: 16
sizeof Matrix3D is: 36
sizeof Matrix4D is: 64

```

### Lessons learned

- It seems that the higher number of members a **struct** contains, the higher number of bytes a structure takes up in memory (size).
- It seems that each **float** member takes up  $(8/2 = 4)$  4 bytes in memory.

## 8x07b – Student Records

6/06/2022- Monday, week 13

### Program aim

From the given source code, implement the TODO instructions. Create a **struct** array that takes in the values in the table below and print the **struct** members in the **struct** array for each element using a loop.

<b>Student Index</b>	<b>Name</b>	<b>ID</b>	<b>Percentage</b>
0	Jane Smith	12345643	55.55%
1	John Jones	22334432	20.75%
2	Jill Johns	99876612	99.50%
3	Jack Bean	46802110	73.25%

### Given Source Code

```

1  #include <stdio.h>
2  #include <string.h>
3
4  struct Student
5  {
6      char name[32];
7      int id;
8      float percentage;
9  };
10
11 void print_student_details(struct Student to_print);
12
13 int main(void)
14 {
15     // TODO: Declare an array of four students:
16     // TODO: Set the first student's record:
17     // TODO: Set the second student's record:
18     // TODO: Set the third student's record:
19     // TODO: Set the fourth student's record:
20     // TODO: Iterate through each student in the array,
21     // TODO: and call print_student_details for each student:
22
23     return 0;
24
25
26 // TODO: Define the print_student_details function here:

```

### Implementation

#### Source Code

```

1  #include <stdio.h>
2  #include <string.h>
3
4  struct Student
5  {
6      char name[32];
7      int id;
8      float percentage;
9  };
10
11 void print_student_details(struct Student to_print);
12
13 int main(void)
14 {
15     // TODO: Declare an array of four students:
16     struct Student students[4];
17
18     // TODO: Set the first student's record:
19     sprintf(students[0].name, "Jane Smith");
20     students[0].id = 12345643;
21     students[0].percentage = 55.55f;
22
23     // TODO: Set the second student's record:
24     sprintf(students[1].name, "John Jones");
25     students[1].id = 22334432;
26     students[1].percentage = 20.75f;
27
28     // TODO: Set the third student's record:
29     sprintf(students[2].name, "Jill Johns");
30     students[2].id = 99876612;
31     students[2].percentage = 99.50f;
32
33     // TODO: Set the fourth student's record:
34     sprintf(students[3].name, "Jack Bean");
35     students[3].id = 46802110;
36     students[3].percentage = 73.25f;
37
38     // TODO: Iterate through each student in the array,
39     // TODO: and call print_student_details for each student:
40     for (int i = 0; i < 4; i++)
41     {
42         print_student_details(students[i]);
43         printf("\n");
44     }
45
46     return 0;
47
48
49 // TODO: Define the print_student_details function here:
50 void print_student_details(struct Student to_print)
51 {
52     printf("ID is: %d\n", to_print.id);
53     printf("Name is: %s\n", to_print.name);
54     printf("Percentage is: %.2f", to_print.percentage);
55 }

```

#### Output/s / Testing

```

ID is: 12345643
Name is: Jane Smith
Percentage is: 55.55

ID is: 22334432
Name is: John Jones
Percentage is: 20.75

ID is: 99876612
Name is: Jill Johns
Percentage is: 99.50

ID is: 46802110
Name is: Jack Bean
Percentage is: 73.25

```

### Lessons learned

- I learnt how to scan into and print the members of structure array elements.

## 9x01b - Pointer Knowledge

6/06/2022- Monday, week 13

*Test your knowledge by writing about the following questions in your Reporting Journal.*

### **1. What is an address?**

An address is the location of data in memory / where it is stored.

### **2. How do you get the address of a variable?**

The & (ampersand) address-of operator will allow the retrieval of a variable's address. Ex:  
`&input`

### **3. What is a pointer?**

A pointer is a variable that can store a memory address by “pointing to” its location in memory.

### **4. How do you declare a pointer?**

A pointer is declared by putting a \* (asterisk) in front of the **variable type** and before the **name**.

Ex: `int* number;`

### **5. How do you initialise a pointer?**

A pointer is initialised when it is assigned and declared at the same time. This is done by putting a \* (asterisk) in front of the **variable type** and before the **name**, then it is assigned to the **address of the variable**. Ex: `int* number = &number;`

### **6. What is dereferencing?**

Dereferencing/indirection is the process of taking the address pointer and putting an asterisk before its name, ex: `*number`, to get the data the pointer holds INSTEAD OF THE ADDRESS.

### **7. What is a null pointer?**

A null pointer points to a memory location where nothing is stored, which is 0.

Ex: `int* address = 0;`

### **8. How do you get access to the data that a pointer points to?**

Via dereferencing/indirection!

### **9. How do you find the address of the first element in an array?**

The name of an array points to the first element of the array! Ex: `printf("%p\n", array);`

### **10. How do you find the address of the last element in an array?**

Use the & address-of operator, followed by the array name, and followed by the index of the last element. Ex: `printf("%p\n", &array_name[last_element]);`

**9x01b - Pointer Knowledge (continued)**

6/06/2022- Monday, week 13

- 11. Write a loop to iterate through an array from the first element to the last element, using pointers.**

Using pointer arithmetic for an array:

Source code	Output
<pre>#include &lt;stdio.h&gt;  int main(void) {     int array_name[] = { 0, 1, 2, 3, 4, 5 };      for (int i = 0; i &lt; 6; i++)     {         printf("Element[%d]: %d, Address[%d]: %p\n",                i, *(array_name + i), i, (array_name + i));     }      return 0; }</pre>	<pre>Element[0]: 0, Address[0]: 009EF6D4 Element[1]: 1, Address[1]: 009EF6D8 Element[2]: 2, Address[2]: 009EF6DC Element[3]: 3, Address[3]: 009EF6E0 Element[4]: 4, Address[4]: 009EF6E4 Element[5]: 5, Address[5]: 009EF6E8</pre>

- 12. Write a loop to iterate through an array from the last element to the first element, using pointers.**

Using pointer arithmetic for an array:

Source code	Output
<pre>#include &lt;stdio.h&gt;  int main(void) {     int array_name[] = { 0, 1, 2, 3, 4, 5 };      for (int i = 5; i &gt;= 0; i--)     {         printf("Element[%d]: %d, Address[%d]: %p\n",                i, *(array_name + i), i, (array_name + i));     }      return 0; }</pre>	<pre>Element[5]: 5, Address[5]: 004FF78C Element[4]: 4, Address[4]: 004FF788 Element[3]: 3, Address[3]: 004FF784 Element[2]: 2, Address[2]: 004FF780 Element[1]: 1, Address[1]: 004FF77C Element[0]: 0, Address[0]: 004FF778</pre>

**13. What are the advantages to using pointers?**

- Allows locally stored variables to be accessed and changed in other functions via pass by reference!
- Pointers can allow multiple return values (from function parameters)
- Allows dynamically allocated memory

**14. What are the disadvantages to using pointers?**

- Allows locally stored variables to be accessed and changed in other functions via pass by reference, but can be bad for some scenarios.
- Pointers can be confusing...
- Introduces the potential for more errors.

## **9x01b - Pointer Knowledge (continued)**

**6/06/2022- Monday, week 13**

### **15. What is a wild pointer?**

A pointer that has been declared, but not initialised (has had no value assigned to it). For example, a wild pointer is: `int* wild_pointer;` but a good pointer is:

`int* good_pointer = 0; or int* good_pointer = &value`

### **16. What is a dangling pointer?**

A pointer that was stored in a valid address, but is now stored in an invalid address. This mostly happens when a local address is called in a different location (which doesn't work because that address is out of scope)

### **17. What is the difference between “pass by value” and “pass by reference”?**

In functions:

- Passing by reference is passing pointers (the address of a variable) as a parameter, ex:  
`void function(int* variable);`
- Passing by value is passing variables as themselves as a parameter, ex:  
`void function(int variable);`

**9x05b - Count Odds in Array**

6/06/2022- Monday, week 13

**Program aim**

From the given source code, complete the function `count_odds(int* data_array, int size)` which receives an integer via pointer, the arrays in size, and returns the number of odd numbers in the array.

**Given Source Code**

```

2   #include <stdio.h>
3
4   // TODO: Insert your function declaration here!
5
6   int main(void)
7   {
8       int data_array_1[] = { 1, 3, 5, 7, 9, 11 };
9       int data_array_2[] = { 2, -4, 6, -8, 10, -12, 14, -16 };
10      int data_array_3[] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0 };
11
12      int result_1 = count_odds(data_array_1, 6);
13
14      printf("data_array_1 has %d odd numbers.\n", result_1);
15
16      int result_2 = count_odds(data_array_2, 8);
17
18      printf("data_array_2 has %d odd numbers.\n", result_2);
19
20      int result_3 = count_odds(data_array_3, 11);
21
22      printf("data_array_3 has %d odd numbers.\n", result_3);
23
24
25      return 0;
26
27  // TODO: Insert your function definition here!

```

**Implementation****Source Code****Output/s / Testing**

```

1   #include <stdio.h>
2
3   // TODO: Insert your function declaration here!
4   int count_odds(int* data_array, int size);
5
6   int main(void)
7   {
8       int data_array_1[] = { 1, 3, 5, 7, 9, 11 };
9       int data_array_2[] = { 2, -4, 6, -8, 10, -12, 14, -16 };
10      int data_array_3[] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0 };
11
12      int result_1 = count_odds(data_array_1, 6);
13
14      printf("data_array_1 has %d odd numbers.\n", result_1);
15
16      int result_2 = count_odds(data_array_2, 8);
17
18      printf("data_array_2 has %d odd numbers.\n", result_2);
19
20      int result_3 = count_odds(data_array_3, 11);
21
22      printf("data_array_3 has %d odd numbers.\n", result_3);
23
24      return 0;
25
26
27  // TODO: Insert your function definition here!
28  int count_odds(int* data_array, int size)
29  {
30      int count = 0;
31
32      for (int i = 0; i < size; i++)
33      {
34          if (data_array[i] % 2 != 0)
35          {
36              count++;
37          }
38      }
39
40      return count;
41

```

Microsoft Visual Studio Debug Console  
**data\_array\_1 has 6 odd numbers.**  
**data\_array\_2 has 0 odd numbers.**  
**data\_array\_3 has 5 odd numbers.**

9x05b - Count Odds in Array (continued)

6/06/2022- Monday, week 13

## Program aim

Add another two array declarations and initialisations in the main function. Call count\_odd with the newly added arrays and print the results.

## Implementation

New Source Code	Output/s / Testing
<pre> 1  #include &lt;stdio.h&gt; 2 3  // TODO: Insert your function declaration here! 4  int count_odds(int* data_array, int size); 5 6 int main(void) 7 { 8     int data_array_1[] = { 1, 3, 5, 7, 9, 11 }; 9     int data_array_2[] = { 2, -4, 6, -8, 10, -12, 14, -16 }; 10    int data_array_3[] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0 }; 11    // TODO: add 2 new array declarations and initialisations 12    int data_array_4[] = { -32, 35, 70, 1300 }; 13    int data_array_5[] = { 0, 2, 4, 8, 4, 9, 9, 10, 11, 13, 6, 7 }; 14 15    int result_1 = count_odds(data_array_1, 6); 16    printf("data_array_1 has %d odd numbers.\n", result_1); 17 18    int result_2 = count_odds(data_array_2, 8); 19    printf("data_array_2 has %d odd numbers.\n", result_2); 20 21    int result_3 = count_odds(data_array_3, 11); 22    printf("data_array_3 has %d odd numbers.\n", result_3); 23 24    int result_4 = count_odds(data_array_4, 4); 25    printf("data_array_4 has %d odd numbers.\n", result_4); 26 27    int result_5 = count_odds(data_array_5, 13); 28    printf("data_array_5 has %d odd numbers.\n", result_5); 29 30    return 0; 31 } 32 33 // TODO: Insert your function definition here! 34 int count_odds(int* data_array, int size) 35 { 36     int count = 0; 37 38     for (int i = 0; i &lt; size; i++) 39     { 40         if (data_array[i] % 2 != 0) 41         { 42             count++; 43         } 44     } 45 46     return count; 47 }</pre>	 Microsoft Visual Studio Debug Console data_array_1 has 6 odd numbers. data_array_2 has 0 odd numbers. data_array_3 has 5 odd numbers. data_array_4 has 1 odd numbers. data_array_5 has 5 odd numbers.

## Lessons learned

- Call by reference allows functions to access the index of an array, without passing the array name by value like: **array\_name []**
- I need to start using call by reference with arrays when working with their elements/index...

## 9x10c - Word Counter

6/06/2022- Monday, week 13

### Program aim

Write a program that can read in a text file and count the number of words in the file.

The program must then print out how many words are in the file.

### Implementation

Source Code	Output/s / Testing
<pre> 1  #include &lt;stdio.h&gt; 2 3  int main(void) 4  { 5      char file_name[255]; 6      int count = 0; 7      char ch = '\0'; 8 9      printf("Filename? "); 10     scanf("%s", file_name); 11 12     FILE* p_file = fopen(file_name, "r"); 13 14     while ((ch = fgetc(p_file)) != EOF) 15     { 16         if (ch == ' ') // spaces = new word 17         { 18             count++; 19         } 20     } 21 22     printf("%s contains %d words.", file_name, count); 23 24     fclose(p_file); 25 26     return 0; 27 }</pre>	 Microsoft Visual Studio Debug Console Filename? outline.txt outline.txt contains 94 words.

### Lessons learned

- I learnt that all I had to do to count a word in a file is to count the number of spaces; however, this depends on the formatting of the text file! This was simple because the provided text file: **output.txt** was a single line of text.

10x01b - Zero and Identity

6/06/2022- Monday, week 13

## Program aim

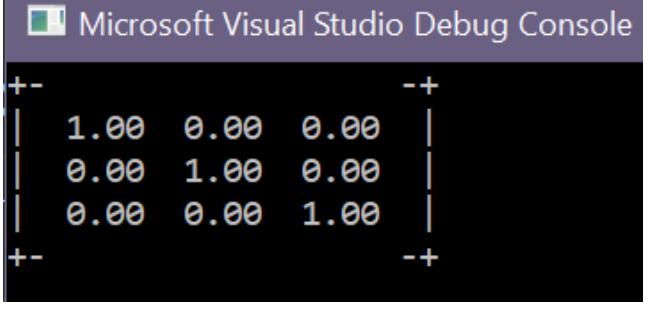
From the given source code and pseudocode, create a program that outputs a 2-dimensional array in table format with its values set to  $0.0\text{f}$ , except when the  $[x][y]$  array dimensions are equal, set those values to  $1.0\text{f}$ .

## Given Source code &amp; pseudocode

```
DECLARE mat_a AS A 3 by 3 float ARRAY
CALL zero WITH mat_a
CALL print WITH mat_a
CALL identity WITH mat_a
CALL print WITH mat_a
```

```
2 | #include <stdio.h>
3 |
4 | void zero(float matrix[3][3]);
5 | void identity(float matrix[3][3]);
6 | void print(float matrix[3][3]);
```

## Implementation

Source Code	Output/s / Testing
<pre>4   void zero(float matrix[3][3]); 5   void identity(float matrix[3][3]); 6   void print(float matrix[3][3]); 7   8   int main(void) 9   { 10       float mat_a[3][3]; 11   12       zero(mat_a); 13       identity(mat_a); 14       print(mat_a); 15   16       return 0; 17   } 18   19   void zero(float matrix[3][3]) 20   { 21       for (int x = 0; x &lt; 3; x++) 22       { 23           for (int y = 0; y &lt; 3; y++) 24           { 25               matrix[x][y] = 0.0f; 26           } 27       } 28   29   30   void identity(float matrix[3][3]) 31   { 32       for (int x = 0; x &lt; 3; x++) 33       { 34           for (int y = 0; y &lt; 3; y++) 35           { 36               if (x == y) 37               { 38                   matrix[x][y] = 1.0f; 39               } 40               else 41               { 42                   matrix[x][y] = 0.0f; 43               } 44           } 45       } 46   47   48   void print(float matrix[3][3]) 49   { 50       printf("+-\n"); 51       for (int x = 0; x &lt; 3; x++) 52       { 53           printf("   "); 54           for (int y = 0; y &lt; 3; y++) 55           { 56   57               printf(" %.2f ", matrix[x][y]); 58   59           } 56           printf("  \n"); 57       } 58       printf("+-\n"); 59   }</pre>	 <pre>+-   1.00  0.00  0.00     0.00  1.00  0.00     0.00  0.00  1.00   +-</pre> <p>^ when dimensions <math>[x][y]</math> are the same, the <math>0.0\text{f}</math> value is changed to <math>1.00\text{f}</math></p>

## Lessons learned

I learnt how to iterate through the values of a 2D array to assign, selectively assign, and print its values

10x03b - main Repeater

6/06/2022- Monday, week 13

## Program aim

Write a program that processes the arguments passed into **main** from the command line.

## Plan

1. The first argument is expected to be a number. The second argument is expected to be a single word.
  2. print the single word argument, the number of times specified by the first argument, each time separated by a comma.
  3. If the program is run with the command: `exercise.exe --help`, output messages to the console

## Error/debugging

My program was outputting nothing from the arguments: "5 cat" being passed into `main`, but this was because I was using `argv[0]` as the integer value, even though `argv[0]`'s argument is the file name!

## Implementation

## Source Code

```
1 #define _CRT_SECURE_NO_WARNINGS
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5
6 int main(int argc, char* argv[])
7 {
8     // atoi from stdlib converts char to int
9     int number = atoi(argv[1]);
10
11    if (argc > 3)
12    {
13        printf("Too many arguments!\n");
14    }
15    if ((strcmp(argv[1], "--help")) == 0)
16    {
17        printf("This program repeats a word a specified number of times.\n\n");
18        printf("To use this program: \n\n");
19        printf("    exercise.exe n word \n\n");
20        printf("Where n is the number of times to repeat the text 'word'");
21    }
22    else
23    {
24        for (int i = 0; i < number; ++i)
25        {
26            printf("%s", argv[2]);
27            if (i < (number - 1))
28            {
29                printf(", ");
30            }
31        }
32    }
33
34    return 0;
35 }
```

## Output/s / Testing

## Passing the arguments using the command line:

```
C:\Users\franc\Desktop\AUTPROJECTS\COMP500\PORTFOLIO B\Exercises\Lab 10 VSS\Debug>repeater --help
This program repeats a word a specified number of times.

To use this program:

  exercise.exe n word

Where n is the number of times to repeat the text 'word'
C:\Users\franc\Desktop\AUTPROJECTS\COMP500\PORTFOLIO B\Exercises\Lab 10 VSS\Debug>repeater 10 CATS
CATS, CATS, CATS, CATS, CATS, CATS, CATS, CATS, CATS, CATS
C:\Users\franc\Desktop\AUTPROJECTS\COMP500\PORTFOLIO B\Exercises\Lab 10 VSS\Debug>repeater 5 DOGS
DOGS, DOGS, DOGS, DOGS, DOGS
C:\Users\franc\Desktop\AUTPROJECTS\COMP500\PORTFOLIO B\Exercises\Lab 10 VSS\Debug>repeater 10 /\
/, /, /, /, /, /, /, /, /, /
```

## Lessons learned

- I learnt how to pass arguments into `main` using command line arguments!
  - I used `stdlib`'s `atoi` to convert a c-string into an `int`, and `string.h`'s `strcmp` to compare two strings

## 10x09b - Typedef struct

6/06/2022- Monday, week 13

### Program aim

Given the following source code, edit it so that it uses a **typedef** with the **struct** declaration.

### Implementation

Given Source Code	Changed Source Code
<pre> 1  #define _CRT_SECURE_NO_WARNINGS 2  #include &lt;stdio.h&gt; 3 4  struct Banking_Account 5  { 6      char name[32]; 7      int pin; 8      float balance; 9  }; 10 11 void print_banking_details(struct Banking_Account account); 12 13 int main(void) 14 { 15     struct Banking_Account accounts[3]; 16 17     sprintf(accounts[0].name, "Steffan Hooper"); 18     accounts[0].pin = 7373; 19     accounts[0].balance = 1250.0f; 20 21     sprintf(accounts[1].name, "Pascal Cross"); 22     accounts[1].pin = 1234; 23     accounts[1].balance = 2150.0f; 24 25     sprintf(accounts[2].name, "Jade Abbott"); 26     accounts[2].pin = 7777; 27     accounts[2].balance = 2250.0f; 28 29     print_banking_details(accounts[2]); 30 31     print_banking_details(accounts[1]); 32 33     print_banking_details(accounts[0]); 34 35     return 0; 36 } 37 38 void print_banking_details(struct Banking_Account account) 39 { 40     printf("Account Name: %s\n", account.name); 41     printf("Account Balance: %.2f\n", account.balance); 42     printf("Account Pin: %d\n", account.pin); 43     printf("\n"); 44 }</pre>	<pre> 1  #define _CRT_SECURE_NO_WARNINGS 2  #include &lt;stdio.h&gt; 3 4  // Banking_Account is a new type 5  typedef struct tag_Banking_Account 6  { 7      char name[32]; 8      int pin; 9      float balance; 10 } Banking_Account; 11 12 void print_banking_details(Banking_Account account); // removed "struct" 13 14 int main(void) 15 { 16     Banking_Account accounts[3]; // removed "struct" 17 18     sprintf(accounts[0].name, "Steffan Hooper"); 19     accounts[0].pin = 7373; 20     accounts[0].balance = 1250.0f; 21 22     sprintf(accounts[1].name, "Pascal Cross"); 23     accounts[1].pin = 1234; 24     accounts[1].balance = 2150.0f; 25 26     sprintf(accounts[2].name, "Jade Abbott"); 27     accounts[2].pin = 7777; 28     accounts[2].balance = 2250.0f; 29 30     print_banking_details(accounts[2]); 31 32     print_banking_details(accounts[1]); 33 34     print_banking_details(accounts[0]); 35 36     return 0; 37 } 38 39 void print_banking_details(Banking_Account account) // removed "struct" 40 { 41     printf("Account Name: %s\n", account.name); 42     printf("Account Balance: %.2f\n", account.balance); 43     printf("Account Pin: %d\n", account.pin); 44     printf("\n"); 45 }</pre>

### Output (Output is the same for both source code)

```

Account Name: Jade Abbott
Account Balance: $2250.00
Account Pin: 7777

Account Name: Pascal Cross
Account Balance: $2150.00
Account Pin: 1234

Account Name: Steffan Hooper
Account Balance: $1250.00
Account Pin: 7373

```

### Lessons learned

- I learnt how to implement a type for a structure using **typedef**...
- When declaring a structure type, start with “**typedef**”, the type, and the “**tag**” name. Declare the structure members, and before the closed bracket’s semicolon ( **;** ), write the NAME of the type (with the same naming convention as a structure, which is using Capital Letters)

## 11x02b - QSort Floats

6/06/2022- Monday, week 13

### Program aim

Write a program that sorts user-inputted float values. After, allow the user to search for a float value with a key.

### Plan

1. Ask the user how many floating-point values they want to store.
2. Allow the user to input the floating-point values. Use a heap to store the array of **float** values.
3. Once all floating-point values have been input by the user, sort the array using **qsort**.
4. Print the array elements, including their index, element address, and contents.
5. Ask the user for a search key. Using the search key and the **bsearch** function, check if the key is present in the array.

### Implementation

#### Source Code

```

2  #include <stdio.h>
3  #include <stdlib.h>
4
5  // compare function used with qsort
6  int compare(const void* a, const void* b)
7  {
8      return (*(float*)a - *(float*)b);
9  }
10
11 int main(void)
12 {
13     int length = 0;
14     float float_input = 0.0f;
15     float* p_heap_sort_array = 0;
16     float search_key = 0.0f;
17     int* item = 0;
18
19     // ask user for length
20     printf("How many? ");
21     scanf("%d", &length);
22     printf("\n");
23
24     // malloc memory according to user inputted length
25     p_heap_sort_array = malloc(sizeof(float) * length);
26
27     // take user input and put into array
28     for (int i = 0; i < length; i++)
29     {
30         printf("[%d] ", i);
31         scanf("%f", &float_input);
32
33         p_heap_sort_array[i] = float_input;
34     }
35
36     qsort(p_heap_sort_array, length, sizeof(float), compare);
37
38     // print sorted array
39     printf("\nSorted: \n\n");
40     for (int i = 0; i < length; i++)
41     {
42         printf("[%d] at %p is %f \n", i, &p_heap_sort_array[i], p_heap_sort_array[i]);
43     }
44
45     // get search key
46     printf("\nEnter key? ");
47     scanf("%f", &search_key);
48
49     item = bsearch(&search_key, p_heap_sort_array, length, sizeof(int), compare);
50
51     if (item)
52     {
53         printf("%f found at %p\n", search_key, item);
54     }
55     else
56     {
57         printf("Not found!\n");
58     }
59
60     // FREE MEMORY
61     free(p_heap_sort_array);
62     p_heap_sort_array = 0;
63
64     return 0;
65 }
```

#### Output/s / Testing

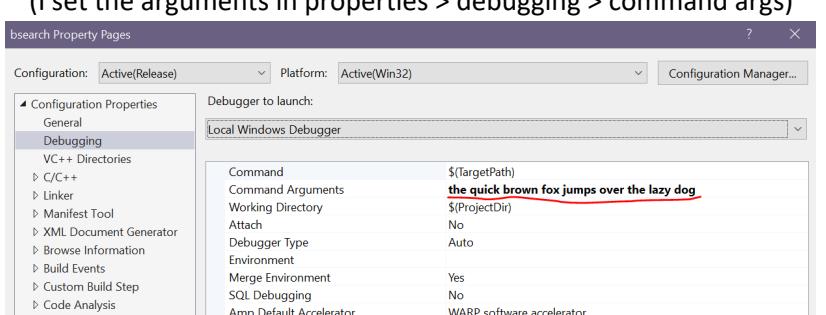
<pre>How many? 3 [0]? 4.4 [1]? 7.7 [2]? 2.42  Sorted:  [0] at 01037678 is 2.420000 [1] at 0103767C is 4.400000 [2] at 01037680 is 7.700000  Search key? 2.42 2.420000 found at 01037678</pre>	<pre>How many? 5 [0]? 2.2 [1]? 53 [2]? 23.2 [3]? 66.4 [4]? 34.4444  Sorted:  [0] at 00D504B0 is 2.200000 [1] at 00D504B4 is 23.200001 [2] at 00D504B8 is 34.444491 [3] at 00D504BC is 53.000000 [4] at 00D504C0 is 66.400002  Search key? 33 Not found!</pre>
<pre>How many? 3 [0]? 33 [1]? 22 [2]? 11  Sorted:  [0] at 00AF7678 is 11.000000 [1] at 00AF767C is 22.000000 [2] at 00AF7680 is 33.000000  Search key? a Not found!</pre>	

### Lessons learned

- I learnt how to sort an array of floats using **qsort** and to search for an element of a **float** array using **bsearch**

## 11x03d - QSort argv

6/06/2022- Monday, week 13

Program aim	
Implementation	
Source Code	Output/s / Testing
<pre> 1  #define _CRT_SECURE_NO_WARNINGS 2  #include &lt;stdio.h&gt; 3  #include &lt;string.h&gt; 4 5  // COMPARE STRINGS 6  int compare(const void* a, const void* b) 7  { 8      char** ia = (char**)a; 9      char** ib = (char**)b; 10 11     return (strcmp(*ia, *ib)); 12 } 13 14 int main(int argc, char* argv[]) 15 { 16     printf("Unsorted arguments: \n"); 17     for (int i = 0; i &lt; argc; i++) 18     { 19         printf("[%d]: %s\n", i, argv[i]); 20     } 21     printf("\n"); 22 23     // CALL QSORT &amp; COMPARE 24     qsort(argv, argc, sizeof(char*), compare); 25 26     printf("Sorted arguments: \n"); 27     for (int i = 0; i &lt; argc; i++) 28     { 29         printf("[%d]: %s\n", i, argv[i]); 30     } 31 32     return 0; 33 }</pre>	<p>(I set the arguments in properties &gt; debugging &gt; command args)</p>  <p>Unsorted arguments:</p> <pre>[0]: C:\Users\franc\Desktop\AUTPROJECTS\COMP500\PORTFOLIO B\Exercises\Lab 11 VSS\Release\bsearch.exe [1]: the [2]: quick [3]: brown [4]: fox [5]: jumps [6]: over [7]: the [8]: lazy [9]: dog</pre> <p>Sorted arguments:</p> <pre>[0]: C:\Users\franc\Desktop\AUTPROJECTS\COMP500\PORTFOLIO B\Exercises\Lab 11 VSS\Release\bsearch.exe [1]: brown [2]: dog [3]: fox [4]: jumps [5]: lazy [6]: over [7]: quick [8]: the [9]: the</pre> <p>Unsorted arguments:</p> <pre>[0]: C:\Users\franc\Desktop\AUTPROJECTS\COMP500\PORTFOLIO B\Exercises\Lab 11 VSS\Release\bsearch.exe [1]: she [2]: sells [3]: seashells [4]: by [5]: the [6]: sea [7]: shore</pre> <p>Sorted arguments:</p> <pre>[0]: C:\Users\franc\Desktop\AUTPROJECTS\COMP500\PORTFOLIO B\Exercises\Lab 11 VSS\Release\bsearch.exe [1]: by [2]: sea [3]: seashells [4]: sells [5]: she [6]: shore [7]: the</pre>

### Lessons learned

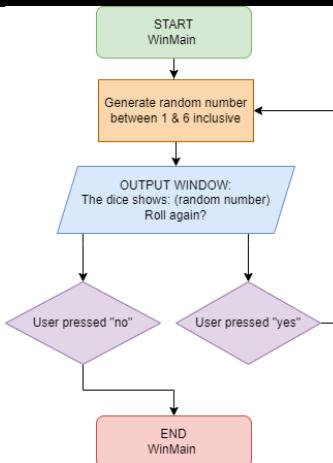
- I learnt how to use **qsort** with **main** command line arguments...

**12x05e - GUI Dice Roller**

6/06/2022- Monday, week 13

**Program aim**Design and implement a program that rolls a six-sided dice for the user using the Windows **MessageBox** functionality.

When the user clicks “yes”, roll the dice again. When the user clicks “no”, the program must exit.

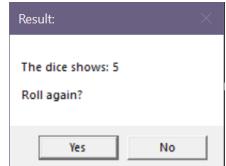
**Design (flowchart)****Implementation****Source Code**

```

1  #include <Windows.h>
2  #include <time.h>
3  #include <stdlib.h>
4
5  int WINAPI WinMain(HINSTANCE hInstance,
6  HINSTANCE hPrevInstance,
7  LPSTR lpCmdLine, int nCmdShow)
8  {
9      srand(time(0));
10     wchar_t buffer[128];
11
12     int loop = 1;
13
14     while (loop)
15     {
16         int rand_number = rand() % 6 + 1;
17
18         wsprintf(buffer, L"The dice shows: %d\n\nRoll again?", rand_number);
19         int result = MessageBox(NULL, buffer,
20             L"Result:", MB_YESNO);
21
22         if (IDYES == result)
23         {
24             continue;
25         }
26         else if (IDNO == result)
27         {
28             MessageBox(NULL, L"Bye!",
29             L"Result:", MB_OK);
30             loop = 0;
31         }
32     }
33
34
35     return 0;
36 }
```

**Output/s / Testing**

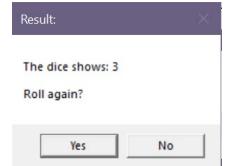
## Program launch



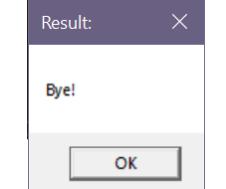
“yes” clicked



“yes” clicked



“no” clicked

**Lessons learned**

- I learnt how to use `wsprintf` to save a line of text I want in a window that contains a variable.
- I learnt how to use `wchar_t` to declare and assign a line of text to a wide c-string array

## Reflection

6/06/2022- Monday, week 13

I feel I still have much to learn and master when it comes to C programming. What are some ways I can improve personally to become a better C programmer, or a better programmer in general? What mistakes have I made?

- **Practice in my free time.** This is a way I can find passion in programming outside of university work. Perhaps I should learn how to code more complex games in C, or perhaps I should pursue learning C++, C#, or other languages through personal projects / personal research.
- **Find the joy in the pain and tedium... or embrace it.** Programming can be very difficult to understand at times, and repeated inability to comprehend programming concepts can cause burnout and self-doubt! I need to remember that programming is all about learning and learning to program is the same as learning an instrument or learning how to draw- the learning process can be ugly, but the results are satisfying and worth it!
- **Don't give up!** If there is a programming problem, there is always a solution! I just have to find it somewhere. If I take too long to solve a programming problem, don't blankly stare at it waiting for something to happen, take a break to clear your brain, come back, and try again! If it doesn't work out, just move to the next problem and come back to it later.
- **Distribute the work evenly.** Programming can take a lot of time, and cramming programming exercises into a small timeframe poses the danger of rushing work and not fully comprehending what I'm programming... I need to learn how to manage my time by doing the exercises that need to be done as soon as possible, instead of leaving it to the last date.

Although I do have much to learn, I feel that I have learnt a lot about C programming during the COMP500 course along with many new programming concepts that I have not touched upon when learning Python (in high school)- such as structures, enumerations, and pointers. I am excited to learn new languages in future courses- such as Java in programming 2- to expand my programming knowledge.

Thank you to the COMP500 teaching team for helping me learn about C programming and many programming concepts!

### APA 7th Referencing

- Arkadio. (2016). *Use of member access operator on a pointer*. Stackoverflow.  
<https://stackoverflow.com/questions/35876567/use-of-member-access-operator-on-a-pointer>
- CalculatorSoup. (n.d.). *Three Dimensional Distance Calculator*.  
<https://www.calculatorsoup.com/calculators/geometry-solids/distance-two-points.php>
- Hooper, S. (2022). *Lecture 017 (S1, 2022, Pre-recorded)* [Video]. Panopto.  
<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=6631564d-7567-4028-b646-ade20169d254>
- Hooper, S. (2022). *Lecture 018 (S1, 2022, Pre-recorded)* [Video]. Panopto.  
<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=bce21bc6-3c7b-4fc7-b09c-ade20169d2eb>
- Hooper, S. (2022). *Lecture 019 (S1, 2022, Pre-recorded)* [Video]. Panopto.  
<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=f8b6a6f3-b264-462a-9eea-ade20169d3a9>
- Hooper, S. (2022). *Lecture 020 (S1, 2022, Pre-recorded)* [Video]. Panopto.  
<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=7c3316e7-b73f-4d3d-908e-ade20169d437>
- Hooper, S. (2022). *Lecture 024 (S1, 2022, Pre-recorded)* [Video]. Panopto.  
<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=12d43f26-eaad-44d2-8b8c-ade20169d6a6>
- Hooper, S. (2022). *Lecture 036 (S1, 2022, Live recording)* [Video]. Panopto.  
<https://aut.au.panopto.com/Panopto/Pages/Viewer.aspx?id=1b3191f3-ba62-4cac-92a0-aea80176930b>
- Tutorialspoint. (n.d.). *C library function - getchar()*.  
[https://www.tutorialspoint.com/c\\_standard\\_library/c\\_function\\_getchar.htm](https://www.tutorialspoint.com/c_standard_library/c_function_getchar.htm)