

14885857

Reporting Journal #4

Assignment #1

Reporting Journal

Mateus Felipe Paludo (Paludo)

IE: 14885857

Tutor: Steffan Hooper

Paludo

Page 1 of 688

Contents

Contents	2
1 Week One	18
1.1 Lectures	18
1.2 Examples	20
1.2.1 Example 1	20
1.2.2 Example 2	21
1.2.3 Example 3	22
1.2.4 Example 4	23
1.2.5 Example 5	25
1.3 Exercises	27
1.3.1 Exercise 1	27
1.3.3 Exercise 2	28
1.3.3 Exercise 3	29
1.3.4 Exercise 4	32
1.3.5 Exercise 5	34
1.3.6 Exercise 6	36
1.3.7 Exercise 7	38
1.3.8 Exercise 8	40

2 Week Two	42
2.1 Lectures	42
2.2 Examples	43
2.2.1 Example 1	43
2.2.2 Example 2	44
2.2.3 Example 3	47
2.2.4 Example 4	49
2.2.5 Example 5	52
2.2.6 Example 6	54
2.2.7 Example 7	56
2.2.8 Example 8	57
2.2.9 Example 9	58
2.2.10 Example 10	59
2.3 Exercises	61
2.3.1 Exercise 1	61
2.3.2 Exercise 2	64
2.3.3 Exercise 3	66
2.3.4 Exercise 4	69
2.3.5 Exercise 5	74

2.3.6 Exercise 6	77
2.3.7 Exercise 7	82
2.3.8 Exercise 8	84
2.3.9 Exercise 9	88
3 Week Three	91
3.1 Lectures	91
3.2 Examples	93
3.2.1 Example 1	93
3.2.2 Example 2	95
3.2.3 Example 3	96
3.2.4 Example 4	97
3.2.5 Example 5	98
3.2.6 Example 6	100
3.2.7 Example 7	102
3.2.8 Example 8	104
3.2.9 Example 9	106
3.2.10 Example 10	108
3.3 Exercises	110
3.3.1 Exercise 1	110

3.3.2	Exercise 2	113
3.3.3	Exercise 3	118
3.3.4	Exercise 4	121
3.3.5	Exercise 5	124
3.3.6	Exercise 6	126
3.3.7	Exercise 7	139
3.3.8	Exercise 8	145
3.3.9	Exercise 9	149
4	Week Four	153
4.1	Lectures	153
4.2	Exercises	154
4.2.1	Exercise 1	154
4.2.2	Exercise 2	155
4.2.3	Exercise 3	157
4.2.4	Exercise 4	159
4.2.5	Exercise 5	161
4.2.6	Exercise 6	163
4.2.7	Exercise 7	165
4.2.8	Exercise 8	167

4.2.9	Exercise 9	169
4.2.10	Exercise 10	172
4.2.11	Exercise 11	174
4.2.12	Exercise 12	176
4.2.13	Exercise 13	178
4.2.14	Exercise 14	180
4.2.15	Exercise 15	183
4.2.16	Exercise 16	185
5	Week Five	188
5.1	Lectures	188
5.2	Examples	189
5.2.1	Example 1	189
5.2.2	Example 2	190
5.2.3	Example 3	192
5.2.4	Example 4	193
5.2.5	Example 5	195
5.2.6	Example 6	197
5.2.7	Example 7	198
5.2.8	Example 8	200

5.2.9	Example 9	202
5.2.10	Example 10	204
5.2.11	Example 11	206
5.2.12	Example 12	208
5.2.13	Example 13	210
5.2.14	Example 14	212
5.2.15	Example 15	214
5.2.16	Example 16	216
5.3	Exercises	218
5.3.1	Exercise 1	218
5.3.2	Exercise 2	222
5.3.3	Exercise 3	224
5.3.4	Exercise 4	229
5.3.5	Exercise 5	231
5.3.6	Exercise 6	233
5.3.7	Exercise 7	236
5.3.8	Exercise 8	240
5.3.9	Exercise 9	243
5.3.10	Exercise 10	245

5.3.11	Exercise 11	247
5.3.12	Exercise 12	251
5.3.13	Exercise 13	256
5.3.14	Exercise 14	260
5.3.15	Exercise 15	263
5.3.16	Exercise 16	271
6	Week Six	283
6.1	Lectures	283
6.2	Examples	285
6.2.1	Example 1	285
6.2.2	Example 2	287
6.2.3	Example 3	288
6.2.4	Example 4	289
6.2.5	Example 5	291
6.2.6	Example 6	294
6.2.7	Example 7	296
6.2.8	Example 8	298
6.3	Exercises	300
6.3.1	Exercise 1	300

6.3.2	Exercise 2	302
6.3.3	Exercise 3	304
6.3.4	Exercise 4	306
6.3.5	Exercise 5	309
6.3.6	Exercise 6	312
6.3.7	Exercise 7	314
6.3.8	Exercise 8	316
6.3.9	Exercise 9	319
6.3.10	Exercise 10	321
6.3.11	Exercise 11	323
6.3.12	Exercise 12	331
6.3.13	Exercise 13	339
6.3.14	Exercise 14	341
6.3.15	Exercise 15	343
6.3.16	Exercise 16	345
6.3.17	Exercise 17	347
6.3.18	Exercise 18	349
6.3.19	Exercise 19	351
6.3.20	Exercise 20	353

6.3.21	Exercise 21	355
6.3.22	Exercise 22	357
6.3.23	Exercise 23	360
6.3.24	Exercise 24	364
7	Week Seven	367
7.1	Lectures	367
7.2	Examples	368
7.2.1	Example 1	368
7.2.2	Example 2	370
7.2.3	Example 3	372
7.2.4	Example 4	374
7.2.5	Example 5	376
7.2.6	Example 6	379
7.2.7	Example 7	382
7.2.8	Example 8	384
7.3	Exercises	386
7.3.1	Exercise 1	386
7.3.2	Exercise 2	388
7.3.3	Exercise 3	390

7.3.4	Exercise 4	392
7.3.5	Exercise 5	395
7.3.6	Exercise 6	397
7.3.7	Exercise 7	400
7.3.8	Exercise 8	402
7.3.9	Exercise 9	405
7.3.10	Exercise 10	408
7.3.11	Exercise 11	410
7.3.12	Exercise 12	412
7.3.13	Exercise 13	414
7.3.14	Exercise 14	416
7.3.15	Exercise 15	418
7.3.16	Exercise 16	420
7.3.17	Exercise 17	423
7.3.18	Exercise 18	427
7.3.19	Exercise 19	430
7.3.20	Exercise 20	434
7.3.21	Exercise 21	448
7.3.22	Exercise 22	453

8 Eight	457
8.1 Lectures	457
8.2 Examples	459
8.2.1 Example 1	459
8.2.2 Example 2	461
8.2.3 Example 3	463
8.2.4 Example 4	465
8.2.5 Example 5	467
8.2.6 Example 6	468
8.2.7 Example 7	470
8.2.8 Example 8	472
8.2.9 Example 9	474
8.2.10 Example 10	476
8.2.11 Example 11	478
8.2.12 Example 12	480
8.2.13 Example 13	482
8.3 Exercises	484
8.3.1 Exercise 1	484
8.3.2 Exercise 2	488

8.3.3	Exercise 3	490
8.3.4	Exercise 4	492
8.3.5	Exercise 5	496
8.3.6	Exercise 6	500
8.3.7	Exercise 7	504
8.3.8	Exercise 8	524
8.3.9	Exercise 9	537
9	Nine	554
9.1	Lectures	554
9.2	Examples	555
9.2.1	Example 1	555
9.2.2	Example 2	557
9.2.3	Example 3	558
9.2.4	Example 4	559
9.2.5	Example 5	561
9.3	Exercises	563
9.3.1	Exercise 1	563
9.3.2	Exercise 2	566
9.3.3	Exercise 3	569

9.3.4	Exercise 4	572
9.3.5	Exercise 5	575
9.3.6	Exercise 6	580
9.3.7	Exercise 7	584
9.3.8	Exercise 8	586
9.3.9	Exercise 9	588
9.3.10	Exercise 10	590
9.3.11	Exercise 11	592
9.3.12	Exercise 12	594
9.3.13	Exercise 13	596
9.3.14	Exercise 14	600
10	Ten	601
10.1	Lectures	601
10.2	Examples	602
10.2.1	Example 1	602
10.2.2	Example 2	603
10.2.3	Example 3	605
10.2.4	Example 4	607
10.3	Exercises	609

10.3.1	Exercise 1	609
10.3.2	Exercise 2	611
10.3.3	Exercise 3	613
10.3.4	Exercise 4	616
10.3.5	Exercise 5	620
10.3.6	Exercise 6	624
10.3.7	Exercise 7	627
10.3.8	Exercise 8	630
10.3.9	Exercise 9	633
11	Eleven	635
11.1	Lectures	635
11.2	Examples	636
11.2.1	Example 1	636
11.2.2	Example 2	638
11.2.3	Example 3	640
11.3	Exercises	641
11.3.1	Exercise 1	641
11.3.2	Exercise 2	642
11.3.3	Exercise 3	643

11.3.4	Exercise 4	644
11.3.5	Exercise 5	645
11.3.6	Exercise 6	646
11.3.7	Exercise 7	647
11.3.8	Exercise 8	648
11.3.9	Exercise 9	649
11.3.10	Exercise 10	650
11.3.11	Exercise 11	651
11.3.12	Exercise 12	652
11.3.13	Exercise 13	653
11.3.14	Exercise 14	654
11.3.15	Exercise 15	656
11.3.16	Exercise 16	657
11.3.17	Exercise 17	658
11.3.18	Exercise 18	659
11.3.19	Exercise 19	661
12	Twelve	663
12.1	Lectures	663
12.2	Examples	664

12.2.1 Example 1	664
12.2.2 Example 2	667
12.2.3 Example 3	668
12.2.4 Example 4	671
12.3 Exercises	674
12.3.1 Exercise 1	674
12.3.2 Exercise 2	677
13 Hours of study	681
References	686

1 Week One

1.1 Lectures

In the first week I learned how to study C programming in a successful way, by thinking logically, understanding how does the program works, understand every line of my program, thinking “out of the box”, which means to think in a different and innovative way. To succeed in this paper and to be a good programmer as well I might practice a lot, try to break the boundaries of my knowledge and search for new challenges. Sharing information, helping and asking help to other students, teachers and friends is very helpful as well, other people may think of another way of solving the same problem, and, if it is not helpful now, it may be in the future.

I have used Microsoft Visual Studio for the first time, the program is very helpful, has a beautiful interface, which allows the user to organize his projects in an amazing way. The Microsoft Visual Studio’s debugger is exceptional, with it I was able to stop my program on the line in which I was having a problem and try to understand what was the problem.

In the class we were advised that programmers, in order to achieve successfullness, should edit, by writing plain text source code; compile, converting the source code into machine code; run, executing the program and analysing it; and test it, checking if the program does what it was supposed to do. But I miss some important steps on this cycle, the first one is: debug, the programmer shall look at the code carefully and understand how the machine interprets it in order to fix eventual errors and problems. The second step that is important in a programmer’s routine is to improve his knowledge, by reading, talking to people, and keep the previous know-how updated. User interface is the third important step that should be added

to the list, the programmer may think of how other people may use the program and what problems they may have and try to overcome these possible problems, after this it would be very important to ask other people to use the program.

While programming, it is not a good idea to write massive lines of code before compiling it and running the program, the best way is to check if there is any problem or glitch in your program constantly while designing and building it.

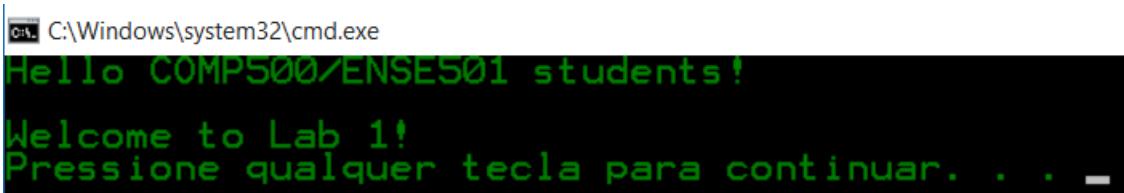
While using Microsoft Visual Studio I create a main solution for the week's laboratory, inside it I create projects for each exercise, and an object in which I write the code, this keeps my programs well organized and easy to manage.

Before writing a code, it is very helpful to think what it does, write down in words how the program will work, this is called pseudo code. Another very helpful way of solve problems is to design flowcharts, they help the programmer think logically and understand what the problem really is.

1.2 Examples

1.2.1 Example 1

The first example of the week one's laboratory prints a greeting message for the class.



A screenshot of a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
Hello COMP500/ENSE501 students!
Welcome to Lab 1!
Pressione qualquer tecla para continuar. . . -

Figure 01: Simple printf.

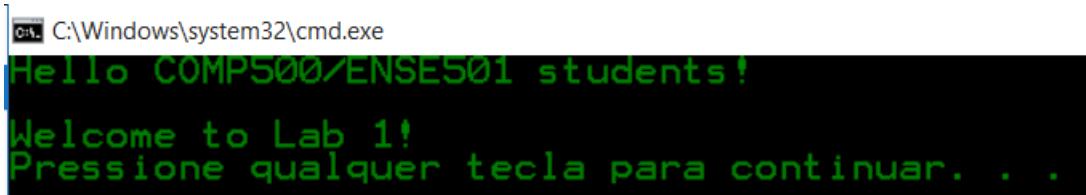
The example's code is this:

```
#include <stdio.h>

int main()
{
    printf("Hello COMP500/ENSE501 students!\n\n");
    printf("Welcome to Lab 1!\n");
    return 0;
}
```

1.2.2 Example 2

The second example was, as well, a simple `printf` that displays the following prompt:



A screenshot of a Windows command prompt window titled 'cmd C:\Windows\system32\cmd.exe'. The window contains the following text:
Hello COMP500/ENSE501 students!
Welcome to Lab 1!
Pressione qualquer tecla para continuar. . .

Figure 02: Another Simple `printf`.

Even if the display is the same, the code is different, which means that sometimes you can write different codes and get the same outputs. The following code is the code written.

```
#include <stdio.h>

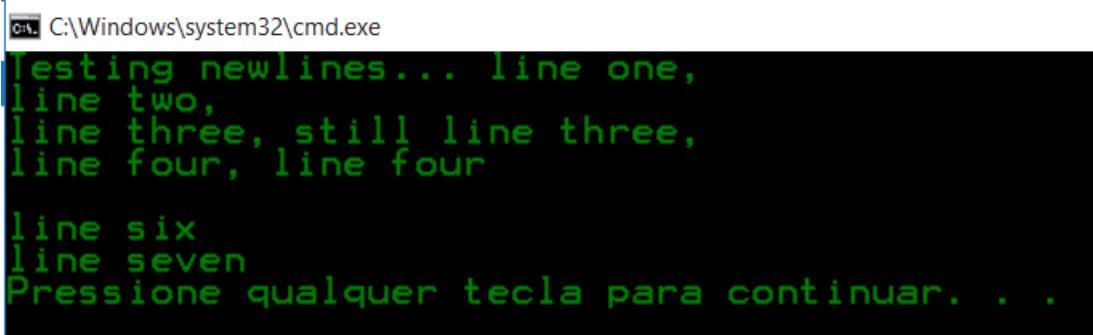
int main()
{
    printf("Hello ");
    printf("COMP500");
    printf("/");
    printf("ENSE501 ");
    printf("students!");
    printf("\n\n");

    printf("Welcome to Lab 1!\n");

    return 0;
}
```

1.2.3 Example 3

The third example was a code composed by a sequence of `printf` calls that displays:



```
C:\Windows\system32\cmd.exe
Testing newlines... line one,
line two,
line three, still line three,
line four, line four

line six
line seven
Pressione qualquer tecla para continuar. . .
```

Figure 03: Newlines.

The code's objective was to illustrate and clarify how to use `\n` to open new lines in C coding. The following code was this third example:

```
#include <stdio.h>

int main()
{
    printf("Testing newlines... ");

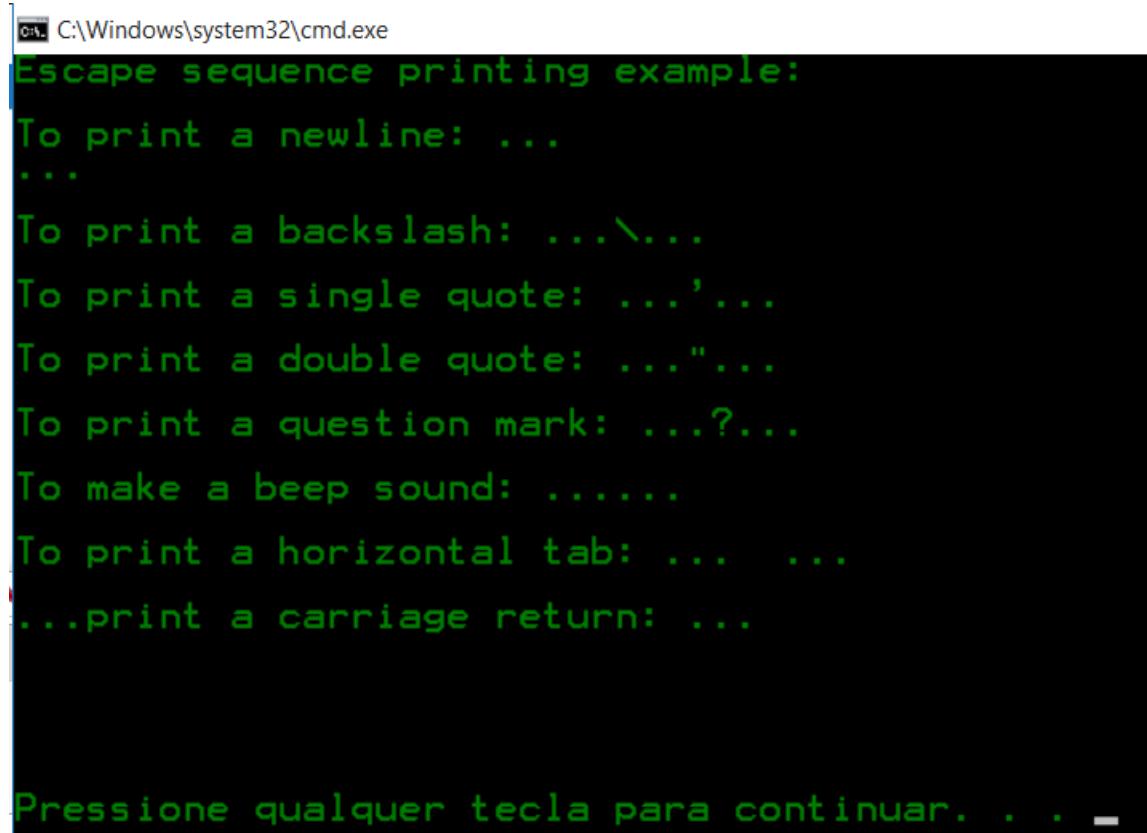
    printf("line one,\nline two,\nline three, ");
    printf("still line three,\n");

    printf("line four, line four\n\n");
    printf("line six\n");
    printf("line seven\n");

    return 0;
}
```

1.2.4 Example 4

The fourth example makes more sense when looking at the code than when looking at the display:



A screenshot of a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
Escape sequence printing example:
To print a newline: ...
...
To print a backslash: ...\\...
To print a single quote: ...'...
To print a double quote: ..."..."
To print a question mark: ...?...
To make a beep sound:
To print a horizontal tab: ... \t ...
...print a carriage return: ...

Pressione qualquer tecla para continuar. . . -

Figure 04: Escape Sequences.

The code gives an example of various printf functions that are very useful for C programmers. This was the code written:

```
#include <stdio.h>

int main()
{
    printf("Escape sequence printing example:\n\n");
    printf("To print a newline: ");
    printf("\\n");
    printf("To print a backslash: ");
    printf("\\\\");
    printf("To print a single quote: ");
    printf("\\'");
    printf("To print a double quote: ");
    printf("\\\"");
    printf("To print a question mark: ");
    printf("\\?");
    printf("To make a beep sound: ");
    printf("\\a");
    printf("To print a horizontal tab: ");
    printf("\\t");
    printf("To print a carriage return: ");
    printf("\\r");
}
```

```
printf("...\\n...");  
printf("\\n\\n");  
printf("To print a backslash: ");  
printf("...\\\\...");  
printf("\\n\\n");  
printf("To print a single quote: ");  
printf("...\\'...");  
printf("\\n\\n");  
printf("To print a double quote: ");  
printf("...\\\"...");  
printf("\\n\\n");  
printf("To print a question mark: ");  
printf("...\\?...");  
printf("\\n\\n");  
printf("To make a beep sound: ");  
printf("...\\a...");  
printf("\\n\\n");  
printf("To print a horizontal tab: ");  
printf("...\\t...");  
printf("\\n\\n");  
printf("To print a carriage return: ");  
printf("...\\r...");  
printf("\\n\\n");  
printf("\\n\\n\\n\\n");  
  
return 0;  
}
```

1.2.5 Example 5

The fifth and last week one's laboratory example returns a shaped diamond and two squares using characters.



Figure 05: Shape Printing.

This code introduces to the student another possibility of the `printf` calls.

The code was a little bit harder to write and it took a lot of tests to achieve the final formating, here follows the final code:

```
#include <stdio.h>

int main()
```

```
{  
    printf("----+\n|     |\n| 1 | \n|     |\n----+\n");  
  
    printf("\n\n");  
  
    printf("   /\\ \\n");  
    printf(" /  \\ \\n");  
    printf(" /    \\ \\n");  
    printf("/    2  \\ \\n");  
    printf("\\\\      /\\n");  
    printf("\\\\    /\\n");  
    printf("\\\\  /\\n");  
    printf("\\\\/\\n");  
  
    printf("\n\n");  
  
    printf("----+\n|     |\n| 3 | \n|     |\n----+\n");  
  
    printf("\n\n");  
    return 0;  
}
```

1.3 Exercises

1.3.1 Exercise 1

The first laboratory exercise was to use a sequence of `printf` calls to output the example's text:

```
Hello
    Programming
        Students
```

Figure 06: Text pattern to be followed.

This is the code I wrote:

```
#include <stdio.h> //standard input and output library

int main() //this starts the program
{
    printf("Hello \n");//text displayed
    printf("    Programming \n");// \n starts a new line
    printf("        Students \n");
    return 0; //ends the program, returning 0}
```

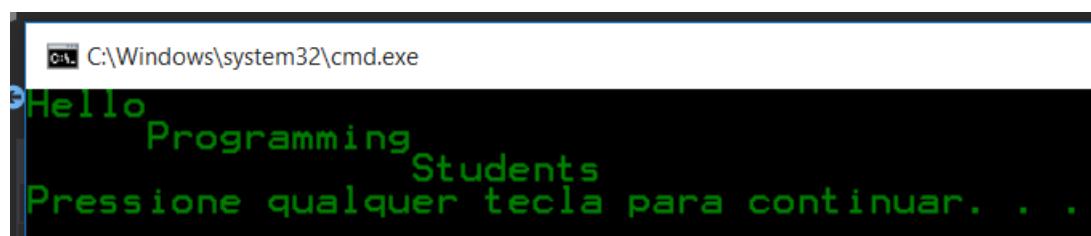


Figure 07: Text Welcome.

The first time I wrote it in one single line, using Tab and \n, but it wasn't looking like the exercise's pattern. Other problem that I found it was that I was missing a semicolon on the 7th line, the program was running, but it wasn't looking the same as the exercise.

1.3.3 Exercise 2

I found the second exercise easier than the first one, I had no problems while doing it, and accomplished the tasks without errors.

As in the first exercise, it should display a pattern.

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

Figure 08: Text pattern to be followed.

I wrote the following code:

```
#include <stdio.h>  
  
int main()  
{  
    printf("# \n");  
    printf("## \n");  
    printf("### \n");  
    printf("#### \n");  
    printf("##### \n");  
    return 0;  
}
```

C:\Windows\system32\cmd.exe

```
#  
##  
###  
####  
#####  
Pressione qualquer tecla para continuar. . .
```

Figure 09: Right Angle Triangle.

1.3.3 Exercise 3

The third exercise consisted in writing a program to display the initials of my name using *, with four spaces between each letter, as in the example:

```
*****      *****      **      **
**          **      **      **
**          **      **      **
**          **      **      **
*****      **          **      *****
      **      **      **      **
      **      **      **      **
      **      **      **      **
**          **      **      **
*****      *****      **      **
```

Figure 10: Exercise's example.

But, by assumption, I just wrote a code to exhibit the same letters and did not count the number of spaces between each letter, assuming that it was just a Tab.

```
#include <stdio.h>

int main()
{
    printf("*****      *****      **      **\n");
    printf("**          **      **      **\n");
    printf("**          **      **      **\n");
    printf("**          **      **      **\n");
    printf("*****      **          **      *****\n");
    printf("      **      **      **      **\n");
    printf("*****      *****      **      **\n");
    return 0;
}
```



Figure 11: SDH letters.

Even after going home and looking at the programs again, I did not realize the mistake, I just noticed when the teacher gave the advise at the classroom, in order to fix the error, I wrote this code:

```
#include <stdio.h>

int main()
{
    printf("*****\n");
    printf("*****\n");
    printf("** ** *\n");
    return 0;
}
```



Figure 12: Three Big Letters.

After making this mistake, I will pay more attention to the exercises' instructions and try to do not make more errors like this.

1.3.4 Exercise 4

The objective of the fourth exercise was to, using a sequence of `printf` calls, display a big X in a box, as in the example:

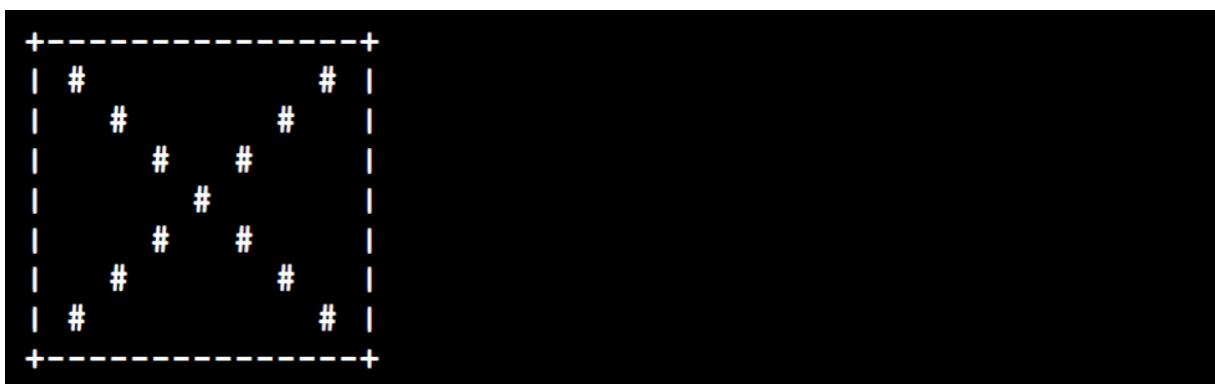


Figure 13: Exercise's pattern.

To complete the exercise I wrote this code:

```
#include <stdio.h>

int main()
{
    printf("-----+\n");
    printf("| #           # |\n");
    printf("|   #       # |\n");
    printf("|       #     |\n");
    printf("|       #     |\n");
    printf("|   #           # |\n");
    printf("| #           # |\n");
    printf("-----+\n");
    return 0;
}
```



Figure 14: X in a box.

I had no problems writing this code. But I realized that the first and last lines were the same, and in the other lines, just the position of the cardinal symbol changed, this allowed me to write the code faster than if I wrote every single line of it from scratch.

1.3.5 Exercise 5

In the fifth exercise, the objective was to write a code with a sequence of `printf` calls to display two mirrored triangles, as shown in the example:

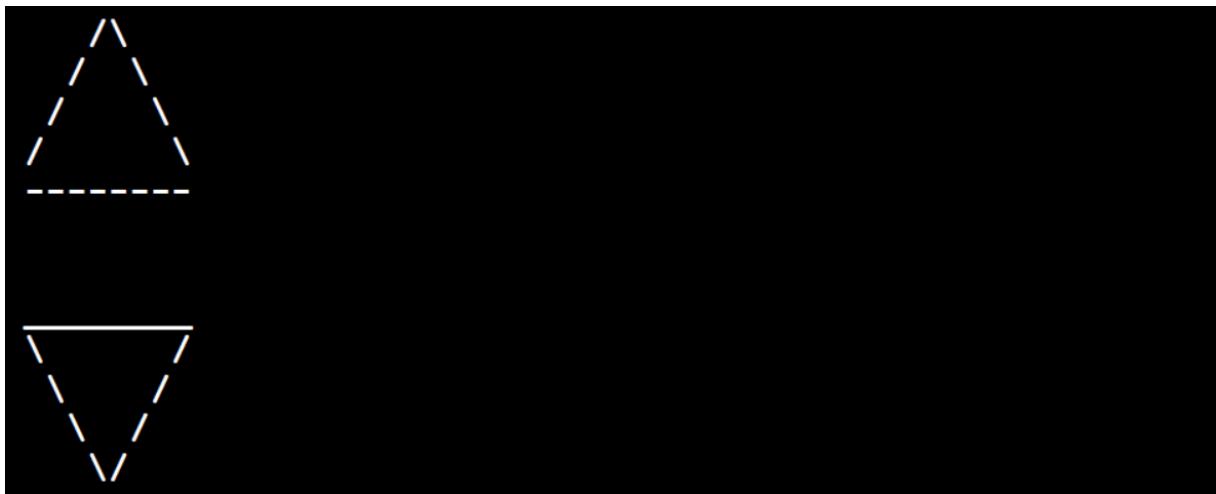


Figure 15: Mirrored Trigangles' example.

To accomplish this task, I wrote the following code:

```
#include <stdio.h>

int main()
{
    printf("      /%c \n", 92); //92 is the decimal in the ASCII
table for \
    printf("      / %c \n", 92); //%c calls a special character to be
determined
    printf("      /     %c \n", 92);
    printf("      /       %c \n", 92);
    printf("      ----- \n");
    printf("\n");
    printf("\n");
    printf("\n");
    printf("\n");
    printf("\n");
    printf("      _____ \n");
```

```
    printf(" %c      / \n", 92);
    printf(" %c      / \n", 92);
    printf("   %c  / \n", 92);
    printf("     %c/ \n", 92);
    return 0;
}
```

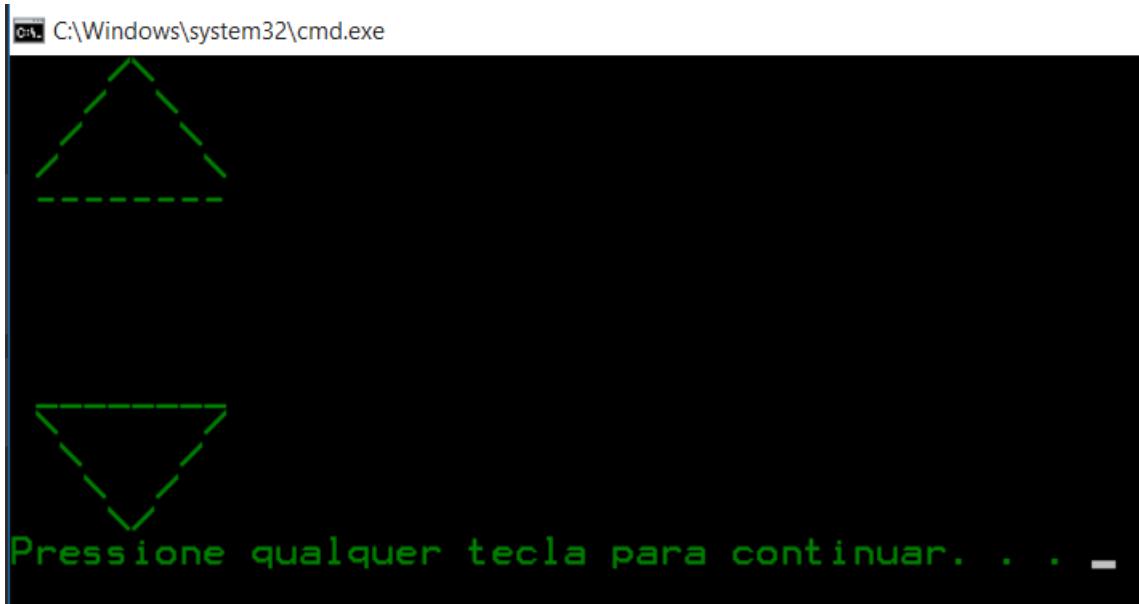


Figure 16: Mirrored Triangles.

This exercise took me longer to finish. In this code I used the ASCII table and the **%c** for the first time, this helped me to understand how to use special characters in my **printf** calls.

1.3.6 Exercise 6

The exercise 6's objective was to write a program to display a sailboat as in the example, and give it a different name.

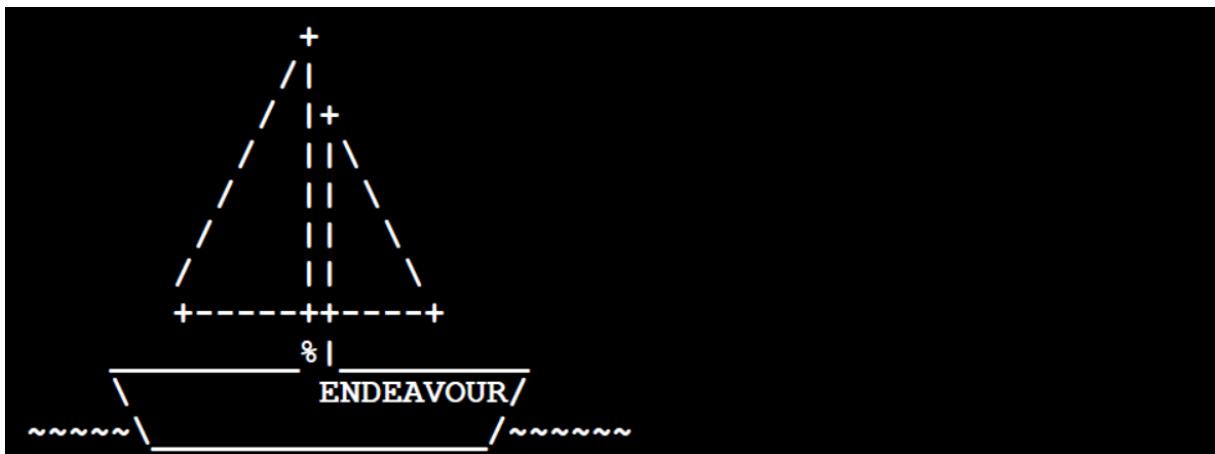


Figure 17: Exercise's example.

The code to display my boat is this:

```
#include <stdio.h>

int main()
{
    printf("          + \n", 92);
    printf("         /| \n", 92);
    printf("        / |+ \n", 92);
    printf("       /  ||%c \n", 92);
    printf("      /   || %c \n", 92);
    printf("     /    ||  %c \n", 92);
    printf("    /     ||   %c \n", 92);
    printf("   +-----+ \n");
    printf("   %c|_____\n", 37);
    printf("   %c THE DRAGONBORN/ \n", 92);
    printf("%c%c%c%c%c%c_____/%c%c%c%c%c \n", 126, 126,
126, 126, 126, 126, 126, 126, 126);
    return 0;
}
```



Figure 18: The Dragonborn boat's code.

To do the “~” symbol on the keyboard I also used a %c because the laboratory’s keyboard layout it is different from my keyboard and I did not find the symbol on the keyboard. Another character that I had to use %c to display was the percentage symbol, another option to display a % on the code is to write two percentage symbols together (%%). As in previous exercises, the boat’s drawing follows a pattern, which makes easier to write it.

1.3.7 Exercise 7

The seventh exercise's objective was to write a code to display a cityscape as in the example:

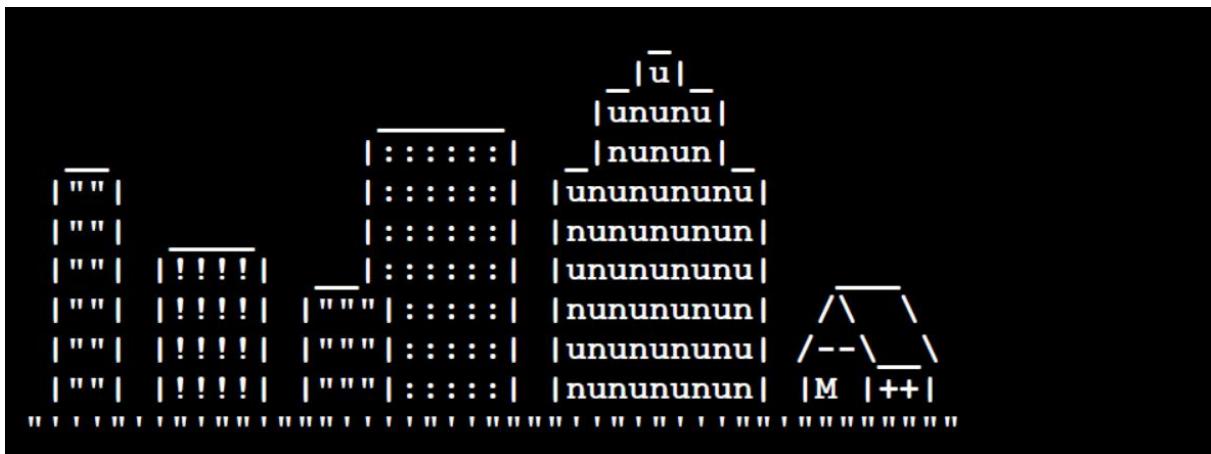


Figure 19: Cityscape’s example.

To do so, I wrote the following code:

```
#include <stdio.h>

int main()
{
    printf("                                     \n");
    printf("                                     _|u|_ \n");
    printf("                                     |ununu| \n");
    printf("                                     |:::::| _|nunun|_ \n");
    printf(" |%c%c|         |:::::| |ununununu| \n", 249, 249);
    printf(" |%c%c|   _____|:::::| |nunununun| \n", 249, 249);
    printf(" |%c%c| |!!!!|  _|:::::| |ununununu|   _ \n", 249,
249);
    printf(" |%c%c| |!!!!| |%c%c%c|:::::| |nunununun| /%c %c \n",
249, 249, 249, 249, 92, 92);
    printf(" |%c%c| |!!!!| |%c%c%c|:::::| |ununununu| /--%c__%c
\n", 249, 249, 249, 249, 249, 92, 92);
    printf(" |%c%c| |!!!!| |%c%c%c|:::::| |nunununun| |M |++| \n",
249, 249, 249, 249, 249);
```

```
    printf("%c''%c''%c'%c%c'%'c%c%c'%'c'%'c''%c%c'%
c%c%c%c%c%c \n", 249, 249, 249, 249, 249, 249, 249, 249, 249,
249, 249, 249, 249, 249, 249, 249, 249, 249, 249, 249, 249,
249, 249);
    return 0;
}
```

To write the code I had to use a lot of %c because it had a large number of special characters, and I did not know which characters can be written normally and which can not, which led me to rewrite some lines of code. Except for the ground line, it was really easy to write the code, because it follows a pattern that is really easy to change.

1.3.8 Exercise 8

The eighth and last week one's exercise was to trace goals for this semester and how to succeed on it. Even the laboratory sheet did not ask for, I decided to write my goals in a C program.

```
#include <stdio.h>

int main()
{
    printf("\n");
    printf("Hello Steffan \n \n");
    printf("My name is: ");
    printf("Mateus Felipe ");
    printf("Paludo \n \n");
    printf("My stream is: ");
    printf("ENSE501/12 \n \n");
    printf("My three key goals for the course are: \n");
    printf(" - Improve my programming abilities to help my team
programming our robots; \n");
    printf(" - Learn other programming applications in
engineering; \n");
    printf(" - Have fun solving the proposed problems; \n \n");
    printf("To ensure my success this semester I will work hard and
think logically to find a solution and surpass the obstacles. \n
\n");
    printf("Thanks \n");
    printf("Matt \n \n");
    return 0;
}
```

```
C:\Windows\system32\cmd.exe
Hello Steffan
My name is: Mateus Felipe Paludo
My stream is: ENSE501/12
My three key goals for the course are:
- Improve my programming abilities to help my team programming our robots;
- Learn other programming applications in engineering;
- Have fun solving the proposed problems;
To ensure my success this semester I will work hard and think logically to find a solution and surpass the obstacles.
Thanks
Matt
Pressione qualquer tecla para continuar. . . -
```

Figure 20: My goals' program.

To trace my goals I read the paper's description again and thought what are my main objectives in this course. As I had to e-mail it to my teacher, this was my answer to the exercise's question:

"My name is: Mateus Felipe Paludo

My stream is: ENSE501/12

My three key goals for the course are:

- Improve my programming abilities to help my team programming our robots;
- Learn other programming applications in engineering;
- Have fun solving the proposed problems;

To ensure my success this semester I will work hard and think logically to find a solution and surpass the obstacles.

Thanks

Matt"

Paludo

2 Week Two

2.1 Lectures

At the second week lectures we learned the definition of variables and how does the computer work with them. The watch window of the Visual Studio Debugger was also introduced to the students, it is extremely efficient, it allows the programmer to see a specific variable changing while debugging the code. Arithmetic was another topic of the lectures, some basic mathematical operations and how to use format specifiers to print them were introduced.

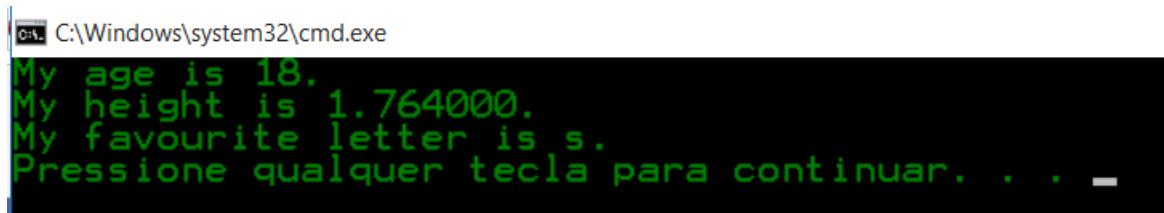
On the second week I also learned how to use chars and the ASCII table with decimal and hexadecimal values of the respective characters. An useful command to print the address of a variable was also mentioned during the classes. The variable truncating method was also presented.

The last lecture of the second week presented some coding standards that the students are supposed to follow in their programs, these coding standards make it easier for the programmer and other people to look and understand what the program exactly does. Another useful technique to help people making the code easier to understand is commenting. Commenting codes can also be used to debug a software.

2.2 Examples

2.2.1 Example 1

The first example of the week 2 lab consisted of a code with three variables and a sequence of `printf` calls to show the value of these variables.



A screenshot of a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
My age is 18.
My height is 1.764000.
My favourite letter is s.
Pressione qualquer tecla para continuar. . . -

Figure 21: Simple printf with Variables.

The code has a int variable to store the age, a float to store the height and a char variable. The code written is this:

```
#include <stdio.h>

int main()
{
    int age = 18;
    float height = 1.764f; //why the f?
    char favourite = 's';
    printf("My age is %d. \n", age);
    printf("My height is %f.\n", height);
    printf("My favourite letter is %c. \n", favourite);

    return 0;
}
```

2.2.2 Example 2

The second example's output is this:



```
C:\Windows\system32\cmd.exe
The value of a is 4.
The value of b is 11.
Pressione qualquer tecla para continuar. . .
```

Figure 22: Addition with Variables.

The program initiates two variables, assigns a value to one of these variables, prints the value of it, sums the first variable with another value and puts the result in the second variable, and then print it.

```
#include <stdio.h>

int main()
{
    int a;
    int b;

    a = 4;
    printf("The value of a is %d.\n", a);

    b = a + 7;
    printf("The value of b is %d.\n", b);

    return 0;
}
```

We can see the variables changing debugging the program and looking at the Watch Window.

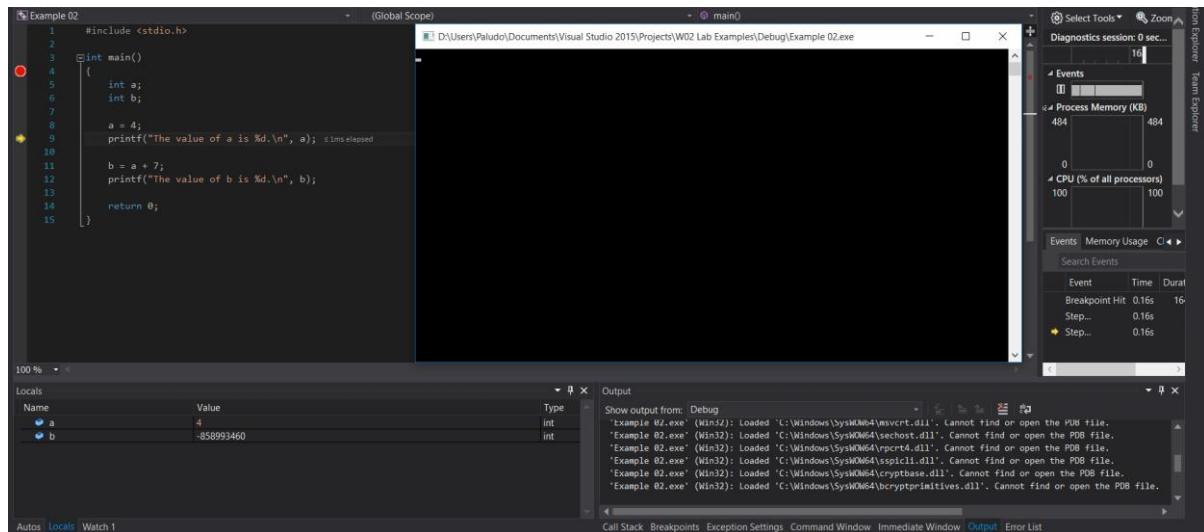


Figure 23: The int *a* receives the value 4.

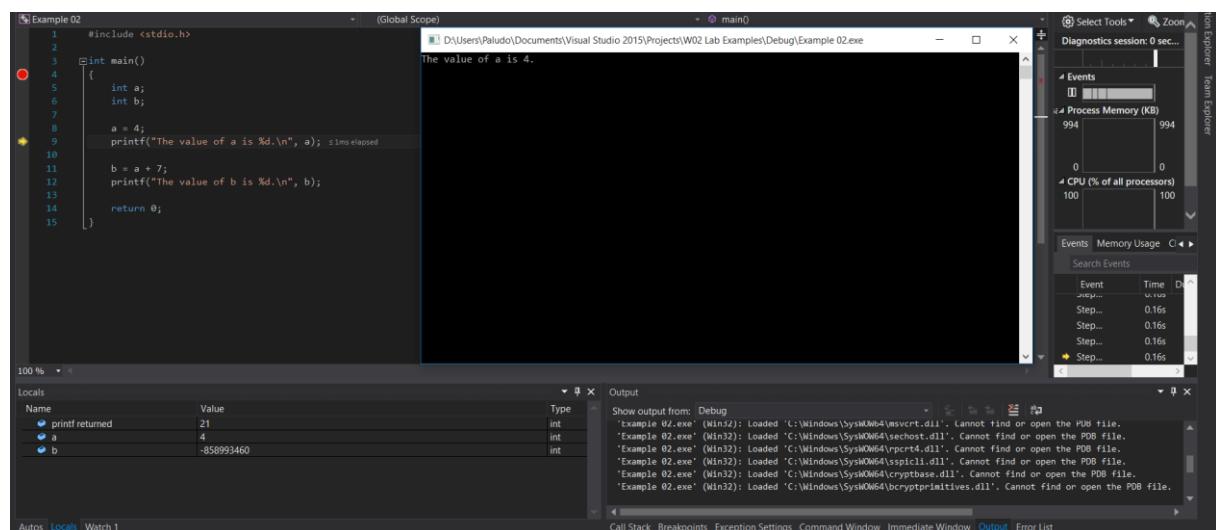


Figure 24: The program outputs the value of *a*.

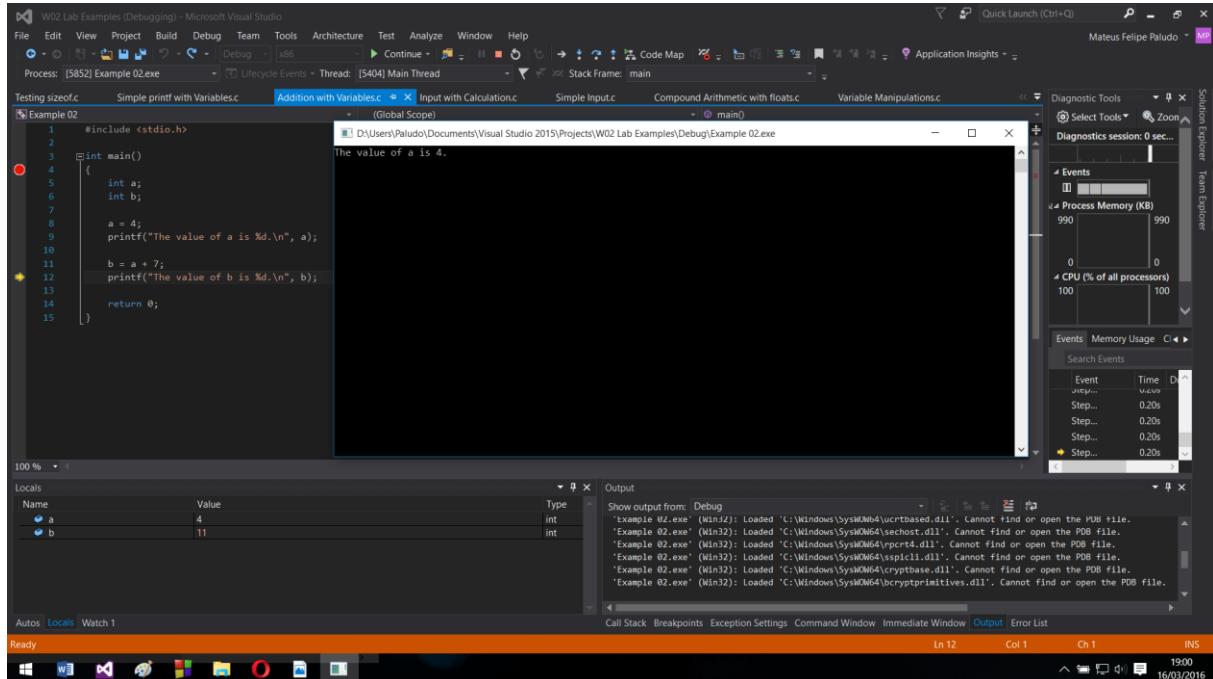


Figure 25: The program sums *a* and 7 and assigns the result to *b*.

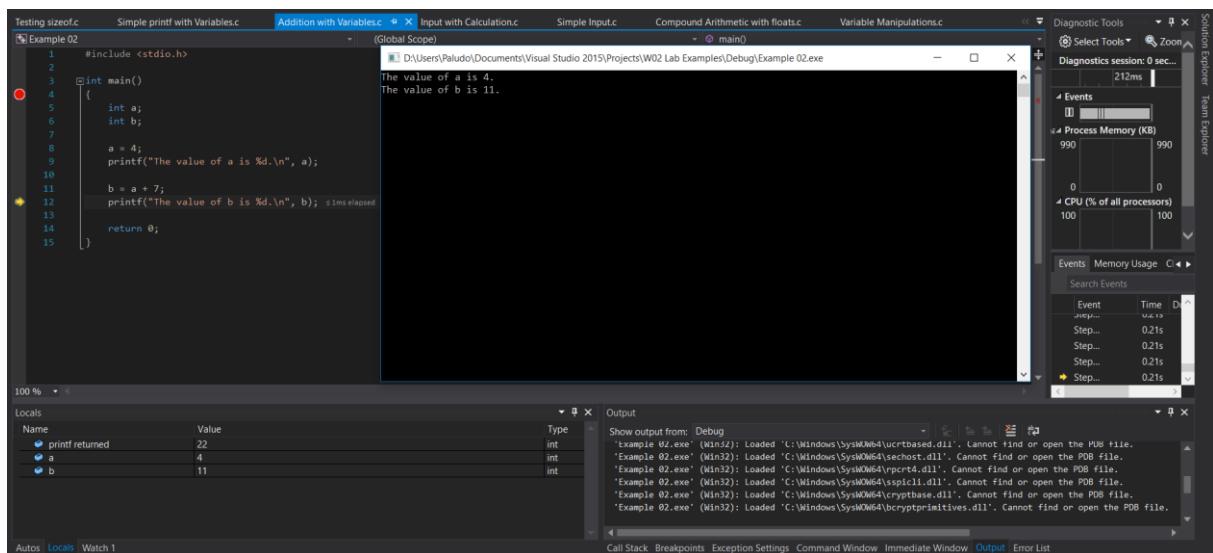
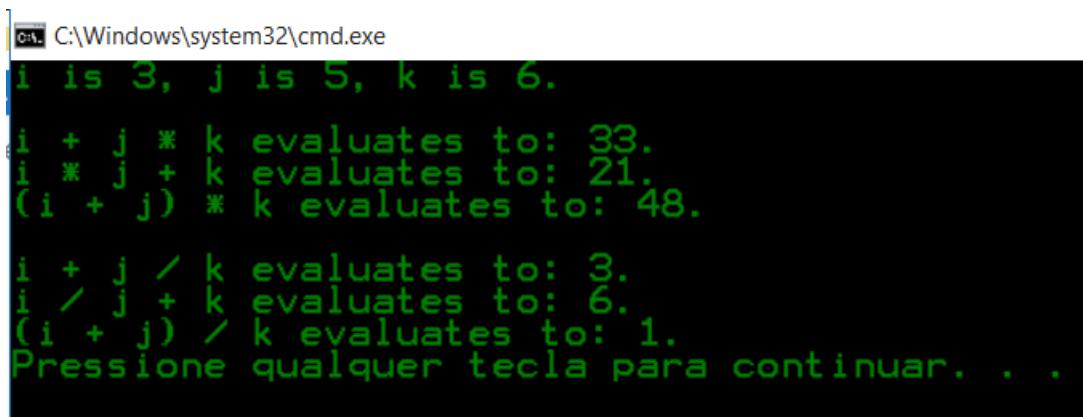


Figure 26: The program prints the value of *b*.

2.2.3 Example 3

The third Week two lab's example initiates and does a sequence of operations with int variables and exhibits the following output:



```
C:\Windows\system32\cmd.exe
i is 3, j is 5, k is 6.
i + j * k evaluates to: 33.
i * j + k evaluates to: 21.
(i + j) * k evaluates to: 48.
i + j / k evaluates to: 3.
i / j + k evaluates to: 6.
(i + j) / k evaluates to: 1.
Pressione qualquer tecla para continuar. . .
```

Figure 27: Order of Operations.

The program does similar mathematical operations changing the order of the operators to demonstrate how the operations work. The code can be seen below:

```
#include <stdio.h>

int main()
{
    int i = 3;
    int j = 5;
    int k = 6;

    int order1 = i + j * k;
    int order2 = i * j + k;
    int order3 = (i + j) * k;

    int order4 = i + j / k;
    int order5 = i / j + k;
    int order6 = (i + j) / k;

    printf("i is %d, j is %d, k is %d.\n\n", i, j, k);
```

```

printf("i + j * k evaluates to: %d.\n", order1);
printf("i * j + k evaluates to: %d.\n", order2);
printf("(i + j) * k evaluates to: %d.\n", order3);

printf("\n");

printf("i + j / k evaluates to: %d.\n", order4);
printf("i / j + k evaluates to: %d.\n", order5);
printf("(i + j) / k evaluates to: %d.\n", order6);

return 0;
}

```

First, the code introduces three variables assigning different values to each of it, straight after, it does a series of sum/multiply and sum/division operations in different orders, charges variables with the results, and prints it.

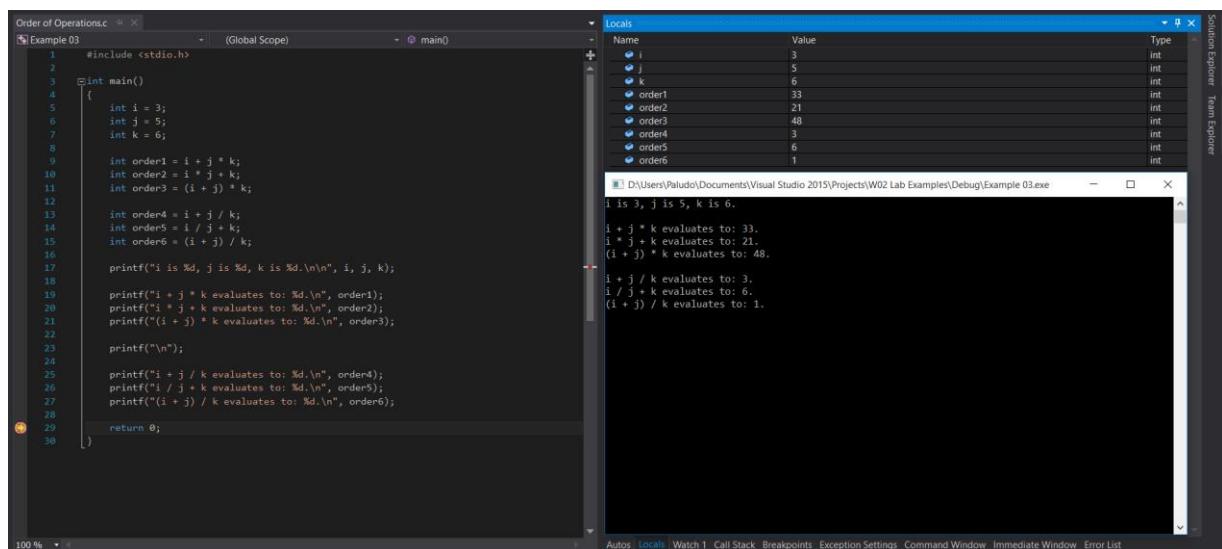
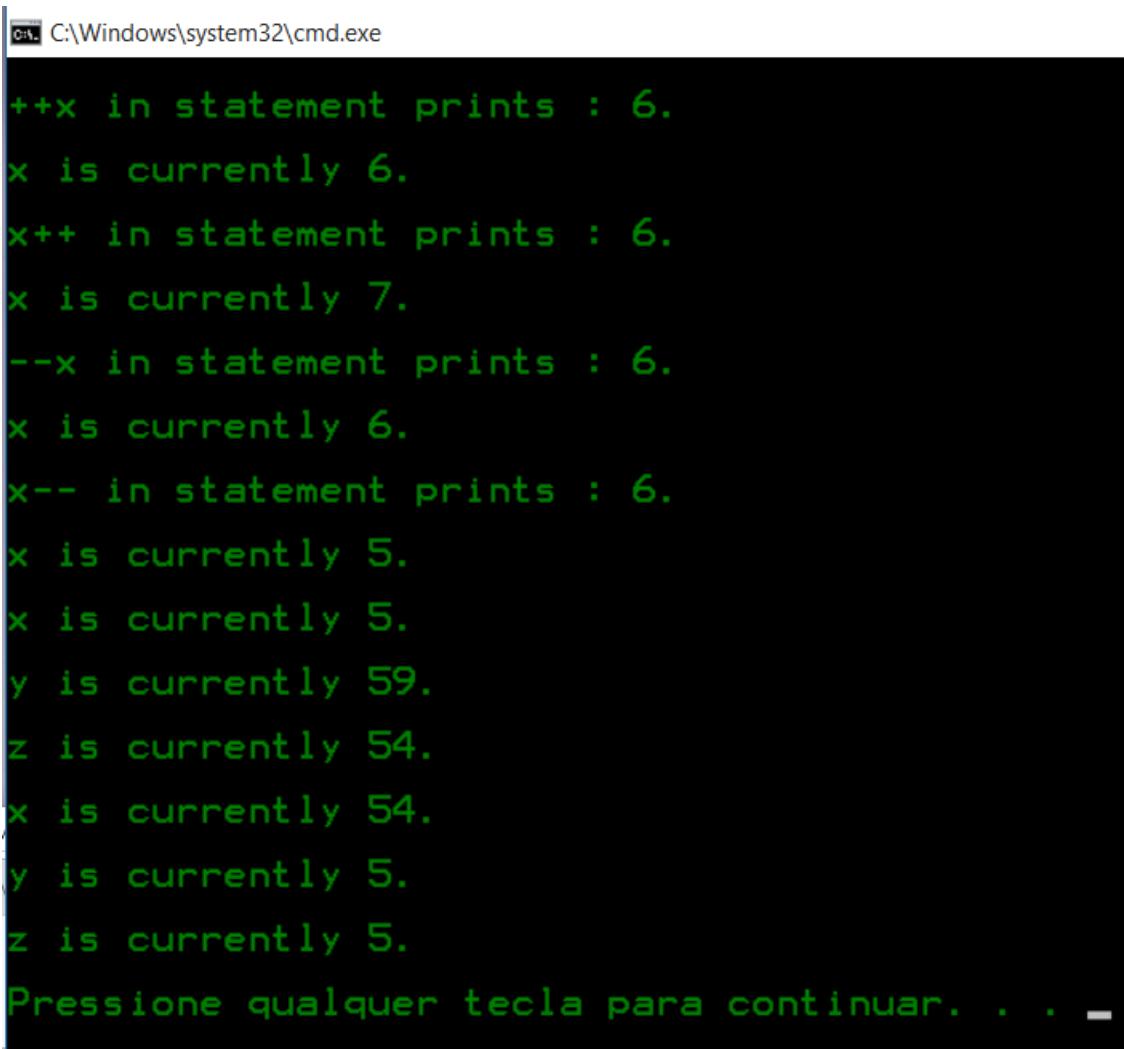


Figure 28: Program debugging and variable observing.

2.2.4 Example 4

The fourth example of the second week laboratory introduces variable manipulations and has the following output:



```
C:\Windows\system32\cmd.exe
++x in statement prints : 6.
x is currently 6.
x++ in statement prints : 6.
x is currently 7.
--x in statement prints : 6.
x is currently 6.
x-- in statement prints : 6.
x is currently 5.
x is currently 5.
y is currently 59.
z is currently 54.
x is currently 54.
y is currently 5.
z is currently 5.
Pressione qualquer tecla para continuar. . . -
```

Figure 29: Variable Manipulations.

The example's code is this:

```
#include <stdio.h>

int main()
{
    int x = 5;

    printf("x is currently %d.\n\n", x);

    printf("++x in statement prints : %d.\n\n", ++x);
    printf("x is currently %d.\n\n", x);

    printf("x++ in statement prints : %d.\n\n", x++);
    printf("x is currently %d.\n\n", x);

    printf("--x in statement prints : %d.\n\n", --x);
    printf("x is currently %d.\n\n", x);

    printf("x-- in statement prints : %d.\n\n", x--);
    printf("x is currently %d.\n\n", x);

    int y = 25 + 34;
    int z = 25 + x + 24;

    printf("x is currently %d.\n\n", x);
    printf("y is currently %d.\n\n", y);
    printf("z is currently %d.\n\n", z);

    y = x;
    x = z;
    z = y;

    printf("x is currently %d.\n\n", x);
    printf("y is currently %d.\n\n", y);
    printf("z is currently %d.\n\n", z);

    return 0;
}
```

Debugging the program, it can be noticed that using `++` or `--` before a variable prints the value of the variable plus one or less one, and prints its value. Otherwise, using `++` or `--` after a variable, prints the value of the variable and adds or takes one from this value, after

printing it. After this, the program sums 25 and 34 and assigns the result to **y**, sums 25, **x** and 24, and assigns the result to **z**, after that, the values of the variables are printed.

At the end, the program switches the variables values using other variables and prints the final value of all the variables.

2.2.5 Example 5

The fifth example of the second week laboratory has the following output:



The screenshot shows a command-line interface window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
a is currently 3.
a is currently 8.
a is currently 40.
a is currently 35.
a is currently 7.
Pressione qualquer tecla para continuar. . . -

Figure 30: Compound Arithmetic with ints.

This is the code written:

```
#include <stdio.h>

int main()
{
    int a = 3;

    printf("a is currently %d. \n", a);

    a += 5;
    printf("a is currently %d. \n", a);

    a *= 5;
    printf("a is currently %d. \n", a);

    a -= 5;
    printf("a is currently %d. \n", a);

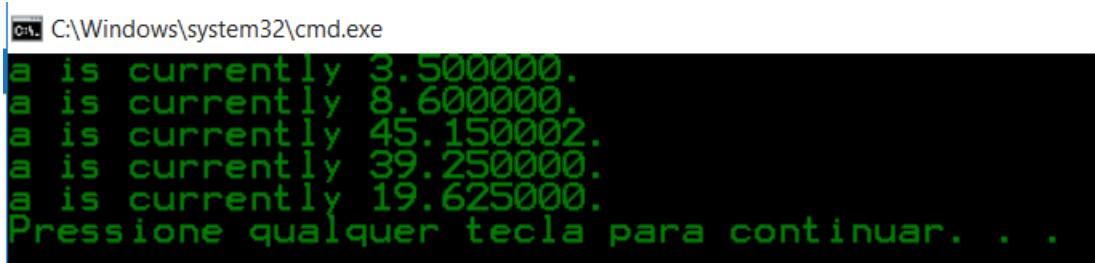
    a /= 5;
    printf("a is currently %d. \n", a);

    return 0;
}
```

Debugging the code we can see that a first receives the value 3, the operation `a += 5` is the same that `a = a + 5` and takes the previous value of the variable, adds 5 to it, and assigns the result to the variable. The same logic is used to the other expressions in this code.

2.2.6 Example 6

Here follows the output of the sixth example of the second week's laboratory:



The screenshot shows a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The output of the program is displayed in green text:
a is currently 3.500000.
a is currently 8.600000.
a is currently 45.150002.
a is currently 39.250000.
a is currently 19.625000.
Pressione qualquer tecla para continuar. . .

Figure 31: Compound Arithmetic with floats.

And its code:

```
#include <stdio.h>

int main()
{
    float a = 3.5f;

    printf("a is currently %f. \n", a);

    a += 5.1f;
    printf("a is currently %f. \n", a);

    a *= 5.25f;
    printf("a is currently %f. \n", a);

    a -= 5.9f;
    printf("a is currently %f. \n", a);

    a /= 2.0f;
    printf("a is currently %f. \n", a);

    return 0;
}
```

First, the value of 3.5 is assigned to the variable, after this, follows a sequence of operation which are very close to the ones in the previous example, but the float variables can have real numbers, while the int variables just assume integer values.

2.2.7 Example 7

The following image is an example of what does the seventh example from week two's laboratory does:

A screenshot of a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
Enter a number: 5
The user input: 5.
Pressione qualquer tecla para continuar. . .

Figure 32: Simple Input.

This is the code written:

```
#include <stdio.h>

int main()
{
    int input;

    printf("Enter a number: ");
    scanf("%d", &input);

    printf("The user input: %d.\n", input);

    return 0;
}
```

The program initiates a variable, asks the user to enter a number, assigns the value entered by the user to the variable and prints its value.

2.2.8 Example 8

The eighth example outputs this screen:



```
C:\Windows\system32\cmd.exe
Enter a number: 5
Five times the input is: 25.
Pressione qualquer tecla para continuar. . .
```

Figure 33: Input with Calculation.

This is the code written:

```
#include <stdio.h>

int main()
{
    int data;

    printf("Enter a number: ");
    scanf("%d", &data);

    data = data * 5;

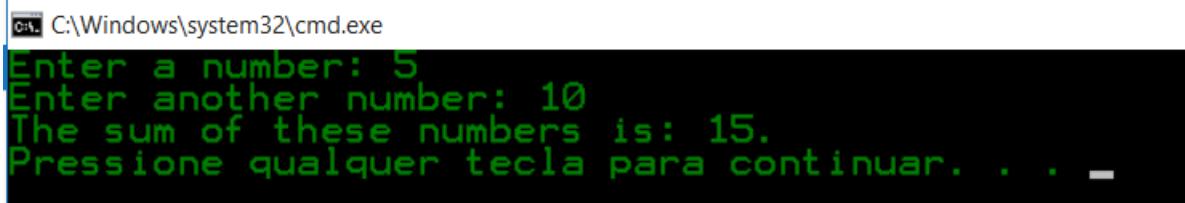
    printf("Five times the input is: %d.\n", data);

    return 0;
}
```

This code takes the user's input, multiplies it by 5 and prints the final result.

2.2.9 Example 9

The ninth example of the second week's laboratory output can be seen below:



```
C:\Windows\system32\cmd.exe
Enter a number: 5
Enter another number: 10
The sum of these numbers is: 15.
Pressione qualquer tecla para continuar. . . -
```

Figure 34: Number Adder.

Here follows the ninth example's code:

```
#include <stdio.h>

int main()
{
    int number1;
    int number2;

    printf("Enter a number: ");
    scanf("%d", &number1);

    printf("Enter another number: ");
    scanf("%d", &number2);

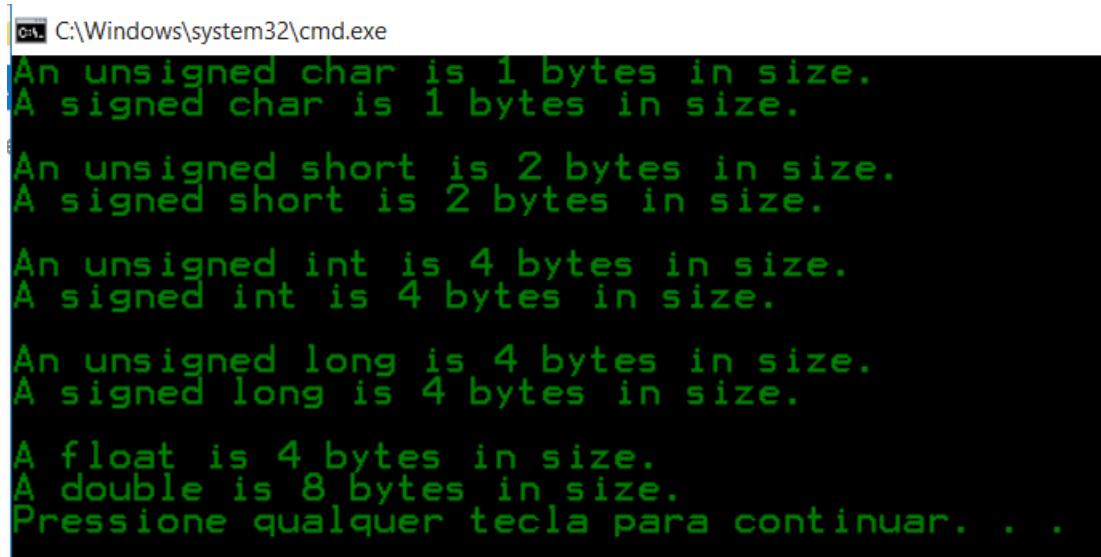
    int sum = number1 + number2;
    printf("The sum of these numbers is: %d.\n", sum);

    return 0;
}
```

Debugging the code we can see that it takes the input of two variables from the user, sums the respective variables and prints the result.

2.2.10 Example 10

The last week two's laboratory example outputs the following screen:



A screenshot of a Windows command prompt window titled 'cmd C:\Windows\system32\cmd.exe'. The window contains the following text output:
An unsigned char is 1 bytes in size.
A signed char is 1 bytes in size.
An unsigned short is 2 bytes in size.
A signed short is 2 bytes in size.
An unsigned int is 4 bytes in size.
A signed int is 4 bytes in size.
An unsigned long is 4 bytes in size.
A signed long is 4 bytes in size.
A float is 4 bytes in size.
A double is 8 bytes in size.
Pressione qualquer tecla para continuar. . .

Figure 35: Testing sizeof.

The following code represents the example:

```
#include <stdio.h>

int main()
{
    printf("An unsigned char is ");
    printf("%d bytes in size.\n", sizeof(unsigned char));
    printf("A signed char is ");
    printf("%d bytes in size. \n\n", sizeof(signed char));

    printf("An unsigned short is ");
    printf("%d bytes in size.\n", sizeof(unsigned short));
    printf("A signed short is ");
    printf("%d bytes in size. \n\n", sizeof(signed short));

    printf("An unsigned int is ");
    printf("%d bytes in size.\n", sizeof(unsigned int));
    printf("A signed int is ");
```

```
printf("%d bytes in size. \n\n", sizeof(signed int));  
  
printf("An unsigned long is ");  
printf("%d bytes in size.\n", sizeof(unsigned long));  
printf("A signed long is ");  
printf("%d bytes in size. \n\n", sizeof(signed long));  
  
printf("A float is ");  
printf("%d bytes in size.\n", sizeof(float));  
  
printf("A double is ");  
printf("%d bytes in size.\n", sizeof(double));  
  
return 0;  
}
```

The shows the size of an unsigned char, a signed char, an unsigned short, a signed short, an unsigned int, a signed int, an unsigned long, a signed long, a float and a double.

2.3 Exercises

2.3.1 Exercise 1

The first exercise of the second week's laboratory asks the student to write a program which asks the user to input three whole numbers, and output these three numbers in entry and reversed orders, as shown:

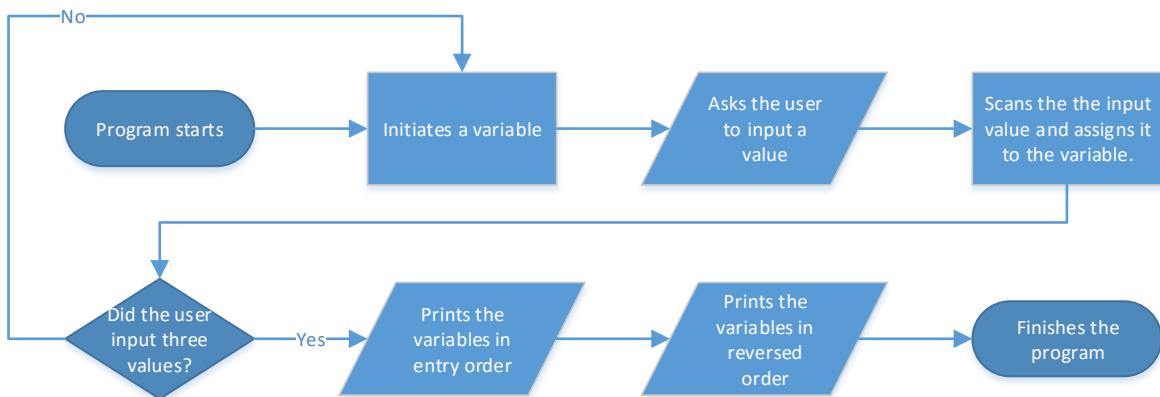
```
Please enter number 1: 15
Please enter number 2: 27
Please enter number 3: 41

Your numbers in entry order:
15
27
41

Your numbers reversed:
41
27
15
```

Figure 36: Three Number Reverse.

To design the program I draw the this flowchart:



With this flowchart, I designed the following code:

```

#include <stdio.h>

int main()
{
    int number_1;
    printf("Please enter number 1: ");
    scanf("%i", &number_1);

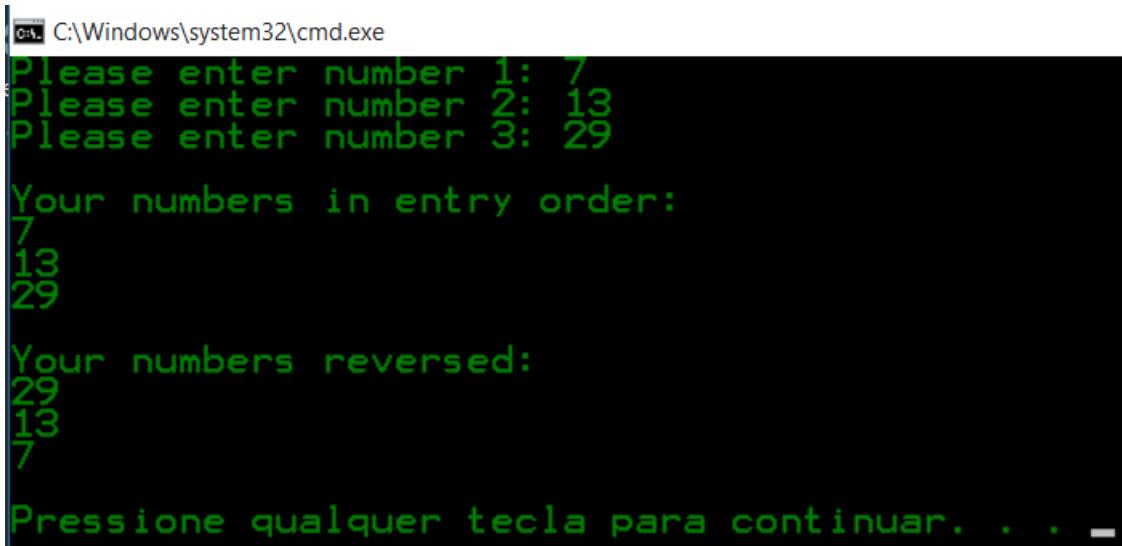
    int number_2;
    printf("Please enter number 2: ");
    scanf("%i", &number_2);

    int number_3;
    printf("Please enter number 3: ");
    scanf("%d", &number_3);

    printf("\nYour numbers in entry order: \n%i\n%i\n%i\n\n",
number_1, number_2, number_3); //number in order
    printf("Your numbers reversed:\n%i\n%i\n%i\n\n", number_3,
number_2, number_1); //numbers reversed

    return 0;
}
  
```

Which outputs the following screen:



```
C:\Windows\system32\cmd.exe
Please enter number 1: 7
Please enter number 2: 13
Please enter number 3: 29

Your numbers in entry order:
7
13
29

Your numbers reversed:
29
13
7

Pressione qualquer tecla para continuar. . . -
```

Figure 37: Three Number Reverse's code output.

While writing the code I had some problems because I forgot to put an & while scanning the variables.

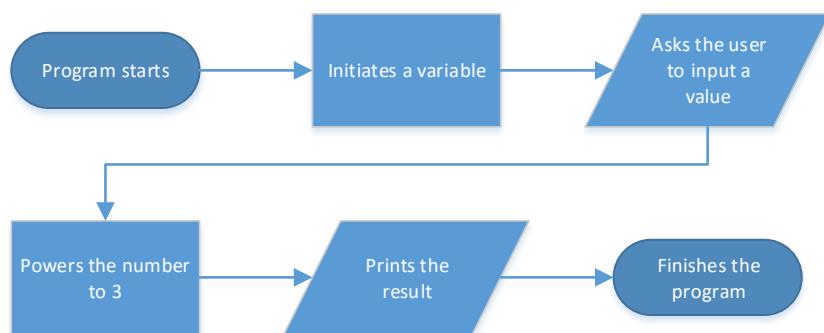
2.3.2 Exercise 2

The second exercise of the week two's laboratory consisted in designing a cube calculator with this output:

```
Please enter a whole number: 5
5 cubed is: 125
```

Figure 38: Cube Calculator.

To design this program I traced the following flowchart:



With this flowchart I developed the following code:

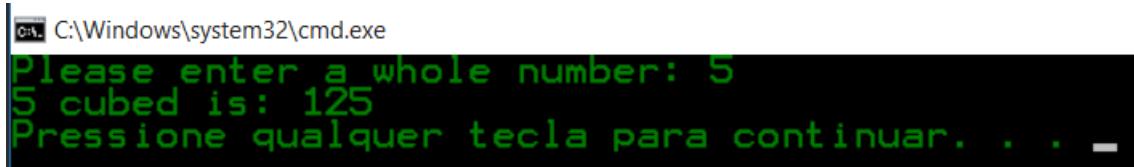
```
#include <stdio.h>
#include <math.h>

int main()
{
    int input;
    printf("Please enter a whole number: ");
    scanf("%d", &input);

    int cube = pow (input, 3);
    printf("%d cubed is: %d\n", input, cube);
```

```
    return 0;  
}
```

That outputs the following prompt:



A screenshot of a Windows Command Prompt window titled 'cmd' located at 'C:\Windows\system32\cmd.exe'. The window displays the following text:
Please enter a whole number: 5
5 cubed is: 125
Pressione qualquer tecla para continuar. . . -

Figure 39: Cube Calculator output.

I decided to use the pow command because, it seems ok to multiply a variable three times, it would be unpracticable to multiply a variable 25 times if you wants to power a number to 25.

2.3.3 Exercise 3

The third exercise of the second week's laboratory asks the programmer to take three internal angles of a triangle and print the third angle and output the following image:

```
A triangle has three internal angles...

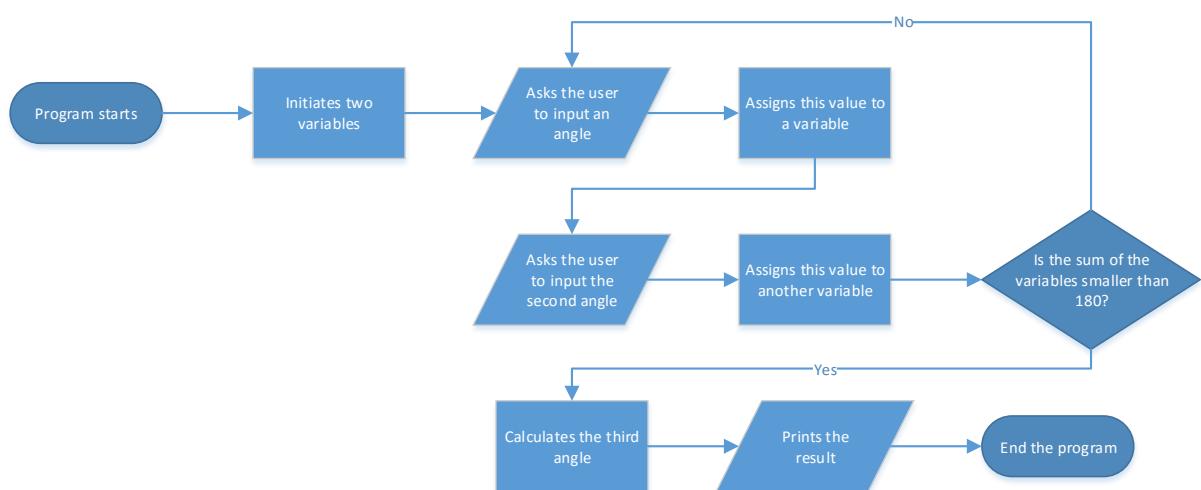
Please enter the first angle: 30
Please enter the second angle: 50

Calculating...

The third angle is: 100 degrees
```

Figure 40: Missing Angle.

To design this code, I draw the following flowchart:



With this flowchart I designed the following code:

```
#include <stdio.h>
#include <math.h>
```

```
int main()
{
    printf("A triangle has three internal angles...");
    printf("\n\n");

    int first_angle;
    printf("Please enter the first angle: ");
    scanf("%d", &first_angle);

    int second_angle;
    printf("Please enter the second angle: ");
    scanf("%d", &second_angle);

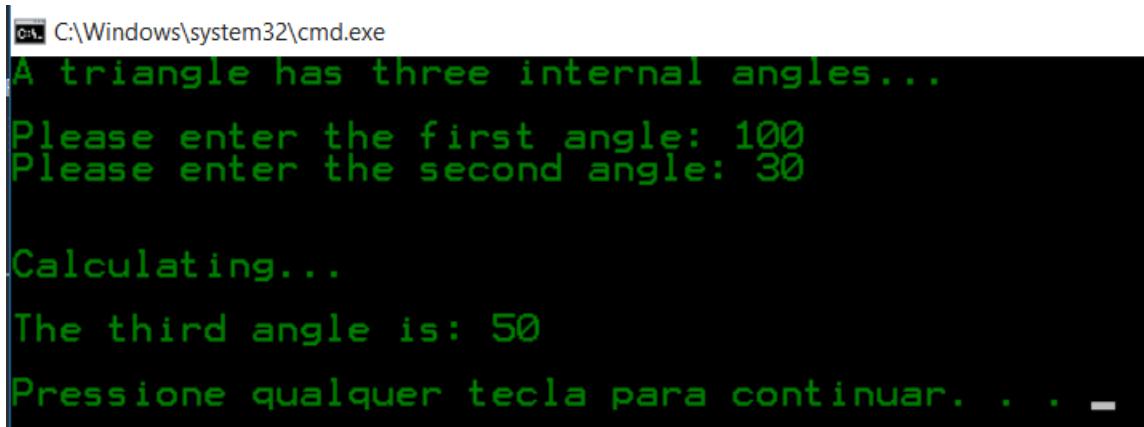
    printf("\n\n");
    printf("Calculating...");
    printf("\n\n");

    while (first_angle + second_angle >= 180)
    {
        printf("I'm sorry Dave, I'm afraid I can't do that.
Please, user, enter the numbers again.");
        printf("\n\n");
        printf("Please enter the first angle: ");
        scanf("%d", &first_angle);
        printf("Please enter the second angle: ");
        scanf("%d", &second_angle);
        printf("\n\n");
        printf("Calculating...");
        printf("\n\n");
    }

    int third_angle = 180 - first_angle - second_angle;
    printf("The third angle is: %d", third_angle);
    printf("\n\n");

    return 0;
}
```

Which outputs:



```
C:\Windows\system32\cmd.exe
A triangle has three internal angles...
Please enter the first angle: 100
Please enter the second angle: 30

Calculating...

The third angle is: 50

Pressione qualquer tecla para continuar. . . -
```

Figure 41: Missing angle output.

To write the code I had to search about the `while` repeating structure. It takes a decision until the parameters meet the ones that follows the `while` command. Davis (2014) says that “The simplest form of looping statement is the while loop.” (p. 72). “With a while loop, the mechanism for repeating a set of statements allows execution to continue for as long as a specified logical expression evaluates to true.” (Horton, 2013, p. 156). According to Perry (1992), “C++ constructs include powerful, but succinct and efficient, looping commands similar to those of other languages you already know. The while loops enable your programs to repeat a series of statements, over and over, as long as a certain condition is always met.” (p. 245). “A repetition statement allows you to specify that an action is to be repeated while some condition remains true.”, stated Deitel and Deitel (2010, p. 63).

2.3.4 Exercise 4

This exercise asks the student to develop a program that takes 5 numbers from the user, calculate and outputs the average of these numbers as this example:

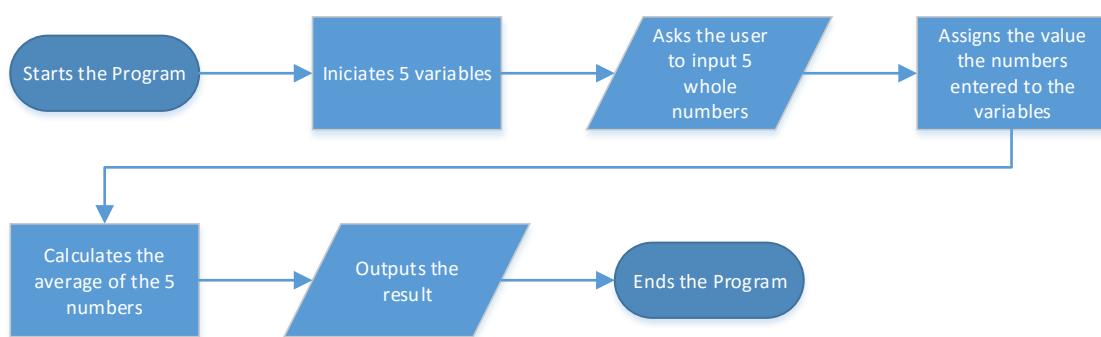
```
Please enter five whole numbers:
```

```
First number: 62
Second number: 34
Third number: 39
Fourth number: 20
Fifth number: 91
```

```
The average of these five numbers is: 49.200001
```

Figure 42: Average of Five.

To designed the code I draw the following flowchart:



Using the flowchart, the following code was written:

```
#include <stdio.h>
#include <math.h>

int main()
{
    printf("Please enter five whole numbers:");
}
```

```
printf("\n\n");

int first_number;
printf("First number: ");
scanf("%d", &first_number);

int second_number;
printf("Second number: ");
scanf("%d", &second_number);

int third_number;
printf("Third number: ");
scanf("%d", &third_number);

int fourth_number;
printf("Fourth number: ");
scanf("%d", &fourth_number);

int fifth_number;
printf("Fifth number: ");
scanf("%d", &fifth_number);

printf("\n\n");

double sum = first_number + second_number + third_number +
fourth_number + fifth_number;
double average = sum / 5;
printf("The average of these five numbers is: %f", average);
printf("\n\n");

return 0;
}
```

The code outputs the following window:

```
C:\Windows\system32\cmd.exe
Please enter five whole numbers:
First number: 5
Second number: 7
Third number: 13
Fourth number: 21
Fifth number: 89

The average of these five numbers is: 27.000000
Pressione qualquer tecla para continuar. . .
```

Figure 43: Average of Five output.

The code worked exactly as it was supposed to.

Before the lecture about coding standards, my code had this style:

```
int main()
{
    int a; //number to input
    int b;
    int c;
    int d;
    int e;

    printf("Please enter five whole numbers:");
    printf("\n\n");

    printf("First number: ");
    scanf("%d", &a);
    printf("Second number: ");
    scanf("%d", &b);
    printf("Third number: ");
    scanf("%d", &c);
    printf("Fourth number: ");
    scanf("%d", &d);
    printf("Fifth number: ");
    scanf("%d", &e);
    printf("\n\n");

    float x = (a + b + c + d + e) / 5; //x is the average
    printf("The average of these five numbers is: %f", x);
    printf("\n\n");

    return 0;
}
```

Figure 44: Coding without standards.

The code after the lecture is this:

```
#include <stdio.h>
#include <math.h>

int main()
{
    printf("Please enter five whole numbers:");
    printf("\n\n");

    int first_number;
    printf("First number: ");
    scanf("%d", &first_number);

    int second_number;
    printf("Second number: ");
    scanf("%d", &second_number);

    int third_number;
    printf("Third number: ");
    scanf("%d", &third_number);

    int fourth_number;
    printf("Fourth number: ");
    scanf("%d", &fourth_number);

    int fifth_number;
    printf("Fifth number: ");
    scanf("%d", &fifth_number);

    printf("\n\n");

    double sum = first_number + second_number + third_number + fourth_number + fifth_number;
    double average = sum / 5;
    printf("The average of these five numbers is: %f", average);
    printf("\n\n");

    return 0;
}
```

Figure 45: Code with standards.

Coding with defined standards is better and easier to other programmers to understand.

2.3.5 Exercise 5

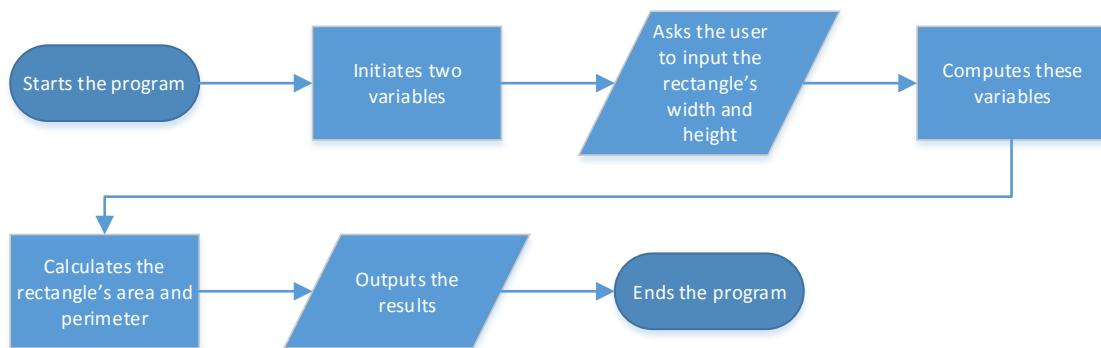
This program should take from the user two numbers, corresponding to the width and height of a rectangle, then calculate the rectangle's perimeter and area, as follows:

```
Rectangle Calculator:
-----
What is the width of the rectangle? 8
What is the height of the rectangle? 5

The rectangle's perimeter is: 26
The rectangle's area is: 40
```

Figure 46: Rectangle Calculator.

To design this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>
#include <math.h>

int main()
{
    printf("Rectangle Calculator:\n");
    // Code for input and calculations goes here
}
```

```
printf("-----");
printf("\n\n");

double width;
printf("What is the width of the rectangle?: ");
scanf("%lf", &width);

double height;
printf("What is the height of the rectangle?: ");
scanf("%lf", &height);

printf("\n\n");

double perimeter = 2 * (width + height);
double area = width * height;

printf("The rectangle's perimeter is: %f", perimeter);
printf("\n");

printf("The rectangle's area is: %f", area);
printf("\n\n");

return 0;
}
```

The code outputs the following window:

```
C:\Windows\system32\cmd.exe
Rectangle Calculator:
-----
What is the width of the rectangle?: 13
What is the height of the rectangle?: 7

The rectangle's perimeter is: 40.000000
The rectangle's area is: 91.000000
Pressione qualquer tecla para continuar. . . -
```

Figure 47: Rectangle Calculator output.

As shown in the code, it initiates the width variable, asks the user to input a value, then assigns the value to the variable. The same sequence is used for the height. The program,

then, calculates the perimeter and area of the rectangle using new variables, and outputs the results.

I decided to use double because the user may want to input some real values to the rectangle's dimensions.

2.3.6 Exercise 6

This exercise asked the students to write a program which asks the user to input two whole numbers. Then perform addition, subtraction, multiplication, integer division, floating-point division, calculate the remainder of the integer division using modulus, and output the results of each calculation.

The example to be followed can be seen beneath:

Math Operation Test:

Please enter the first whole number: 5
Please enter the second whole number: 9

Calculating...

5 + 9 is 14

5 - 9 is -4

5 * 9 is 45

With integer division:

5 / 9 is 0

With floating point division:

5 / 9 is 0.555556

The remainder of 5 divided by 9 is 5

Figure 48: Math Operation Test

To develop this program, I wrote the following pseudo code:

```
Include the basic and math libs;  
  
Start the program;  
  
    Prints the program's name;  
  
    Initiates the first number variable, an integer;  
    Asks the user to input a whole number;  
    Assigns the entered value to the respective variable;  
  
    Initiates the second number variable, also an integer;  
    Asks the user to input another whole number;  
    Assigns the this value to the second number variable;  
  
    Calculates the sum of the numbers and assigns it to a new  
variable;  
    Prints the result;  
  
    Calculates the subtraction of the numbers and assigns it to  
another variable;  
    Prints the result;  
  
    Multiplies the two values and assigns it to a variable;  
    Prints the result;  
  
    Divides the two numbers and assigns it to an integer variable;  
    Prints the result;  
  
    Divides the two numbers and assigns it to a float variable;  
    Prints the result;  
  
    Calculates the remainder of the integer division using modulus;  
    Prints the result;  
  
Finishes the program.
```

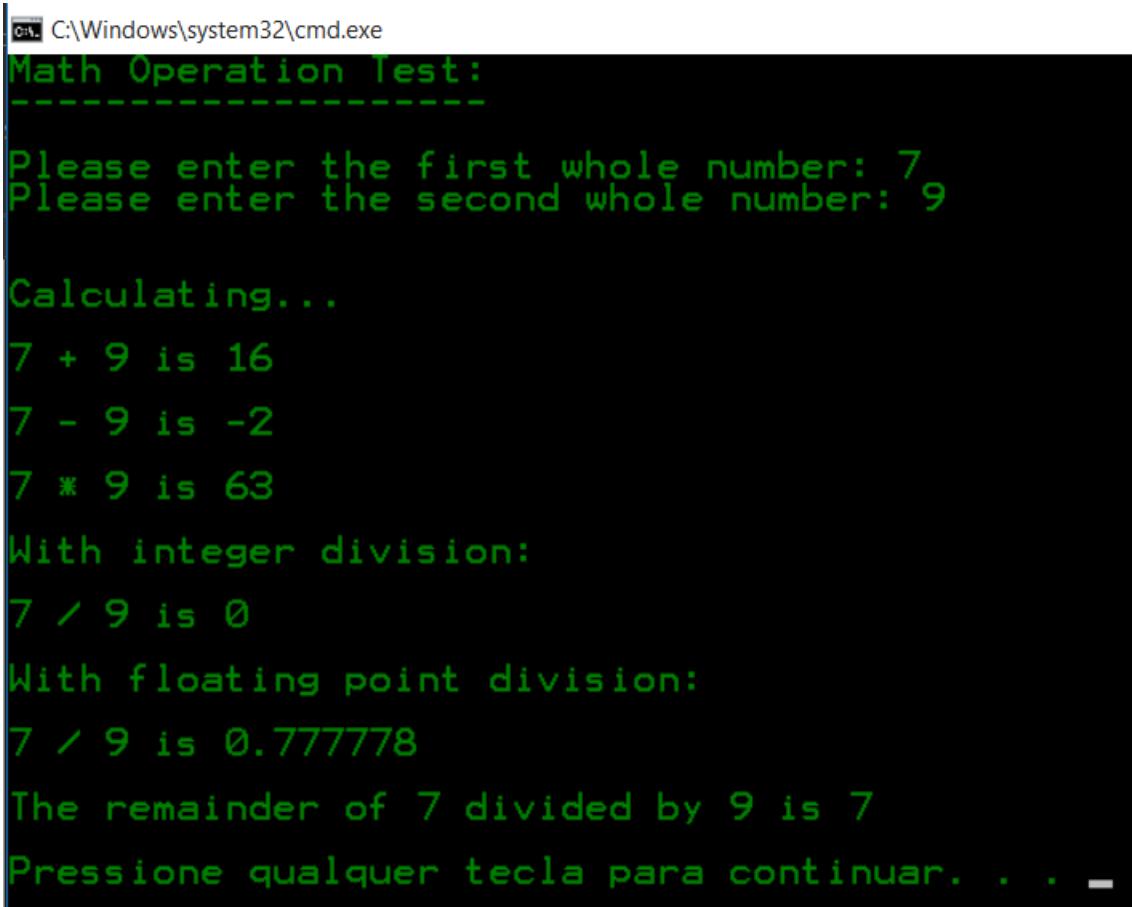
Using this pseudo code, I wrote the following code:

```
#include <stdio.h>  
#include <math.h>  
  
int main()
```

```
{  
    printf("Math Operation Test:\n");  
    printf("-----");  
    printf("\n\n");  
  
    int first_number;  
    printf("Please enter the first whole number: ");  
    scanf("%d", &first_number);  
  
    int second_number;  
    printf("Please enter the second whole number: ");  
    scanf("%d", &second_number);  
  
    printf("\n\n");  
    printf("Calculating...");  
    printf("\n\n");  
  
    int x = first_number + second_number;  
    printf("%d + %d is %d", first_number, second_number, x);  
    printf("\n\n");  
  
    int y = first_number - second_number;  
    printf("%d - %d is %d", first_number, second_number, y);  
    printf("\n\n");  
  
    int z = first_number * second_number;  
    printf("%d * %d is %d", first_number, second_number, z);  
    printf("\n\n");  
  
    printf("With integer division:");  
    printf("\n\n");  
  
    int m = first_number / second_number;  
    printf("%d / %d is %d", first_number, second_number, m);  
    printf("\n\n");  
  
    printf("With floating point division:");  
    printf("\n\n");  
  
    float n = first_number / (float)second_number;  
    printf("%d / %d is %f", first_number, second_number, n);  
    printf("\n\n");  
  
    int o = first_number % second_number;  
    printf("The remainder of %d divided by %d is %d", first_number,  
    second_number, o);  
    printf("\n\n");
```

```
    return 0;  
}
```

This program outputs the following window:



C:\Windows\system32\cmd.exe
Math Operation Test:

Please enter the first whole number: 7
Please enter the second whole number: 9

Calculating...
7 + 9 is 16
7 - 9 is -2
7 * 9 is 63
With integer division:
7 / 9 is 0
With floating point division:
7 / 9 is 0.777778
The remainder of 7 divided by 9 is 7
Pressione qualquer tecla para continuar. . . -

Figure 49: Mat Operation Test output.

The program was tested with a variety of numbers and it worked without problems in all of them.

2.3.7 Exercise 7

This exercise gives a pseudo code and asks the student to write the corresponding source code, that outputs this example:

```
Enter the x value: -2  
Enter the y value: 6  
Result: 448.000000
```

Figure 50: Pseudo Code to Source Code.

This is the pseudo code given:

```
READ x  
READ y  
COMPUTE a AS x * y  
COMPUTE b AS x + y  
COMPUTE result AS b2 + a * (b - x) * (a + y)  
PRINT result
```

Figure 51: Pseudo Code.

This is the code written:

```
#include <stdio.h>  
#include <math.h>  
  
int main()  
{  
    int x;  
    printf("Enter the x value: ");  
    scanf("%d", &x);  
    printf("\n\n");  
  
    int y;  
    printf("Enter the y value: ");
```

```
scanf("%d", &y);
printf("\n\n");

float a = x * y;
float b = x + y;
float c = pow(b, 2) + a * (b - x) * (a + y);

printf("Result: %f",
printf("\n\n");

return 0;
}
```

The program outputs the following window:

```
C:\Windows\system32\cmd.exe
Enter the x value: 13

Enter the y value: 7

Result: 62826.000000

Pressione qualquer tecla para continuar. . .
```

Figure 52: Pseudo Code to Source Code output.

The program was tested with a long range of variables and any bugs were found on it.

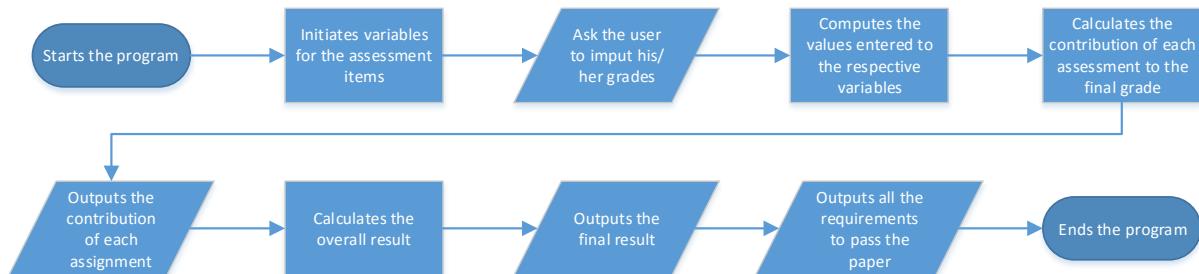
2.3.8 Exercise 8

This exercise asks the user to write a code that calculates the Programming 1 classes' grades, and the contribution per assessment item as follows:

```
COMP500/ENSE501 Result Calculator:  
=====  
  
Enter the score (out of 100) for the Reporting Journal: 50  
Enter the score (out of 100) for the Practical Test 1: 25  
Enter the score (out of 100) for the Practical Test 2: 50  
Enter the score (out of 100) for the Practical Test 3: 75  
Enter the score (out of 100) for the Final Practical Exam: 50  
  
Calculating...  
  
Reporting Journal (worth 15%) contributes: 7.500000  
Practical Test 1 (worth 10%) contributes: 2.500000  
Practical Test 2 (worth 10%) contributes: 5.000000  
Practical Test 3 (worth 15%) contributes: 11.250000  
Final Practical Exam (worth 50%) contributes: 25.000000  
  
Overall result total: 51.250000%  
  
Remember, to pass the paper, a student must achieve:  
- At least 80% attendance and participation in the  
individual's scheduled lab tutorial stream, AND  
- A minimum mark of 40% for the Final Practical Exam, AND  
- A minimum C- (50%) overall grade.
```

Figure 53: P1 Result Calculator.

To design this program, I draw the following flowchart:



With this flowchart, I wrote the following code:

```

#include <stdio.h>
#include <math.h>

int main()
{
    printf("COMP500/ENSE501 Result Calculator:\n");
    printf("===== \n");
    printf("\n\n");

    double reporting_journal = 0;
    printf("Enter the score (out of 100) for the Reporting Journal:");
    scanf("%lf", &reporting_journal);

    double test_1 = 0;
    printf("Enter the score (out of 100) for the Practical Test 1:");
    scanf("%lf", &test_1);

    double test_2 = 0;
    printf("Enter the score (out of 100) for the Practical Test 2:");
    scanf("%lf", &test_2);

    double test_3 = 0;
    printf("Enter the score (out of 100) for the Practical Test 3:");
    scanf("%lf", &test_3);
  
```

```

double final_exam = 0;
printf("Enter the score (out of 100) for the Final Practical
Exam: ");
scanf("%lf", &final_exam);

printf("\n\n");
printf("Calculating... ");
printf("\n\n");

double journal_contribution = 0.15 * reporting_journal;
printf("Reporting Journal (worth 15%%) contributes: %f\n",
journal_contribution);

double test_1_contribution = 0.1 * test_1;
printf("Practical Test 1 (worth 10%%) contributes: %f\n",
test_1_contribution);

double test_2_contribution = 0.1 * test_2;
printf("Practical Test 2 (worth 10%%) contributes: %f\n",
test_2_contribution);

double test_3_contribution = 0.15 * test_3;
printf("Practical Test 3 (worth 15%%) contributes: %f\n",
test_3_contribution);

double final_exam_contribution = 0.5 * final_exam;
printf("Final Practical Exam (worth 50%%) contributes: %f\n",
final_exam_contribution);
printf("\n\n");

double overall_grade = journal_contribution +
test_1_contribution + test_2_contribution + test_3_contribution +
final_exam_contribution;

printf("Overall result total: %f", overall_grade);
printf("\n\n");

printf("Remember, to pass the paper, a student must
achieve:\n");
printf("- At least 80% attendance and participation in the\n");
printf("individual's scheduled lab tutorial stream, AND\n");
printf("- A minimum mark of 40%% for the Final Practical Exam,
AND\n");
printf("- A minimum C- (50%%) overall grade.");
printf("\n\n");

return 0;

```

}

The program outputs the following window:

```
C:\> C:\Windows\system32\cmd.exe
COMP500/ENSE501 Result Calculator:
=====
Enter the score (out of 100) for the Reporting Journal: 100
Enter the score (out of 100) for the Practical Test 1: 55
Enter the score (out of 100) for the Practical Test 2: 60
Enter the score (out of 100) for the Practical Test 3: 75
Enter the score (out of 100) for the Final Practical Exam: 60

Calculating...

Reporting Journal (worth 15%) contributes: 15.000000
Practical Test 1 (worth 10%) contributes: 5.500000
Practical Test 2 (worth 10%) contributes: 6.000000
Practical Test 3 (worth 15%) contributes: 11.250000
Final Practical Exam (worth 50%) contributes: 30.000000

Overall result total: 67.750000

Remember, to pass the paper, a student must achieve:
- At least 80% attendance and participation in the
individual's scheduled lab tutorial stream, AND
- A minimum mark of 40% for the Final Practical Exam, AND
- A minimum C- (50%) overall grade.

Pressione qualquer tecla para continuar. . .
```

Figure 54: P1 Result Calculator output.

This code asks from the student his/her grades on all tests, reporting journal, and exam, gives the respective contribution to the final grade, outputs the final grade and all the requirements that the student must have to pass the paper.

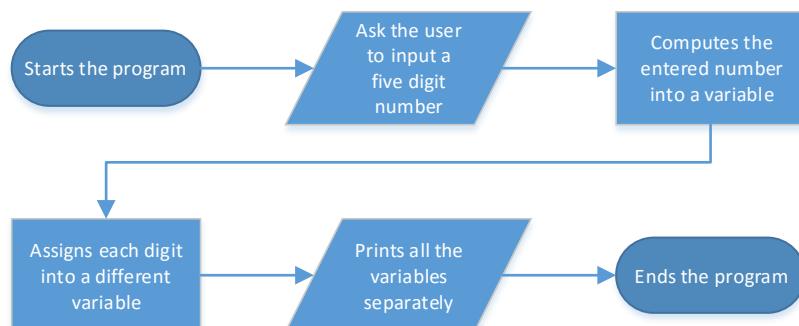
2.3.9 Exercise 9

The ninth exercise asks the student to write a code that asks the user to input a five digit number and then print the number as shown:

```
Please enter a five digit number: 37892
3...7...8...9...and...2
```

Figure 55: Number Separation.

To write this program, I draw the following flowchart:



With this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    int input = 0;
    printf("Please enter a five digit number:");
    scanf("%d", &input);
    printf("\n");

    //Example: 68573
    int first = input / 10000;
        //It returns the last number (6)
    int second = (input / 1000) - (first * 10);
```

```

    //I know the brackets are not necessary, but it looks
better for me this way
    //((input/10000) will return the two last numbers (68)
    //Now our first number is on the tens, so we can just
multiply it by 10 and subtract from the result
    int third = (input / 100) - (second * 10) - (first * 100);
    //Since the program has a logical pattern, it can do
automatically with any number, but I couldn't think of how to do it
yet
    //It will probably use an if and a while to generate the
variables
    //if, for, while and switch may be used
    //to print the variables use a switch inside another
structure
    int fourth = (input / 10) - (third * 10) - (second * 100) -
(first * 1000);
    int fifth = (input)-(fourth * 10) - (third * 100) - (second *
1000) - (first * 10000);

    printf("%d...%d...%d...%d...and...%d", first, second, third,
fourth, fifth);
    printf("\n\n");
}

```

That outputs the following window:

```

C:\Windows\system32\cmd.exe
Please enter a five digit number:59833
5...9...8...3...and...3
Pressione qualquer tecla para continuar. . .

```

Figure 56: Number Separation output.

This program was extremely hard to design and it uses a lot of mathematical logic to do so. All the variables are integers, the first variable stores the value entered by the user. The other five variables, stores each digit of the input number.

The first number just has to be divided by 10000 and it will output the first digit, because that's the truncated result of the operation.

To take the second digit, we divide the number by 1000, it will result the two last digits, then we have to take out the first digit, to do so, subtracted the first result multiplied by ten from this operation.

The other digits follows the same logic.

3 Week Three

3.1 Lectures

In the third week we had lectures about integer overflow, which occurs when a numerical value is larger than the number that can be represented by the variable. For example, in the Donkey Kong game, due to an integer overflow, it is not possible to progress past level 22.

The C math library was also presented, this library includes very useful functions, such as: pow (to power a number), sqrt (returns the square root of a number), exp (raises e to a number).

Random numbers are very useful to a big diversity of programs, dice rolling and other lucky games, for example, are programs that need this kind of function.

Arrays can be used when the programmer wants to declare a large number of variables, with arrays, it can be done in one single statement. Different values can be assigned to each array, and these values can be set when the array is initialized. Each element of the array can be read separately.

The Visual Studio Memory Window can be very helpful to see the changes done in the variables while the program is running and where exactly they are stored in the memory.

The compiler needs a size for an array when it is compiling the code, before it runs the program, because the size of an array cannot be changed while the program is being executed. The dimension of an array can be avoided by declaring all its elements when it is initialized.

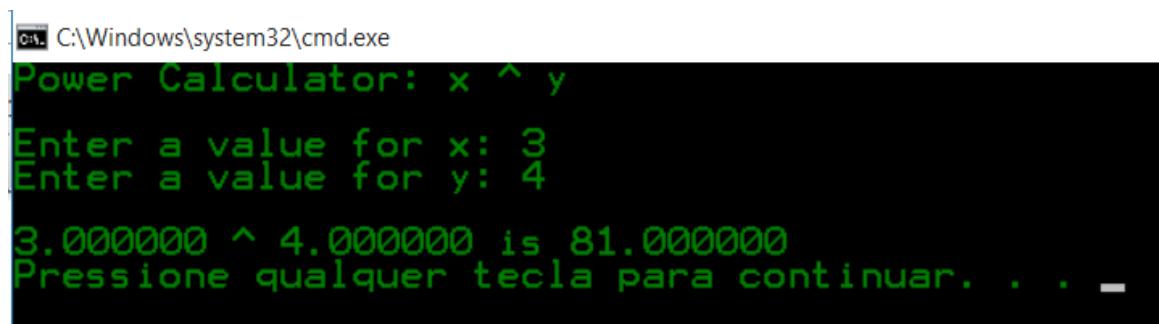
A C-String is an array of characters, these arrays always have an \0 to sign the end of the string.

These week it was also mentioned by the lecturer some orientations about how to proceed to be a successful programmer. Read the question, desire to solve a problem, understand the problem, note what I know and don't know how to do, look for shortcuts, have a plan are some of these orientations.

3.2 Examples

3.2.1 Example 1

The first example is a program which asks the user to input two values, pow the first variable to the second and prints the result with this input:



```
C:\Windows\system32\cmd.exe
Power Calculator: x ^ y
Enter a value for x: 3
Enter a value for y: 4
3.000000 ^ 4.000000 is 81.000000
Pressione qualquer tecla para continuar. . . -
```

Figure 57: Math Power.

The written code is this:

```
#include <stdio.h>
#include <math.h>

int main()
{
    float x = 0.0f;
    float y = 0.0f;

    printf("Power Calculator: x ^ y\n\n");

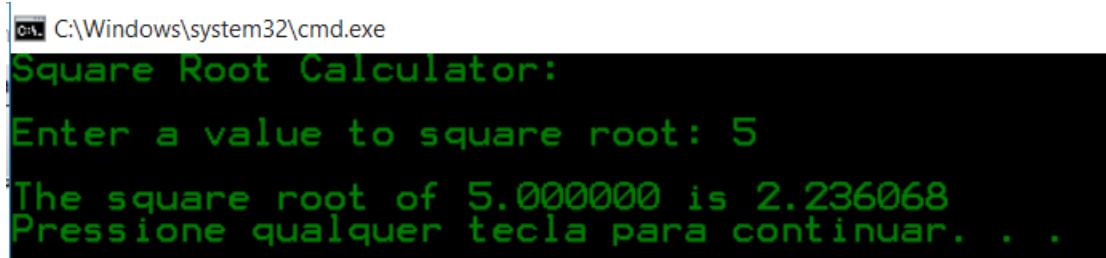
    printf("Enter a value for x: ");
    scanf("%f", &x);
    printf("Enter a value for y: ");
    scanf("%f", &y);

    double result = pow(x, y);
```

```
printf("\n%f ^ %f is %f\n", x, y, result);  
return 0;  
}
```

3.2.2 Example 2

This example code asks the user to input a value to square root, calculates the square root of the value entered by the user and outputs the value.



The screenshot shows a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
Square Root Calculator:
Enter a value to square root: 5
The square root of 5.000000 is 2.236068
Pressione qualquer tecla para continuar. . .

Figure 58: Square root.

The written code is this:

```
#include <stdio.h>
#include <math.h>

int main()
{
    float x = 0.0f;

    printf("Square Root Calculator:\n\n");

    printf("Enter a value to square root: ");
    scanf("%f", &x);

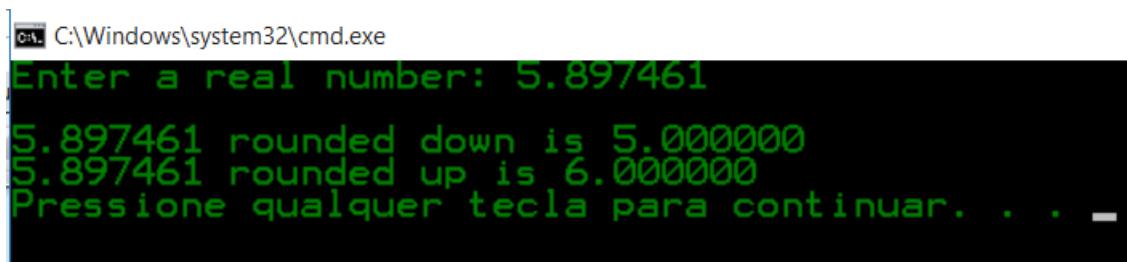
    double result = sqrt(x);

    printf("\nThe square root of %f is %f\n", x, result);

    return 0;
}
```

3.2.3 Example 3

This example asks the user to input a real number, processes the floored and ceiling values of this variable.



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The user has entered the command 'Enter a real number: 5.897461'. The program then outputs three lines: '5.897461 rounded down is 5.000000', '5.897461 rounded up is 6.000000', and 'Pressione qualquer tecla para continuar. . . -'. The text is in green on a black background.

Figure 59: Math Floor and Ceiling.

This is the written code:

```
#include <stdio.h>
#include <math.h>

int main()
{
    float x = 0.0f;

    printf("Enter a real number: ");
    scanf("%f", &x);

    double x_floored = floor(x);
    double x_ceiling = ceil(x);

    printf("\n%f rounded down is %f\n", x, x_floored);
    printf("%f rounded up is %f\n", x, x_ceiling);

    return 0;
}
```

3.2.4 Example 4

The fourth example rolls a dice with random results.



A screenshot of a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
You rolled a: 1
Pressione qualquer tecla para continuar. . .

Figure 60: Simple Random Number.

The written code is this:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
    srand(time(0));

    int a_dice = 0;

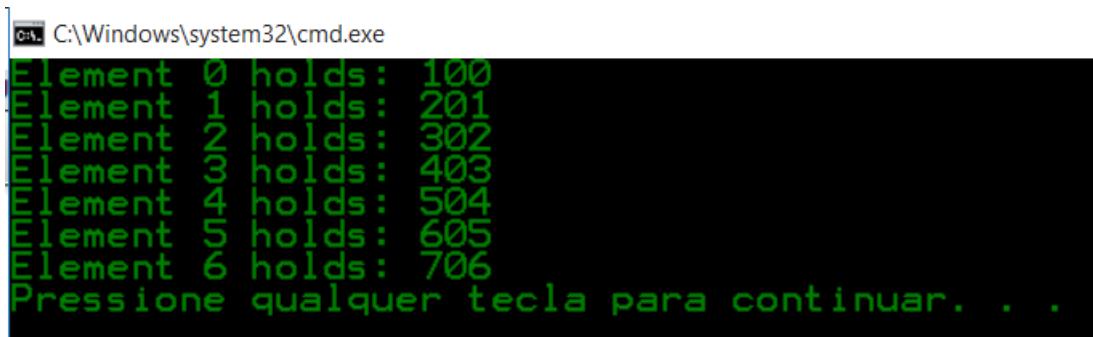
    a_dice = (rand() % 6) + 1;

    printf("You rolled a: %d\n", a_dice);

    return 0;
}
```

3.2.5 Example 5

This example initiates an int array with 7 elements without defining its values, define the value of each element separately and prints each element in one different line.



```
C:\Windows\system32\cmd.exe
Element 0 holds: 100
Element 1 holds: 201
Element 2 holds: 302
Element 3 holds: 403
Element 4 holds: 504
Element 5 holds: 605
Element 6 holds: 706
Pressione qualquer tecla para continuar. . .
```

Figure 61: Simple int Array.

The written code is this:

```
#include <stdio.h>

int main()
{
    int basic_array[7];

    basic_array[0] = 100;
    basic_array[1] = 201;
    basic_array[2] = 302;
    basic_array[3] = 403;
    basic_array[4] = 504;
    basic_array[5] = 605;
    basic_array[6] = 706;

    printf("Element 0 holds: %d\n", basic_array[0]);
    printf("Element 1 holds: %d\n", basic_array[1]);
    printf("Element 2 holds: %d\n", basic_array[2]);
    printf("Element 3 holds: %d\n", basic_array[3]);
    printf("Element 4 holds: %d\n", basic_array[4]);
    printf("Element 5 holds: %d\n", basic_array[5]);
    printf("Element 6 holds: %d\n", basic_array[6]);
```

```
    return 0;  
}
```

3.2.6 Example 6

This example initiates an array with 7 elements, without defining the size of the array, but defining the value of each element, prints the value of each element, multiplies all the elements by -1 and prints all the results.



```
C:\Windows\system32\cmd.exe
Element 0 holds: 10
Element 1 holds: 21
Element 2 holds: 32
Element 3 holds: 43
Element 4 holds: 54
Element 5 holds: 65
Element 6 holds: 76
Element 0 holds: -10
Element 1 holds: -21
Element 2 holds: -32
Element 3 holds: -43
Element 4 holds: -54
Element 5 holds: -65
Element 6 holds: -76
Pressione qualquer tecla para continuar. . .
```

Figure 62: Array With Initialisation.

The written code is this:

```
#include <stdio.h>
int main()
{
    int basic_array[] = { 10, 21, 32, 43, 54, 65, 76 };

    printf("Element 0 holds: %d\n", basic_array[0]);
    printf("Element 1 holds: %d\n", basic_array[1]);
    printf("Element 2 holds: %d\n", basic_array[2]);
    printf("Element 3 holds: %d\n", basic_array[3]);
    printf("Element 4 holds: %d\n", basic_array[4]);
    printf("Element 5 holds: %d\n", basic_array[5]);
    printf("Element 6 holds: %d\n", basic_array[6]);
```

```
// Flip the sign on each element:  
basic_array[0] *= -1;  
basic_array[1] *= -1;  
basic_array[2] *= -1;  
basic_array[3] *= -1;  
basic_array[4] *= -1;  
basic_array[5] *= -1;  
basic_array[6] *= -1;  
  
printf("Element 0 holds: %d\n", basic_array[0]);  
printf("Element 1 holds: %d\n", basic_array[1]);  
printf("Element 2 holds: %d\n", basic_array[2]);  
printf("Element 3 holds: %d\n", basic_array[3]);  
printf("Element 4 holds: %d\n", basic_array[4]);  
printf("Element 5 holds: %d\n", basic_array[5]);  
printf("Element 6 holds: %d\n", basic_array[6]);  
  
return 0;  
}
```

3.2.7 Example 7

This example initiates a float array, prints the value of all elements, initiates another array which takes the values of the other array using the fabs command, and prints the value both of the arrays.

```
C:\Windows\system32\cmd.exe
Element 0 holds: -2.100000
Element 1 holds: 3.700000
Element 2 holds: 5.100000
Element 3 holds: -0.500000
Element 4 holds: -2.900000

|-2.100000| is 2.100000
|3.700000| is 3.700000
|5.100000| is 5.100000
|-0.500000| is 0.500000
|-2.900000| is 2.900000
Pressione qualquer tecla para continuar. . .
```

Figure 63: Simple float Array with Absolute Value.

The written code is this:

```
#include <stdio.h>
#include <math.h>

int main()
{
    float data[] = { -2.1f, 3.7f, 5.1f, -0.5f, -2.9f };

    printf("Element 0 holds: %f\n", data[0]);
    printf("Element 1 holds: %f\n", data[1]);
    printf("Element 2 holds: %f\n", data[2]);
    printf("Element 3 holds: %f\n", data[3]);
    printf("Element 4 holds: %f\n", data[4]);
    printf("\n");

    float absolute[5];
```

```
// Calculate the absolute value of each element:  
absolute[0] = fabs(data[0]);  
absolute[1] = fabs(data[1]);  
absolute[2] = fabs(data[2]);  
absolute[3] = fabs(data[3]);  
absolute[4] = fabs(data[4]);  
  
printf("|\%f| is %f\n", data[0], absolute[0]);  
printf("|\%f| is %f\n", data[1], absolute[1]);  
printf("|\%f| is %f\n", data[2], absolute[2]);  
printf("|\%f| is %f\n", data[3], absolute[3]);  
printf("|\%f| is %f\n", data[4], absolute[4]);  
  
return 0;  
}
```

3.2.8 Example 8

This example rolls 5 Yahtzee random dices and outputs the results in simple and in a fancy outputs:

```
C:\Windows\system32\cmd.exe
First dice : 5
Second dice : 4
Third dice : 2
Fourth dice : 6
Fifth dice : 2

+---+ +---+ +---+ +---+ +---+
| 5 | | 4 | | 2 | | 6 | | 2 |
+---+ +---+ +---+ +---+ +---+
Pressione qualquer tecla para continuar. . .
```

Figure 64: Yahtzee Dice Roller with Array.

The written code is this:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
    srand(time(0));

    int dice[5];

    // Roll five Yahtzee (Milton Bradley, 1973) dice:
    dice[0] = (rand() % 6) + 1;
    dice[1] = (rand() % 6) + 1;
    dice[2] = (rand() % 6) + 1;
    dice[3] = (rand() % 6) + 1;
    dice[4] = (rand() % 6) + 1;

    // Simple output:
    printf("First dice : %d\n", dice[0]);
    printf("Second dice : %d\n", dice[1]);
```

```
printf("Third dice : %d\n", dice[2]);
printf("Fourth dice : %d\n", dice[3]);
printf("Fifth dice : %d\n", dice[4]);
printf("\n");

// Fancy output:
printf("----+ ----+ ----+ ----+ ----+\n");
printf("| %d | | %d | ", dice[0], dice[1]);
printf(" | %d | | %d | ", dice[2], dice[3]);
printf(" | %d |\n", dice[4]);
printf("----+ ----+ ----+ ----+ ----+\n");

return 0;
}
```

3.2.9 Example 9

This example initiates two different arrays (an int and a char array) with initial values, prints the values of all the elements, changes these values and prints again. For the char array, the elements are printed in two different ways.

```
C:\Windows\system32\cmd.exe
n, each element: 0 0 0 0
t, each element: a b c
t, as C-String: abc

Next... assigning to each element in the arrays...

n, each element: 15 25 35 45
t, each element: S D H
t, as C-String: SDH
Pressione qualquer tecla para continuar. . .
```

Figure 65: int and char Arrays.

The written code is this:

```
#include <stdio.h>

int main()
{
    int n[4] = { 0, 0, 0, 0 };
    char t[4] = { 'a', 'b', 'c', '\0' };

    // Print out the arrays:
    printf("n, each element: ");
    printf("%d %d %d %d\n", n[0], n[1], n[2], n[3]);
    printf("t, each element: ");
    printf("%c %c %c %c\n", t[0], t[1], t[2], t[3]);
    printf("t, as C-String: %s\n", t);

    printf("\nNext... assigning ");
    printf("to each element in the arrays...\n\n");
```

```
// Assign to each element in the n array:  
n[0] = 15;  
n[1] = 25;  
n[2] = 35;  
n[3] = 45;  
  
// Assign to each element in the t array:  
t[0] = 'S';  
t[1] = 'D';  
t[2] = 'H';  
t[3] = '\0';  
  
// Print out the arrays again:  
printf("n, each element: ");  
printf("%d %d %d %d\n", n[0], n[1], n[2], n[3]);  
printf("t, each element: ");  
printf("%c %c %c %c\n", t[0], t[1], t[2], t[3]);  
  
printf("t, as C-String: %s\n", t);  
  
return 0;  
}
```

3.2.10 Example 10

This example initiates two char arrays without defined elements, asks the user to input the values for each array, with a maximum of 9 chars for the first and 4 to the second. The first array gets all the values in one input and the second array gets the values one symbol at a time.

```
C:\Windows\system32\cmd.exe
Username (9 chars max!): Paludo
Enter your PIN (4 symbols), one symbol at a time...
First symbol: 4
Second symbol: 1
Third symbol: 3
Fourth symbol: 9

Username is: Paludo
PIN is: 4139
ASCII for PIN: 52, 49, 51, 57
Pressione qualquer tecla para continuar. . . -
```

Figure 66: Using char Arrays.

The written code is:

```
#include <stdio.h>

int main()
{
    char username[10];
    char pin[5];

    printf("Username (9 chars max!): ");
    scanf("%9s", username);
```

```
int index_count = 0;
printf("\n\nEnter your PIN (4 symbols),");
printf(" one symbol at a time...\n\n");
printf("First symbol: ");
scanf(" %c", &pin[index_count]);
++index_count;

printf("Second symbol: ");
scanf(" %c", &pin[index_count]);
++index_count;

printf("Third symbol: ");
scanf(" %c", &pin[index_count]);
++index_count;

printf("Fourth symbol: ");
scanf(" %c", &pin[index_count]);
++index_count;

pin[index_count] = '\0';

printf("\n\n");

printf("Username is: %s\n\n", username);
printf("PIN is: %s\n\n", pin);

printf("ASCII for PIN: ");
printf("%d, %d, ", pin[0], pin[1]);
printf("%d, %d\n\n", pin[2], pin[3]);

return 0;
}
```

3.3 Exercises

3.3.1 Exercise 1

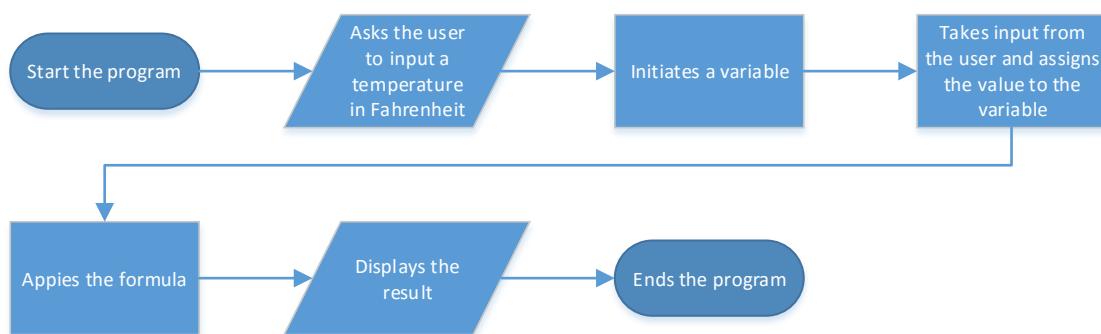
The first exercise asks the user to develop a program that takes from the user a temperature in Fahrenheit, converts it into Celsius and outputs the result. Example output:

```
+-----+
| Temperature Converter |
+-----+

Please enter a temperature in Fahrenheit: 65.3
Converting...
65.300003 degrees Fahrenheit is the same as...
18.500002 degrees Celsius.
```

Figure 67: Temperature Converter.

To design this program I draw the following flowchart:



The code written is this:

```
#include <stdio.h>
#include <math.h>

int main()
{
```

```
printf("-----+\n");
printf(" | Temperature Converter | \n");
printf("-----+\n");

printf("Please enter a temperature in Fahrenheit: ");

float temperature_fahrenheit;
scanf("%f", &temperature_fahrenheit);
printf("\n\n");

printf("Converting...");
printf("\n\n");

float temperature_celsius = (temperature_fahrenheit - 32) * 5 /
9;

printf("%f degrees Fahrenheit is the same as... \n",
temperature_fahrenheit);
printf("%lf degrees Celsius.", temperature_celsius);

printf("\n\n");
return 0;
}
```

This code gives the following output:

```
C:\Windows\system32\cmd.exe
-----
| Temperature Converter |
-----
Please enter a temperature in Fahrenheit: 38,49

Converting...
38.000000 degrees Fahrenheit is the same as...
3.333333 degrees Celsius.

Pressione qualquer tecla para continuar. . .
```

Figure 68: Temperature Converter Output.

The program initiates a float variable, asks the user to input a temperature in Fahrenheit, calculates the temperature in Celsius and prints both of the temperatures.

14885857

Reporting Journal #4

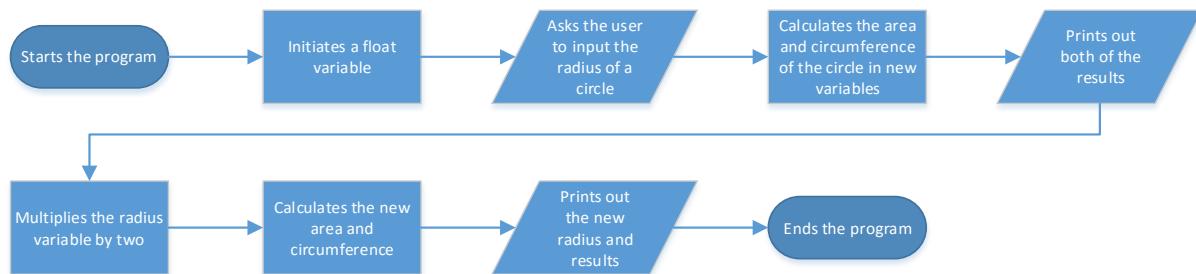
3.3.2 Exercise 2

This exercise asks the programmer to write a program that asks the user to input the radius of a circle as a real number, calculates and outputs the area and circumference of the circle and repeats this process with a circle with twice the radius that the user input. The program must have this display:

```
Circle Calculator:  
*****  
  
Please enter the radius of a circle: 5.5  
  
Calculating...  
  
A circle with radius 5.500000 has:  
  
- An area of: 95.033104  
- A circumference of: 34.557491  
  
Also, did you know a circle with radius 11.000000 has:  
  
- An area of: 380.132416  
- A circumference of: 69.114983
```

Figure 69: Circle Calculator.

To design this program I draw the following flowchart:



Before writing this flowchart, I wrote the following code:

```

#include <stdio.h>
#include <math.h>

int main()
{
    printf("Circle Calculator:\n");
    printf("*****\n");
    printf("\n\n");

    float radius;
    printf("Please enter the radius of a circle: ");
    scanf("%f", &radius);
    printf("\n\n");

    printf("Calculating... ");
    printf("\n\n");

    float area = 3.14159f * pow(radius, 2);
    float circumference = 2 * 3.14159f * radius;

    printf("A circle with radius %f has:", radius);
    printf("\n\n");
    printf(" -An area of: %f\n", area);
    printf(" -A circumference of: %f", circumference);
    printf("\n\n");

    radius = 2 * radius;
    area = 2 * area;
    circumference = 2 * circumference;
}
  
```

```

    printf("Also, did you know a circle with radius %f has:", radius);
    printf("\n\n");
    printf("    -An area of: %f\n", area);
    printf("    -A circumference of: %f", circumference);

    printf("\n\n");
    return 0;
}

```

That outputs:

```

C:\Windows\system32\cmd.exe
Circle Calculator:
Please enter the radius of a circle: 5.13

Calculating...

A circle with radius 5.000000 has:
    -An area of: 78.539749
    -A circumference of: 31.415901

Also, did you know a circle with radius 10.000000 has:
    -An area of: 157.079498
    -A circumference of: 62.831802

Pressione qualquer tecla para continuar. . .

```

Figure 70: Circle Calculator wrong output.

This program does not output the right answer for the calculus of the area for a circle with double of the radius, although it works for the circumference. The code written after designing the flowchart is the following code:

```

#include <stdio.h>
#include <math.h>

int main()
{
    printf("Circle Calculator:\n");

```

```
printf("*****\n");
printf("\n\n");

float radius;
printf("Please enter the radius of a circle: ");
scanf("%f", &radius);
printf("\n\n");

printf("Calculating... ");
printf("\n\n");

float area = 3.14159f * pow(radius, 2);
float circumference = 2 * 3.14159f * radius;

printf("A circle with radius %f has:", radius);
printf("\n\n");
printf("    -An area of: %f\n", area);
printf("    -A circumference of: %f", circumference);
printf("\n\n");

radius = 2 * radius;
area = 3.14159f * pow(radius, 2);
circumference = 2 * 3.14159f * radius;

printf("Also, did you know a circle with radius %f has:", radius);
printf("\n\n");
printf("    -An area of: %f\n", area);
printf("    -A circumference of: %f", circumference);

printf("\n\n");
return 0;
}
```

This code outputs, now, the correct answer for the purposed exercise.

```
C:\Windows\system32\cmd.exe
Circle Calculator:
*****
Please enter the radius of a circle: 5.13

Calculating...

A circle with radius 5.000000 has:
    -An area of: 78.539749
    -A circumference of: 31.415901

Also, did you know a circle with radius 10.000000 has:
    -An area of: 314.158997
    -A circumference of: 62.831802

Pressione qualquer tecla para continuar. . . -
```

Figure 71: Circle Calculator correct output.

3.3.3 Exercise 3

This exercise asks the student to develop a program which asks the user to input a single uppercase letter, and then display some interesting facts about this letter related to the ASCII table. The facts are: the lowercase letter, the letter before, the letter after, and the number of the letter in the alphabet. With the following output:

```
Alphabet Facts!
^^^^^^^^^^^^^^^^^

Please input an uppercase letter: H

Some interesting facts:

1) The lowercase version of the letter is 'h'.

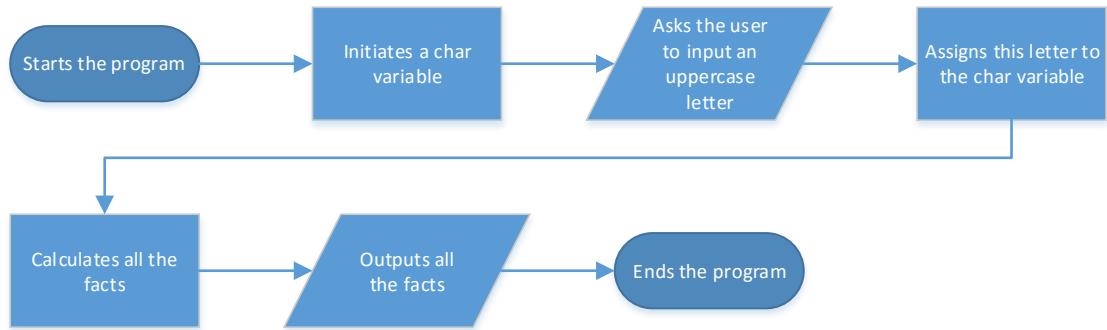
2) The letter 'G' comes before 'H' in the alphabet.

3) The letter 'I' comes after 'H' in the alphabet.

4) H is letter number 8 in the alphabet!
```

Figure 72: Alphabet Facts.

Before drawing the flowchart for the program, I was really confused about what to do, I was thinking about using multiple repeating and decision structures, such as for, int, do...while, and switch case, for example, but I was not sure about how to do it, then I decided to draw the flowchart looking at the ASCII table, and I noticed a pattern which can be useful to write this code.



Follows the written code:

```

#include <stdio.h>

int main()
{
    printf("Alphabet Facts!\n");
    printf("^^^^^^^^^^^^^^^^^");
    printf("\n\n");

    char letter;
    printf("Please input an uppercase letter: ");
    scanf("%c", &letter);
    printf("\n\n");

    int lowercase = letter + 32;
    int letter_before = letter - 1;
    int letter_after = letter + 1;
    int position = letter - 64;

    printf("Some interesting facts:");
    printf("\n\n");

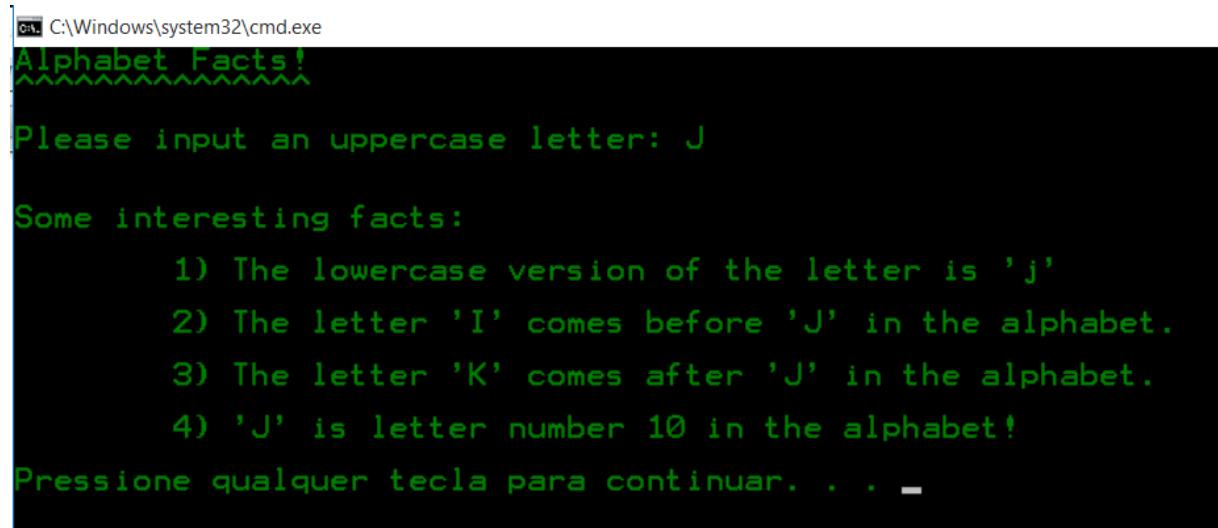
    printf(" 1) The lowercase version of the letter is '%c',\n",
lowercase);
    printf("\n\n");

    printf(" 2) The letter '%c' comes before '%c' in the\n",
alphabet., letter_before, letter);
    printf("\n\n");

    printf(" 3) The letter '%c' comes after '%c' in the\n",
alphabet., letter_after, letter);
    printf("\n\n");
  
```

```
    printf(" 4) '%c' is letter number %d in the alphabet!",  
letter, position);  
  
    printf("\n\n");  
    return 0;  
}
```

We can use int variables to calculate all the facts due to the fact that all the variables in the ASCII table can be represented by an integer number. The lowercase letter is always 32 positions after the lowercase letter, then to print the lowercase letter we just add 32 to the char variable and prints the result as a char. The letter before, logically, is just one position before, and the letter after, is one position after. The uppercase alphabet starts in the position 65 in the ASCII table, to calculate a uppercase letter's position in the alphabet we just need to subtract 64 from the char variable. The program outputs this window:



```
C:\Windows\system32\cmd.exe  
Alphabet Facts!  
Please input an uppercase letter: J  
  
Some interesting facts:  
 1) The lowercase version of the letter is 'j'  
 2) The letter 'I' comes before 'J' in the alphabet.  
 3) The letter 'K' comes after 'J' in the alphabet.  
 4) 'J' is letter number 10 in the alphabet!  
  
Pressione qualquer tecla para continuar. . . -
```

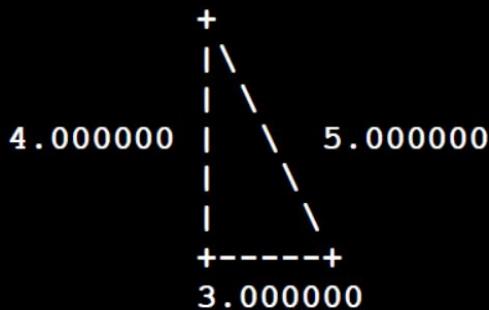
Figure 73: Alphabet Facts output.

3.3.4 Exercise 4

This program asks the user to input two sides of a right angle triangle, the adjacent and opposite sides, calculate the third size of this triangle, the hypotenuse, using the Pythagorean Theorem and prints the result in an ASCII drawing triangle, and inform the user that the right-angle triangle is not drawn to scale, with this output:

```
Enter the length of the adjacent side: 3
Enter the length of the opposite side: 4
```

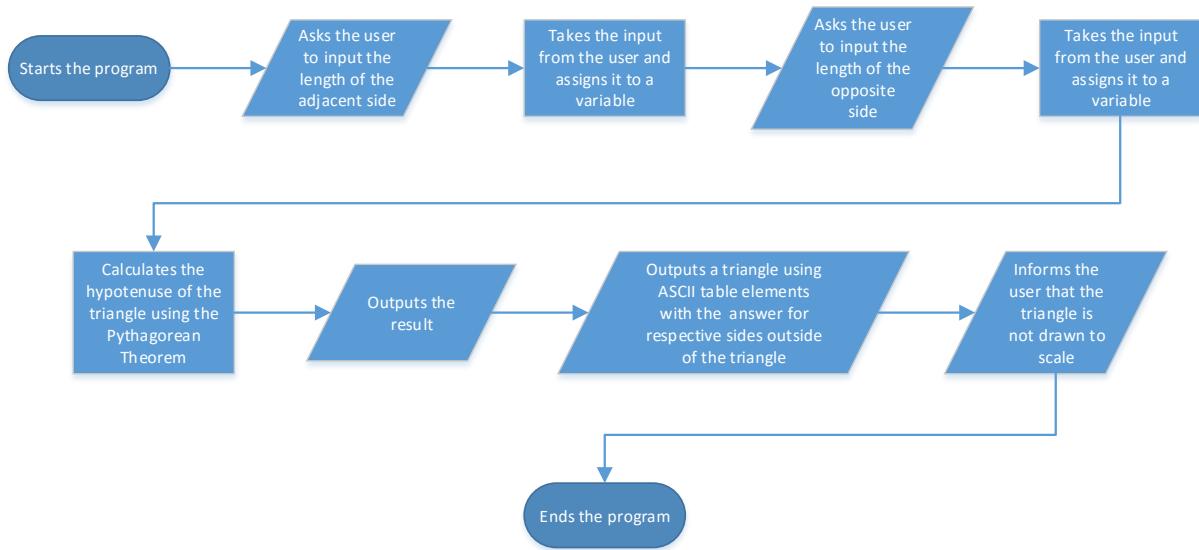
```
The hypotenuse is 5.000000 in length.
```



```
Note: Right-angle triangle is not drawn to scale!
```

Figure 74: Pythagorean Theorem.

To design this program I draw the following flowchart:



Using this flowchart I wrote the following code:

```

#include <stdio.h>
#include <math.h>

int main()
{
    float adjacent;
    printf("Enter the length of the adjacent side: ");
    scanf("%f", &adjacent);

    float opposite;
    printf("Enter the length of the opposite side: ");
    scanf("%f", &opposite);

    float hypotenuse = pow(((pow(adjacent, 2)) + (pow(opposite, 2))), (1.0 / 2));
    printf("The hypotenuse is %f in length.", hypotenuse);
    printf("\n\n");

    printf("      +\n");
    printf("      |\n");
    printf("      | \n");
  
```

```
    printf("%f  |  \\  %f\n", adjacent, opposite);
    printf("           |  \\\n");
    printf("           |  \\\n");
    printf("           +---+\n");
    printf("           %f\n", hypotenuse);
    printf("\n\n");

    printf("Note: Right-angle triangle is not drawn to scale!");

    printf("\n\n");
    return 0;
}
```

The program outputs the following window:

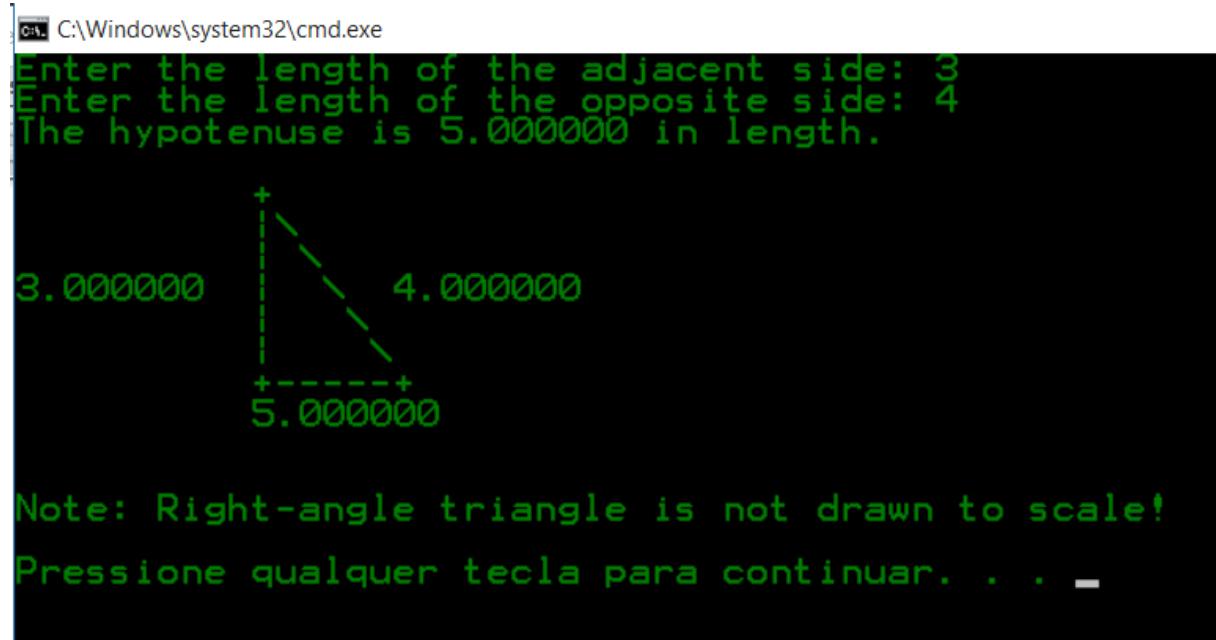


Figure 75: Pythagorean Theorem Output.

3.3.5 Exercise 5

This exercise asks the programmer write a program that rolls two twenty-sided dices (D20), and print these values, with this output:

```
D20 Dice Roller:  
=====
```

The first die rolls the value: 18

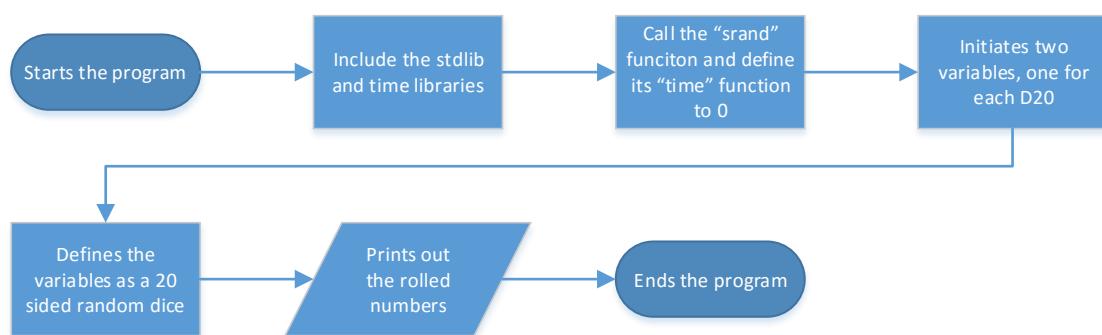
Then...

The second die rolls the value: 9

Figure 76: D20 Dice Roller.

To write this program, I had to look again at the lecture notes and laboratory examples.

After this, I draw the following flowchart:



Using this flowchart I wrote the following code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
```

```
    srand(time(0));

    int d20_1 = (rand() % 20) + 1;
    int d20_2 = (rand() % 20) + 1;

    printf("D20 Dice Roller:\n");
    printf("=====\n");
    printf("\n\n");

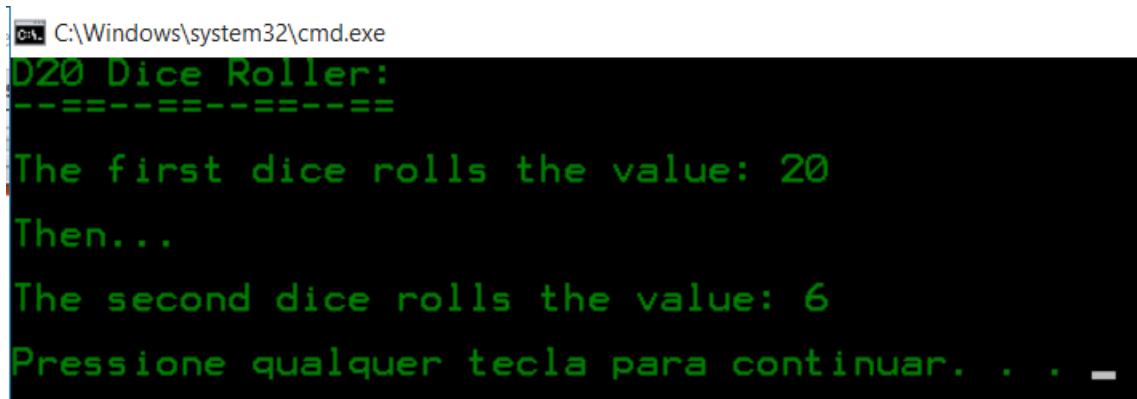
    printf("The first dice rolls the value: %d", d20_1);

    printf("\n\n");
    printf("Then...");
    printf("\n\n");

    printf("The second dice rolls the value: %d", d20_2);

    printf("\n\n");
    return 0;
}
```

This program outputs the following window:



The screenshot shows a Windows command prompt window titled 'cmd' with the path 'C:\Windows\system32\cmd.exe'. The window displays the output of the dice roller program. It starts with the title 'D20 Dice Roller:' followed by a separator line '====='. Then it prints 'The first dice rolls the value: 20', 'Then...', 'The second dice rolls the value: 6', and finally 'Pressione qualquer tecla para continuar. . .'. The text is in white on a black background.

Figure 77: D20 Dice Roller Output.

When the dice is set to a 20 sided dice, in the end of the function we have to add 1, because the program usually starts rolling at 0.

3.3.6 Exercise 6

This exercise asks the student to develop a program that takes 5 real number inputs from the user, stores these numbers in a float array, prints all the elements of this array and calculates the average of these numbers.

```
Please enter five real numbers:
```

```
First number: 62.5
Second number: 31.2
Third number: 3.5
Fourth number: 21.7
Fifth number: 12.9
```

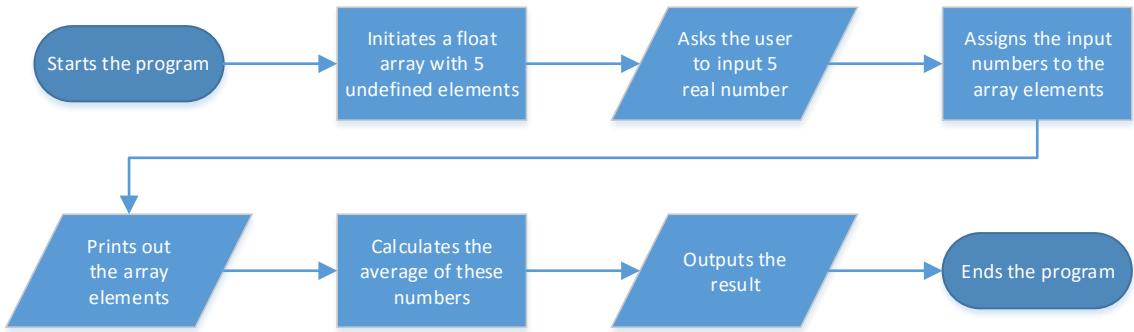
```
The user entered:
```

```
62.500000 then,
31.200001 then,
3.500000 then,
21.700001 then,
12.900000
```

```
The average of these five numbers is: 26.359997
```

Figure 78: Average of Five using an Array.

To write this program, I draw the following flowchart:



Using these flowchart, the “Average of Five” exercise, and the array examples, I wrote the following code to test the program:

```

#include <stdio.h>

int main()
{
    float average[5];

    printf("Please enter five real numbers:");
    printf("\n\n");

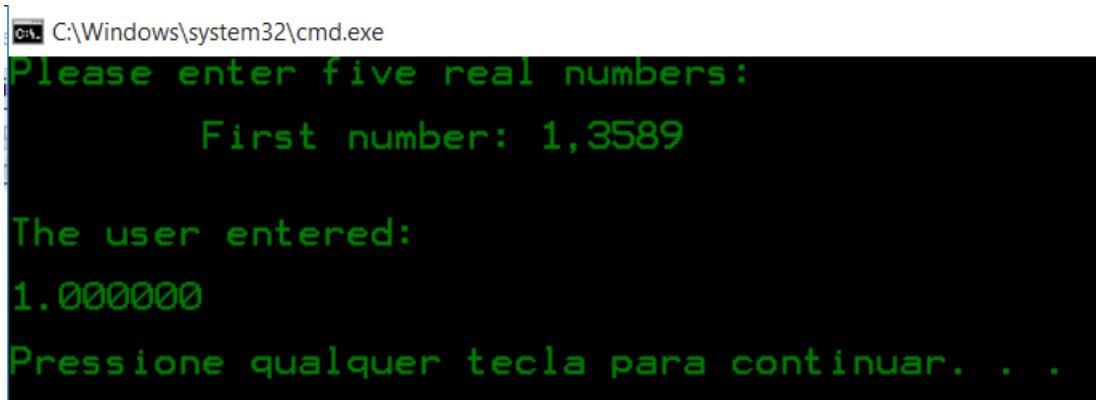
    printf("First number: ");
    scanf("%f", &average[1]);
    printf("\n\n");

    printf("The user entered:");
    printf("\n\n");

    printf("%f", average[1]);

    printf("\n\n");
    return 0;
}
  
```

But this program floors all the array elements, as shown in the image:



The screenshot shows a command-line interface window titled 'C:\Windows\system32\cmd.exe'. It displays the following text:

```
Please enter five real numbers:  
First number: 1,3589  
  
The user entered:  
1.000000  
Pressione qualquer tecla para continuar. . .
```

Figure 79: Wrong input.

After a lot of changes, running and debugging the problem, I had to take a break, and when I came back, I realised that the problem was that I was using a coma instead of a point. After overcoming this problem, I wrote the following code:

```
#include <stdio.h>  
  
int main()  
{  
    int size = 5;  
    float array[5];  
  
    printf("Please enter five real numbers:");  
    printf("\n\n");  
  
    printf(" First number: ");  
    scanf("%f", &array[1]);  
    printf("\n Second number: ");  
    scanf("%f", &array[2]);  
    printf("\n Third number: ");  
    scanf("%f", &array[3]);  
    printf("\n Fourth number: ");  
    scanf("%f", &array[4]);  
    printf("\n Fifth number: ");  
    scanf("%f", &array[5]);  
    printf("\n\n");  
  
    printf("The user entered:");  
    printf("\n\n");
```

```

printf("    %f then,\n", array[1]);
printf("    %f then,\n", array[2]);
printf("    %f then,\n", array[3]);
printf("    %f then,\n", array[4]);
printf("    %f", array[5]);
printf("\n\n");

float average;

for (size = 4; size > 0; size--)
{
    average += array[size];
}

average /= size;
printf("The average of these five numbers is: %f", average);

printf("\n\n");
return 0;
}

```

This code runs, but it outputs another error:

C:\Windows\system32\cmd.exe

Please enter five real numbers:

First number: 1.3215
 Second number: 2.3265
 Third number: 3.6589
 Fourth number: 4.5619
 Fifth number: 5.329

The user entered:

1.321500 then,
 2.326500 then,
 3.658900 then,
 4.561900 then,
 5.329000

Microsoft Visual C++ Runtime Library

Debug Error!

Program: ...ents\Visual Studio 2015\Projects\W03 Lab\Debug\Exercise 6.exe
 Module: ...ents\Visual Studio 2015\Projects\W03 Lab\Debug\Exercise 6.exe
 File:
 Run-Time Check Failure #3 - T
 (Press Retry to debug the application)

Anular Repetir Ignorar

Figure 80: Another error.

I thought the problem was in the for function, then I tried commenting this part and running the program again, but the problem persisted, then I tried debugging the program and looking at the variables.

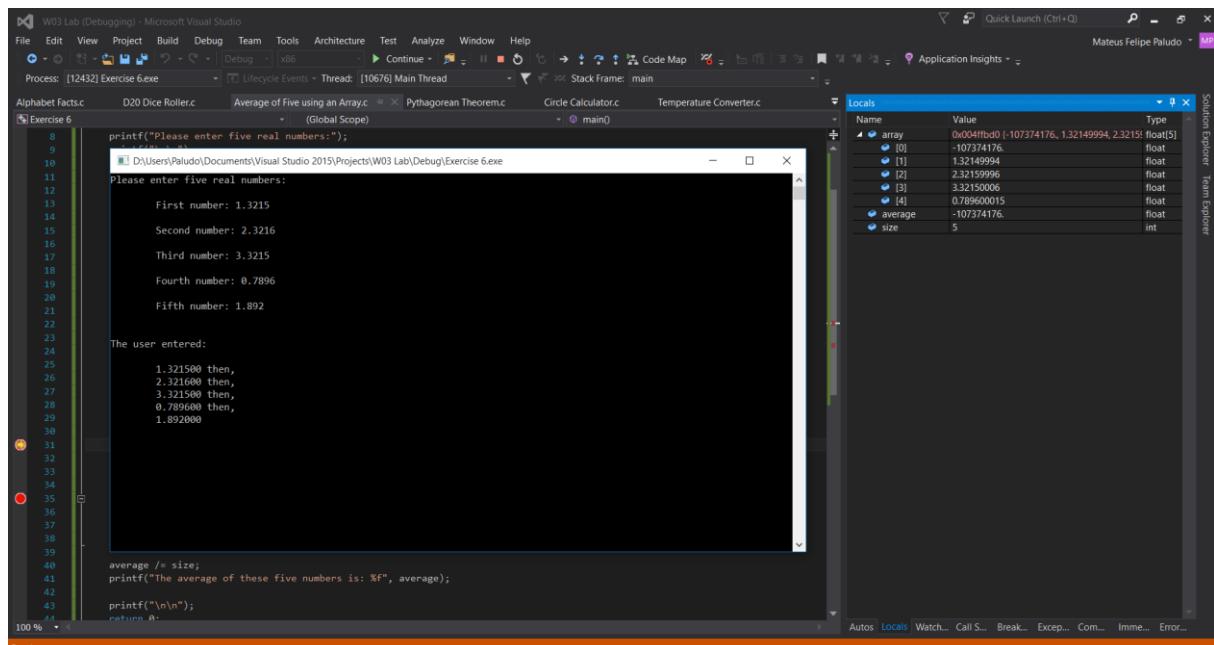


Figure 81: Debugging.

While debugging I noticed that I forgot to set the array elements from 0 to 4, I used 1 to 5 instead. When I compiled and ran the program, the problem persisted, and I figured out, that it had another problem, I was subtracting the “size” variable to 0 and using it to do a division, but the problem persisted. Visual Studio Debugger reported that the “average” was being used without being initialized, then I figured out that I had to declare the variable before using it inside the for method. The written code is this:

```
#include <stdio.h>
```

```
int main()
{
    int size = 5;
    float array[5];

    printf("Please enter five real numbers:");
    printf("\n\n");

    printf("    First number: ");
    scanf("%f", &array[0]);
    printf("\n Second number: ");
    scanf("%f", &array[1]);
    printf("\n Third number: ");
    scanf("%f", &array[2]);
    printf("\n Fourth number: ");
    scanf("%f", &array[3]);
    printf("\n Fifth number: ");
    scanf("%f", &array[4]);
    printf("\n\n");

    printf("The user entered:");
    printf("\n\n");

    printf("    %f then,\n", array[0]);
    printf("    %f then,\n", array[1]);
    printf("    %f then,\n", array[2]);
    printf("    %f then,\n", array[3]);
    printf("    %f", array[4]);
    printf("\n\n");

    float average = 0;

    for (size = 4; size > 0; size--)
    {
        average += array[size];
    }

    average /= 5;
    printf("The average of these five numbers is: %f", average);

    printf("\n\n");
    return 0;
}
```

The output is this:

```
C:\Windows\system32\cmd.exe
Please enter five real numbers:
    First number: 1.3215
    Second number: 3.3215
    Third number: 2.3215
    Fourth number: 0.7851
    Fifth number: 1.3215

The user entered:
    1.321500 then,
    3.321500 then,
    2.321500 then,
    0.785100 then,
    1.321500

The average of these five numbers is: 1.549920
Pressione qualquer tecla para continuar. . . -
```

Figure 82: Average of Five using an Array wrong calculus.

But I decided to write another code to test if the answer is correct.

```
#include <stdio.h>

int main()
{
    int size = 5;
    float array[5];
```

```
printf("Please enter five real numbers:");
printf("\n\n");

printf("    First number: ");
scanf("%f", &array[0]);
printf("\n Second number: ");
scanf("%f", &array[1]);
printf("\n Third number: ");
scanf("%f", &array[2]);
printf("\n Fourth number: ");
scanf("%f", &array[3]);
printf("\n Fifth number: ");
scanf("%f", &array[4]);
printf("\n\n");

printf("The user entered:");
printf("\n\n");

printf("    %f then,\n", array[0]);
printf("    %f then,\n", array[1]);
printf("    %f then,\n", array[2]);
printf("    %f then,\n", array[3]);
printf("    %f", array[4]);
printf("\n\n");

float average = 0;

for (size = 4; size > 0; size--)
{
    average += array[size];
}

float test = (array[0] + array[1] + array[2] + array[3] +
array[4]) / 5;
average /= 5;
printf("The average of these five numbers is: %f", average);
printf("\nTest %f", test);

printf("\n\n");
return 0;
}
```

This code outputs:

```
C:\Windows\system32\cmd.exe
Please enter five real numbers:
    First number: 1.3215
    Second number: 2.3628
    Third number: 0.7956
    Fourth number: 3.3629
    Fifth number: 0.9878

The user entered:
    1.321500 then,
    2.362800 then,
    0.795600 then,
    3.362900 then,
    0.987800

The average of these five numbers is: 1.501820
Test 1.766120

Pressione qualquer tecla para continuar. . .
```

Figure 83: Explaining the error.

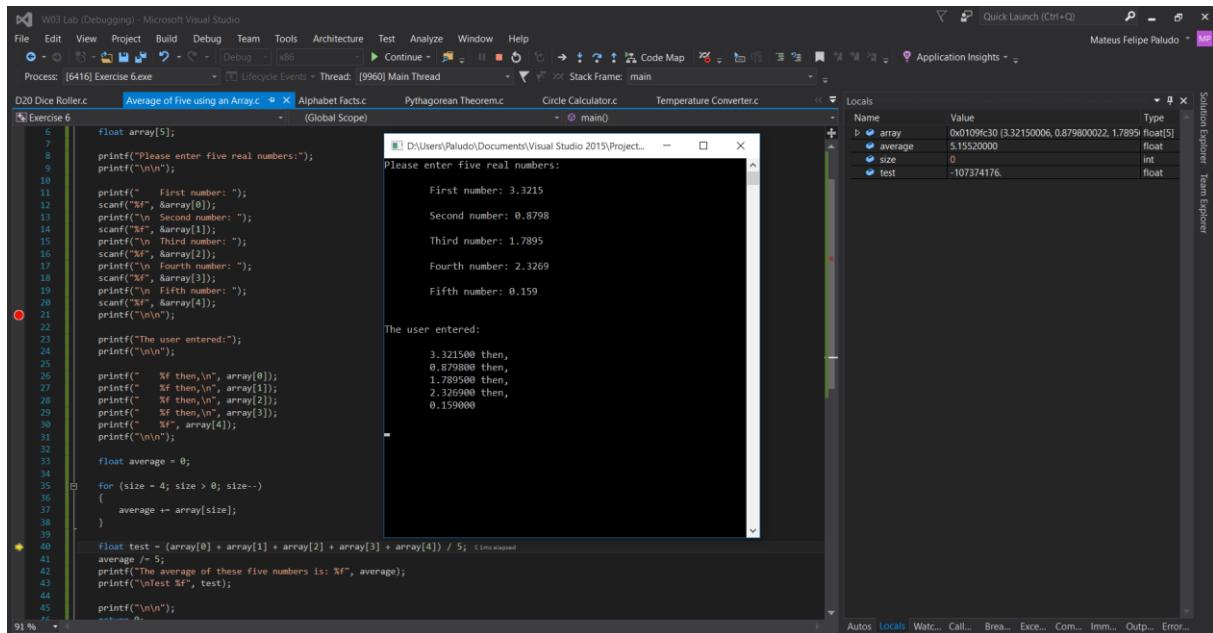


Figure 84: Debugging screen.

While debugging, I noticed that the problem was that it was just adding four of the array elements, because it starts in the element four, and just do it until the element one, the element zero stops the program, so I just had to make a little change in the program, follows the final code:

```
#include <stdio.h>

int main()
{
    int size = 5;
    float array[5];

    printf("Please enter five real numbers:");
    printf("\n\n");

    printf(" First number: ");
    scanf("%f", &array[0]);
    printf("\n Second number: ");
    scanf("%f", &array[1]);
    printf("\n Third number: ");
    scanf("%f", &array[2]);
    printf("\n Fourth number: ");
    scanf("%f", &array[3]);
    printf("\n Fifth number: ");
    scanf("%f", &array[4]);
    printf("\n\n");

    float average = 0;
    for (size = 4; size > 0; size--)
    {
        average += array[size];
    }

    float test = (array[0] + array[1] + array[2] + array[3] + array[4]) / 5; // line changed
    average /= 5;
    printf("The average of these five numbers is: %f", average);
    printf("\nTest %f", test);

    printf("\n\n");
}
```

```
scanf("%f", &array[2]);
printf("\n Fourth number: ");
scanf("%f", &array[3]);
printf("\n Fifth number: ");
scanf("%f", &array[4]);
printf("\n\n");

printf("The user entered:");
printf("\n\n");

printf("    %f then,\n", array[0]);
printf("    %f then,\n", array[1]);
printf("    %f then,\n", array[2]);
printf("    %f then,\n", array[3]);
printf("    %f", array[4]);
printf("\n\n");

float average = 0;

for (size = 4; size >= 0; size--)
{
    average += array[size];
}

average /= 5;
printf("The average of these five numbers is: %f", average);

printf("\n\n");
return 0;
}
```

This program has this output:

```
C:\Windows\system32\cmd.exe
Please enter five real numbers:
First number: 1.3256
Second number: 0.8796
Third number: 3.3268
Fourth number: 2.2178
Fifth number: 0.329

The user entered:
1.325600 then,
0.879600 then,
3.326800 then,
2.217800 then,
0.329000

The average of these five numbers is: 1.615760
Pressione qualquer tecla para continuar. . .
```

Figure 85: Average of Five using an Array Output.

To write this program I had to do a lot of research in books. The for structure is very similar to the while structure. “You typically use the for loop to execute a block of statements a given number of times ... The for loop operation is controlled by what appears between the parentheses that follow the keyword for .” (Horton, 2013, p. 137). As stated by Sempf, Chuck and Davis, “When the for loop is encountered, the program first executes the initExpression expression and then executes the condition. If the condition expression is true, the program executes the body of the loop, which is surrounded by the braces immediately following the for command. When the program reaches the closed brace, control passes to incrementExpression and then back to condition, where the next pass through the loop

begins." (2010, p. 104). "The for repetition statement handles all the details of counter-controlled repetition", as wrote Deitel and Deitel (2010, p. 100). As written in Perry (1992) "The for loop enables you to repeat sections of your program for a specific number of times. Unlike the while and do-while loops, the for loop is a determinate loop. This means when you write your program you can usually determine how many times the loop iterates. The while and do-while loops repeat only until a condition is met. The for loop does this and more: It continues looping until a count (or countdown) is reached. After the final for loop count is reached, execution continues with the next statement, in sequence. " (p. 273)

3.3.7 Exercise 7

This exercise asks the programmer to write a program that takes 3 outputs from the user, creates an array with this numbers, asks more three numbers, creates another array, sums, subtracts and multiplies both arrays and prints out the result.

```
First Array Setup...
Please enter the 1st value: 6
Please enter the 2nd value: 5
Please enter the 3rd value: 1

First array is: { 6, 5, 1 }

Second Array Setup...
Please enter the 1st value: 4
Please enter the 2nd value: 1
Please enter the 3rd value: 8

Second array is: { 4, 1, 8 }

C...a...l...c...u...l...a...t...i...n...g...

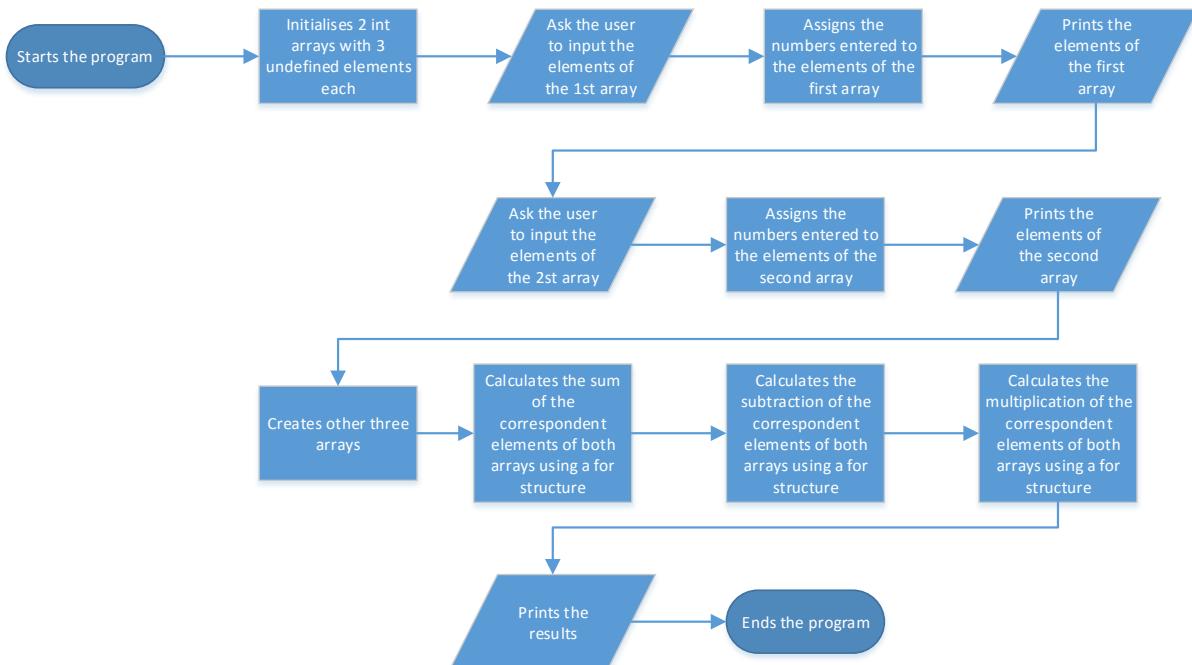
Adding corresponding elements: { 10, 6, 9 }

Subtracting corresponding elements: { 2, 4, -7 }

Multiplying corresponding elements: { 24, 5, 8 }
```

Figure 86: Three Table Maths.

To write this code, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

int main()
{
    int array1[3];
    int array2[3];

    printf("First Array Setup...\n");
    printf("\n\n");

    printf("    Please enter the 1st value: ");
    scanf("%d", &array1[0]);
    printf("    Please enter the 2nd value: ");
    scanf("%d", &array1[1]);
    printf("    Please enter the 3rd value: ");
    scanf("%d", &array1[2]);
    printf("\n\n");

    printf("First array is: { %d, %d, %d }", array1[0], array1[1],
array1[2]);
  
```

```
printf("\n\n");

printf("Second Array Setup...") ;
printf("\n\n");

printf("    Please enter the 1st value: ");
scanf("%d", array2[0]);
printf("    Please enter the 2nd value: ");
scanf("%d", array2[1]);
printf("    Please enter the 3rd value: ");
scanf("%d", array2[2]);
printf("\n\n");

printf("Second array is: { %d, %d, %d }", array2[0], array2[1],
array2[2]);
printf("\n\n");

printf("    C..a..l..c..u..l..a..t..i..n..g..");
printf("\n\n");

int sum[3];
int subtract[3];
int multiply[3];

for (int i = 0; i < 3; i++)
{
    sum[i] = array1[i] + array2[i];
    subtract[i] = array1[i] - array2[i];
    multiply[i] = array1[i] * array2[i];
}

printf("Adding corresponding elements: { %d, %d, %d }", sum[0],
sum[1], sum[2]);
printf("\n\n");

printf("Subtracting corresponding elements: { %d, %d, %d }",
subtract[0], subtract[1], subtract[2]);
printf("\n\n");

printf("Multiplying corresponding elements: { %d, %d, %d }",
multiply[0], multiply[1], multiply[2]);
printf("\n\n");

return 0;
}
```

But I forgot to put the “&” symbol in the scanf function, after fixing this issue, I wrote this code:

```
#include <stdio.h>

int main()
{
    int array1[3];
    int array2[3];

    printf("First Array Setup...") ;
    printf("\n\n");

    printf("    Please enter the 1st value: ") ;
    scanf("%d", &array1[0]) ;
    printf("    Please enter the 2nd value: ") ;
    scanf("%d", &array1[1]) ;
    printf("    Please enter the 3rd value: ") ;
    scanf("%d", &array1[2]) ;
    printf("\n\n");

    printf("First array is: { %d, %d, %d }", array1[0], array1[1],
array1[2]) ;
    printf("\n\n");

    printf("Second Array Setup...") ;
    printf("\n\n");

    printf("    Please enter the 1st value: ") ;
    scanf("%d", &array2[0]) ;
    printf("    Please enter the 2nd value: ") ;
    scanf("%d", &array2[1]) ;
    printf("    Please enter the 3rd value: ") ;
    scanf("%d", &array2[2]) ;
    printf("\n\n");

    printf("Second array is: { %d, %d, %d }", array2[0], array2[1],
array2[2]) ;
    printf("\n\n");

    printf("    C..a..l..c..u..l..a..t..i..n..g..") ;
    printf("\n\n");

    int sum[3];
    int subtract[3];
```

```
int multiply[3];

for (int i = 0; i < 3; i++)
{
    sum[i] = array1[i] + array2[i];
    subtract[i] = array1[i] - array2[i];
    multiply[i] = array1[i] * array2[i];
}

printf("Adding corresponding elements: { %d, %d, %d }", sum[0],
sum[1], sum[2]);
printf("\n\n");

printf("Subtracting corresponding elements: { %d, %d, %d }",
subtract[0], subtract[1], subtract[2]);
printf("\n\n");

printf("Multiplying corresponding elements: { %d, %d, %d }",
multiply[0], multiply[1], multiply[2]);
printf("\n\n");

return 0;
}
```

The program outputs the following window:

```
C:\Windows\system32\cmd.exe
First Array Setup...
Please enter the 1st value: 3
Please enter the 2nd value: 7
Please enter the 3rd value: 9

First array is: { 3, 7, 9 }

Second Array Setup...
Please enter the 1st value: 2
Please enter the 2nd value: 7
Please enter the 3rd value: 8

Second array is: { 2, 7, 8 }

C...a...l...c...u...l...a...t...i...n...g...

Adding corresponding elements: { 5, 14, 17 }
Subtracting corresponding elements: { 1, 0, 1 }
Multiplying corresponding elements: { 6, 49, 72 }
Pressione qualquer tecla para continuar. . . -
```

Figure 87: Three Tuple Maths Output.

To do the math operations with the arrays I used a for statement, this allowed me to write the program with less lines, it is more efficient and it helped me understanding the for structure.

3.3.8 Exercise 8

In this exercise I had to ask the user to input a real number, assign this input to a float variable, use a char array to take this value using sprintf. The program should have this output:

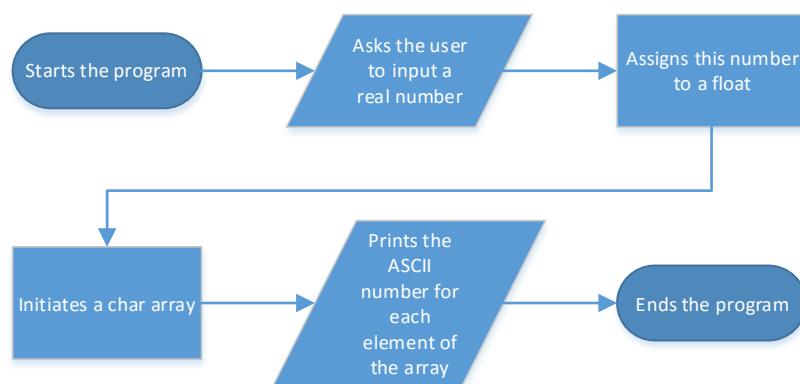
```
Enter a real number: 2.718

Converting float into ASCII...

Element 0 is: '2' which is ASCII 50
Element 1 is: '.' which is ASCII 46
Element 2 is: '7' which is ASCII 55
Element 3 is: '1' which is ASCII 49
Element 4 is: '8' which is ASCII 56
Element 5 is: '0' which is ASCII 48
Element 6 is: '0' which is ASCII 48
Element 7 is: '0' which is ASCII 48
```

Figure 88: float Input to ASCII char Array.

To develop this program, I draw the following flowchart:



With this flowchart, I wrote the following code:

The screenshot shows a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. It displays the following text:

```
Enter a real number: 1.321
Converting float into ASCII...
Element 0 is: '1' which is ASCII 49
Element 1 is: '.' which is ASCII 46
Element 2 is: '3' which is ASCII 51
Element 3 is: '2' which is ASCII 50
Element 4 is: '1' which is ASCII 49
Element 5 is: '0' which is ASCII 48
Element 6 is: '0' which is ASCII 48
```

Below the command prompt is a 'Microsoft Visual C++ Runtime Library' dialog box titled 'Debug Error!'. The dialog contains the following information:

Program: ...ents\Visual Studio 2015\Projects\W03 Lab\Debug\Exercise 8.exe
Module: ...ents\Visual Studio 2015\Projects\W03 Lab\Debug\Exercise 8.exe
File:
Run-Time Check Failure #2 - S
(Press Retry to debug the application)

At the bottom of the dialog are three buttons: 'Anular', 'Repetir', and 'Ignorar'.

Figure 89: float Input to ASCII char Array error.

The program runs without problems, but it returns an error. Debugging it, I found out that the program's error happens after the program ends:

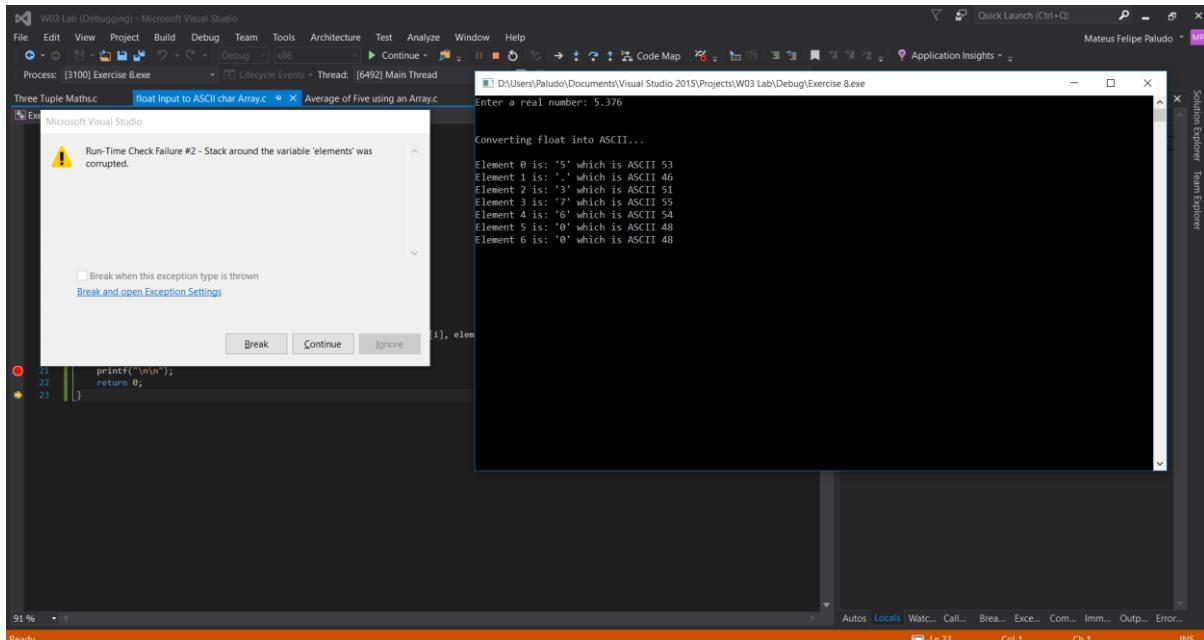


Figure 90: Debugging error.

I found out that the program was not closing the char array, so I put in the array one more slot than the for structure was going to use, here follows the final code:

```

#include <stdio.h>

int main()
{
    float input;
    printf("Enter a real number: ");
    scanf("%f", &input);
    printf("\n\n");

    printf("Converting float into ASCII... ");
    printf("\n\n");

    char elements[9];
    sprintf(elements, "%f", input);

    for (int i = 0; i < 8; i++)
    {
        printf("Element %d is: '%c' which is ASCII %d\n", i,
elements[i], elements[i]);
    }
}

```

```
    printf("\n\n");
    return 0;
}
```

This code outputs the following window:

The screenshot shows a command-line interface window titled 'C:\Windows\system32\cmd.exe'. The user has entered the command 'Enter a real number: 1.376'. The program then outputs the following text:
Converting float into ASCII...
Element 0 is: '1' which is ASCII 49
Element 1 is: '.' which is ASCII 46
Element 2 is: '3' which is ASCII 51
Element 3 is: '7' which is ASCII 55
Element 4 is: '6' which is ASCII 54
Element 5 is: '0' which is ASCII 48
Element 6 is: '0' which is ASCII 48
Element 7 is: '0' which is ASCII 48

Pressione qualquer tecla para continuar. . .

Figure 91: float Input to ASCII char Array Output.

3.3.9 Exercise 9

This exercise asks the programmer to write a program that asks the user to input a 8 bit binary number and gives the respective decimal number for it.

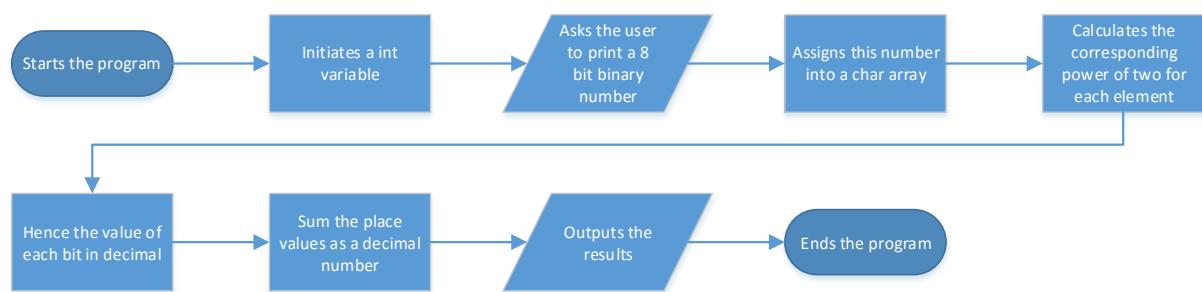
```
Input an 8 bit binary number: 11011011

((2 ^ 7) * 1) is 128
((2 ^ 6) * 1) is 64
((2 ^ 5) * 0) is 0
((2 ^ 4) * 1) is 16
((2 ^ 3) * 1) is 8
((2 ^ 2) * 0) is 0
((2 ^ 1) * 1) is 2
((2 ^ 0) * 1) is 1

-----
11011011 in Binary is 219 in Decimal.
-----
```

Figure 92: Binary to Decimal.

To write this code, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    int input;
    printf("Input an 8 bit binary number: ");
    scanf("%d", &input);
    printf("\n\n");

    char binary[9];
    sprintf(binary, "%d", input);

    int sum;
    int result;

    for (int i = 0; i < 8; i++)
    {
        result = (pow(2, i) * (binary[i]));
        sum += result;
        printf("((2 ^ %d) * %c) is %d\n", i, binary[i], result);
    }

    printf("\n");
    printf("-----\n");
    printf("%s in Binary is %d in Decimal", binary, sum);
    printf("-----\n");

    printf("\n\n");
    return 0;
}
```

But it returns the following error:

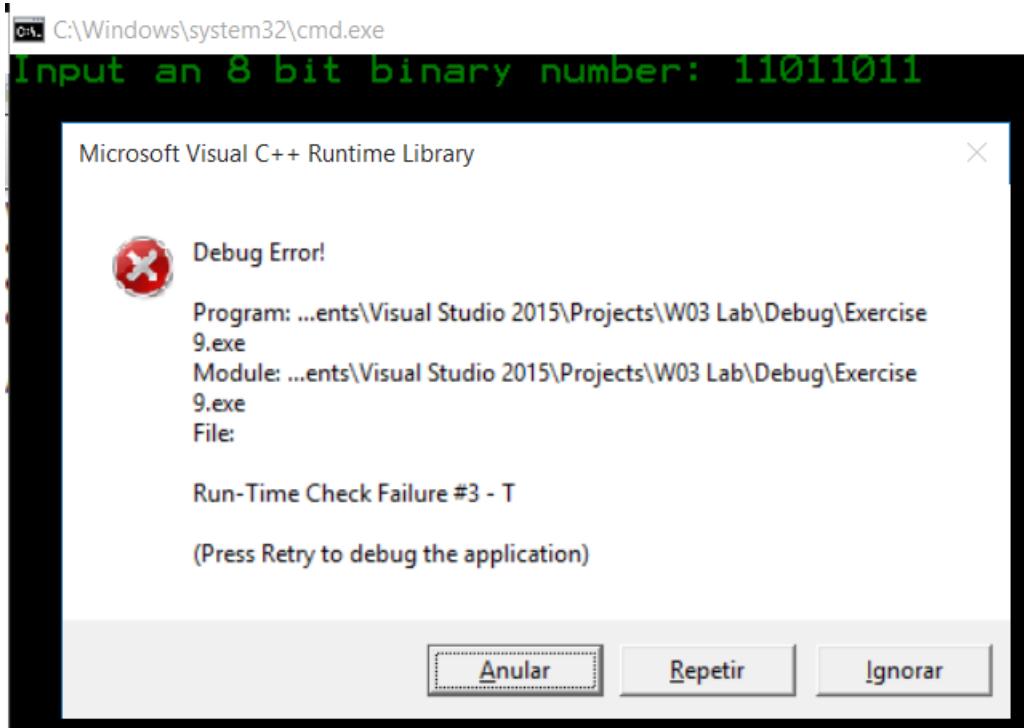


Figure 93: Output error.

I tried cleaning, building the solution, but the error persisted. After a long stop, I went back, tried to run, and the code ran without problems, but the calculus was wrong, follows the final version of the code:

```
#include <stdio.h>
#include <math.h>

int main()
{
    char binary[9];
    printf("Input an 8 bit binary number: ");
    scanf("%s", binary);
    printf("\n\n");

    int sum = 0;
    int result = 0;

    for (int i = 7; i >= 0; i--)
    {
```

```
        result = pow(2, i) * (binary[i] - 48);
        sum += result;
        printf("((2 ^ %d) * %c) is %d\n", i, binary[i], result);
    }

    printf("\n");
    printf("-----\n");
    printf("%s in Binary is %d in Decimal\n", binary, sum);
    printf("-----\n");

    printf("\n\n");
    return 0;
}
```

This program outputs:

The screenshot shows a Windows command prompt window titled 'cmd C:\Windows\system32\cmd.exe'. The user has inputted the binary number '11011011' and the program has output the following results:

```
Input an 8 bit binary number: 11011011

((2 ^ 7) * 1) is 128
((2 ^ 6) * 1) is 64
((2 ^ 5) * 0) is 0
((2 ^ 4) * 1) is 16
((2 ^ 3) * 1) is 8
((2 ^ 2) * 0) is 0
((2 ^ 1) * 1) is 2
((2 ^ 0) * 1) is 1

-----
11011011 in Binary is 219 in Decimal
-----

Pressione qualquer tecla para continuar. . .
```

Figure 94: Binary to Decimal Output.

4 Week Four

4.1 Lectures

In this week we learned how to use the if/else selection structure. This structure takes a condition, if this condition is true, the program takes an action, else, the program takes another action. If/else structures using flowcharts and pseudo codes were also taught.

The logic boolean operators “not”, “or” and “and” were also mentioned during the classes, their use is very intuitive, they can be understood by writing the pseudo code and looking how does the code works.

The “switch” selector has to be used with the “case” selector, each case selector will take a different action, and a “break” must be used in the end of each selector if the programmer wants to stop the program to keep going.

4.2 Exercises

4.2.1 Exercise 1

The exercise “I Can Program” asks the programmer to write a code that prints ‘I can program!’ on the screen. To write this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    printf("I can program!");
    printf("\n\n");
    return 0;
}
```

Which outputs:

```
C:\Windows\system32\cmd.exe
I can program!
Pressione qualquer tecla para continuar. . . -
```

Figure 95: I Can Program Output.

4.2.2 Exercise 2

This exercise asks the programmer to write a code that initializes two integer and one float variable, initialize them to 20, 32, 4, and 75, respectively, then print these variables. To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

int main()
{
    int x;
    int y;
    float z;

    x = 20;
    y = 32;
    z = 4.75f;

    printf("%d\n%d\n%f\n", x, y, z);
}
  
```

This program outputs:

A screenshot of a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
20
32
4.750000
Pressione qualquer tecla para continuar. . .

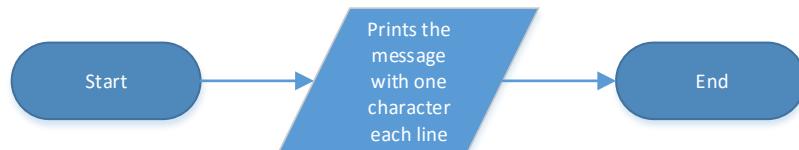
Figure 96: Basic Variable Usage Output.

14885857

Reporting Journal #4

4.2.3 Exercise 3

This exercise asks the student to write a program that outputs 'I can program!' vertically, to write this code, I draw the following flowchart.



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    printf("I\n \nc\na\nn\n \n p\nr\no\nng\nr\na\nm\n!");
    printf("\n\n");
    return 0;
}
```

This code outputs:

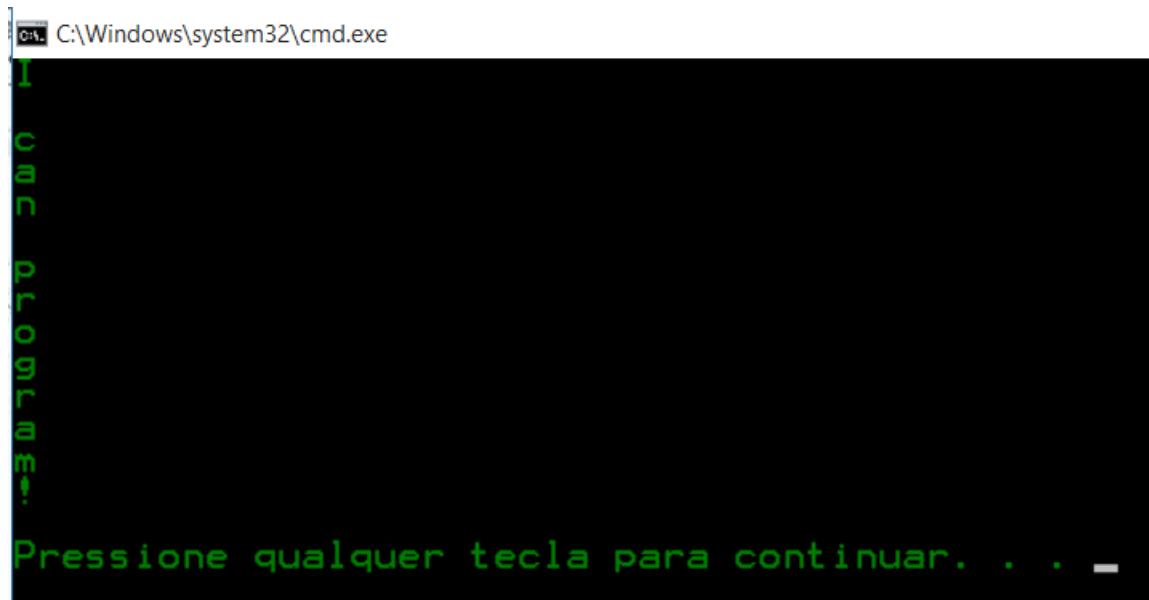
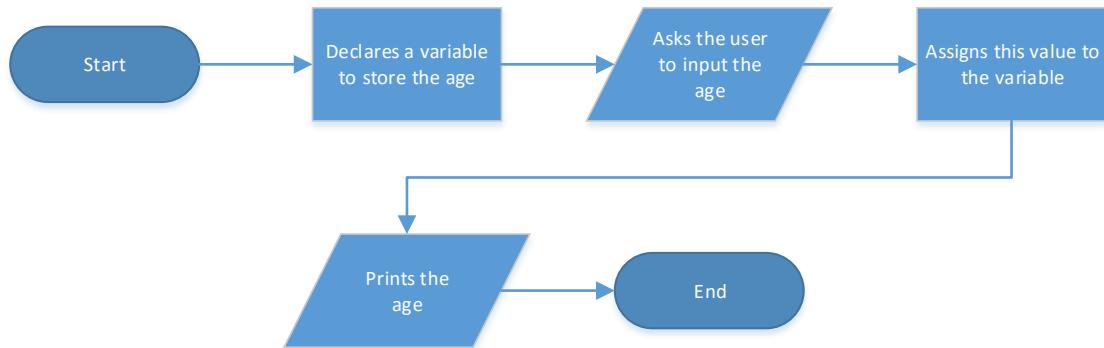


Figure 97: I Can Program Again Output.

4.2.4 Exercise 4

This program must ask the user to input their age and output this age. To write this program, I draw the following flowchart:



With this flowchart, I wrote the following code:

```

#include <stdio.h>

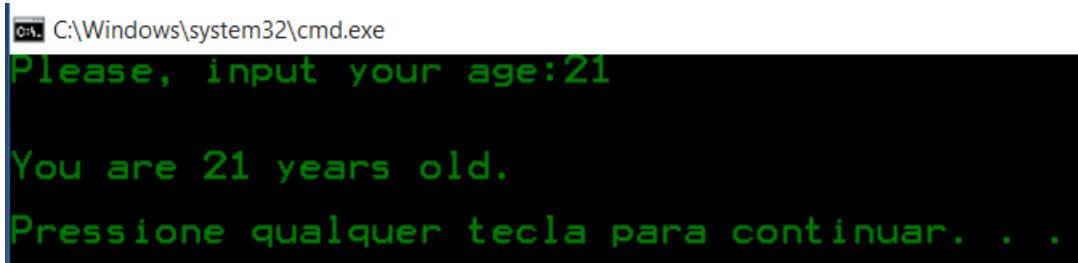
int main()
{
    int age;

    printf("Please, input your age:");
    scanf("%d", &age);
    printf("\n\n");

    printf("You are %d years old.", age);

    printf("\n\n");
    return 0;
}
  
```

This program outputs:



```
C:\Windows\system32\cmd.exe
Please, input your age:21
You are 21 years old.
Pressione qualquer tecla para continuar. . .
```

Figure 98: A Question of Age Output.

4.2.5 Exercise 5

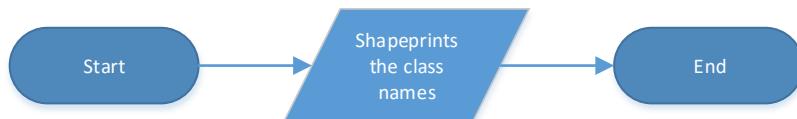
This exercise asks the student to write a code to display this pattern:

```
* P R O G R A M M I N G 1 *
C                           E
O                         N
M                         S
P                           E
5           5
0       0
0   1
*

```

Figure 99: Pattern Printing.

To write this program, I draw the following flowchart:



With this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    printf(" *PROGRAMMING1*\n");
    printf("C           E\n");
    printf(" O           N\n");
    printf("  M           S\n");
    printf("    P           E\n");
    printf("      5           5\n");
    printf("        0           0\n");
    printf("          0           1\n");
    printf("            *\n");

    printf("\n\n");
}
```

```
    return 0;  
}
```

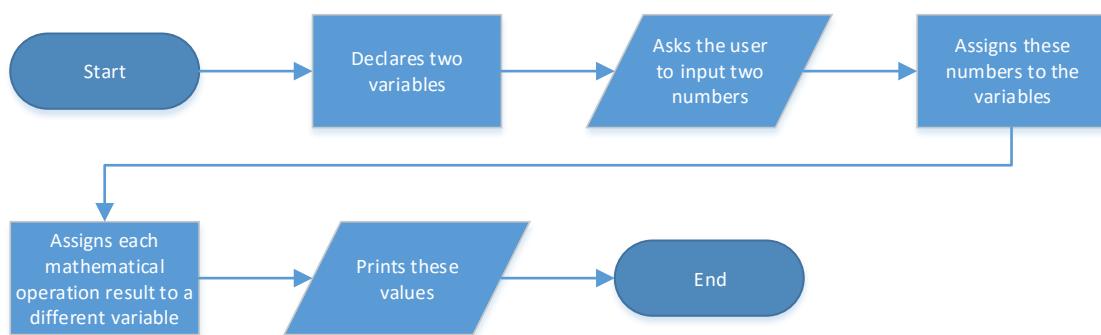
This program outputs:



Figure 100: Pattern Printing Output.

4.2.6 Exercise 6

This program must take two numbers as input, calculate the average, sum, and sum of the squares of these numbers and output the results. To write this program, I draw the following flowchart:



With this flowchart, I wrote the following code:

```

#include <stdio.h>
#include <math.h>

int main()
{
    float x;
    printf("Input the first number:");
    scanf("%f", &x);

    float y;
    printf("Input the second number:");
    scanf("%f", &y);
    printf("\n\n");

    float sum = x + y;
    printf("The sum is: %f\n", sum);

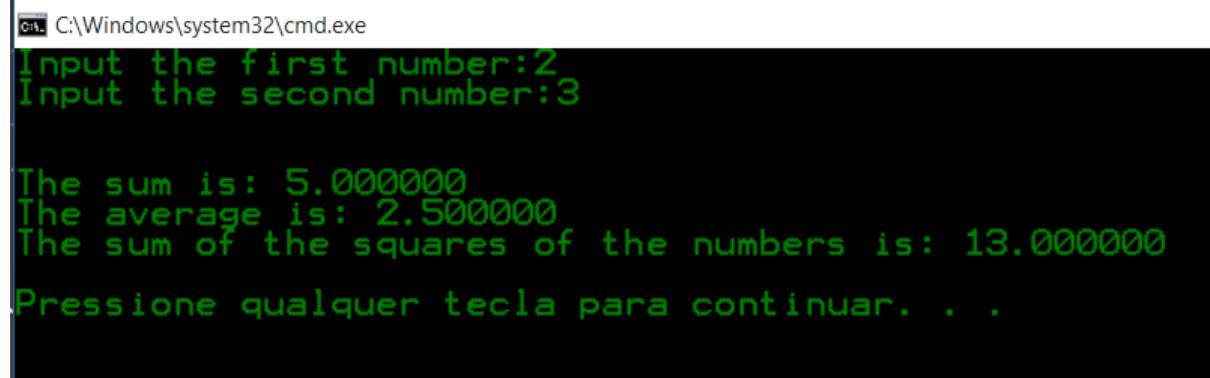
    float average = sum / 2;
    printf("The average is: %f\n", average);

    float square_sum = pow(x, 2) + pow(y, 2);
  
```

```
    printf("The sum of the squares of the numbers is: %f",
square_sum);

    printf("\n\n");
    return 0;
}
```

This program outputs:



```
C:\Windows\system32\cmd.exe
Input the first number:2
Input the second number:3

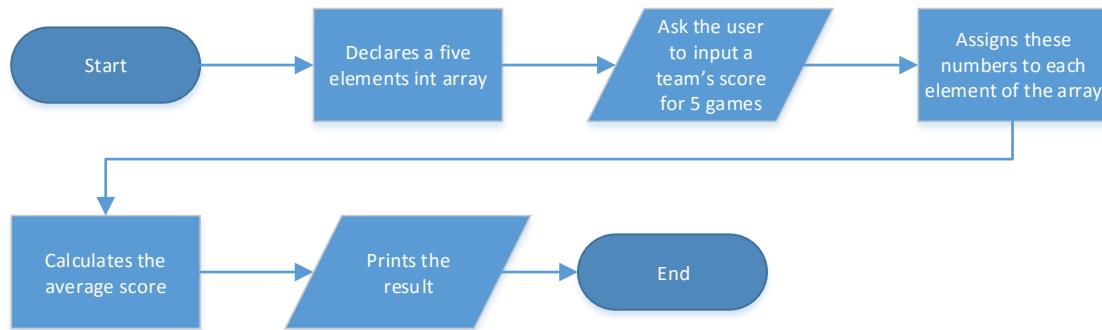
The sum is: 5.000000
The average is: 2.500000
The sum of the squares of the numbers is: 13.000000

Pressione qualquer tecla para continuar. . .
```

Figure 101: Two Number Input Output.

4.2.7 Exercise 7

This program must take from the user inputs to store a team's score for the last five matches and print out the average score. To write this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

int main()
{
    int games[5];

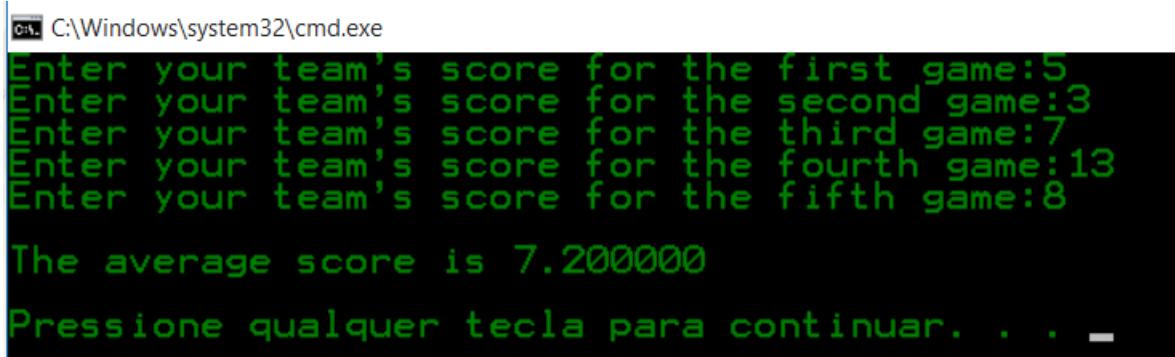
    printf("Enter your team's score for the first game:");
    scanf("%d", &games[0]);
    printf("Enter your team's score for the second game:");
    scanf("%d", &games[1]);
    printf("Enter your team's score for the third game:");
    scanf("%d", &games[2]);
    printf("Enter your team's score for the fourth game:");
    scanf("%d", &games[3]);
    printf("Enter your team's score for the fifth game:");
    scanf("%d", &games[4]);
    printf("\n");

    float sum = games[0] + games[1] + games[2] + games[3] +
    games[4];
    float average = sum / 5;

    printf("The average score is %f", average);
    
```

```
    printf("\n\n");
    return 0;
}
```

This program outputs:



```
C:\Windows\system32\cmd.exe
Enter your team's score for the first game:5
Enter your team's score for the second game:3
Enter your team's score for the third game:7
Enter your team's score for the fourth game:13
Enter your team's score for the fifth game:8

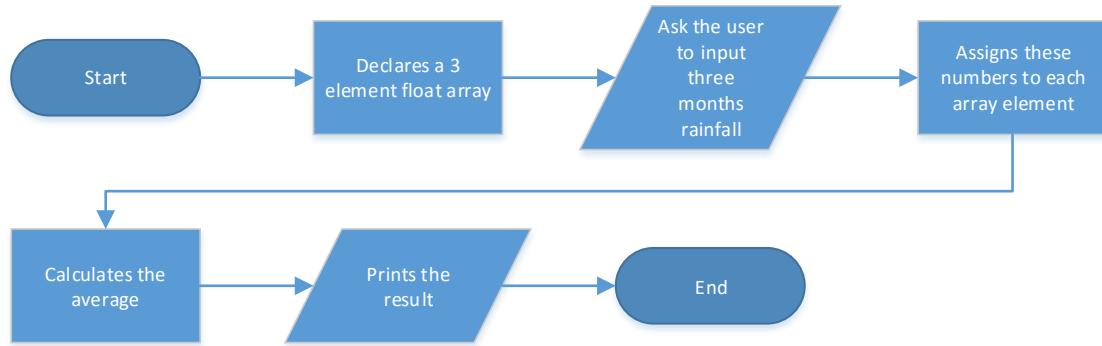
The average score is 7.200000

Pressione qualquer tecla para continuar. . . -
```

Figure 102: Rugby Scores Output.

4.2.8 Exercise 8

This program must take from the user the rainfall for June, July, and August, then, display the average rainfall. To write this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

int main()
{
    float rainfall[3];

    printf("Enter June's rainfall: ");
    scanf("%f", &rainfall[0]);
    printf("Enter July's rainfall: ");
    scanf("%f", &rainfall[1]);
    printf("Enter August's rainfall: ");
    scanf("%f", &rainfall[2]);
    printf("\n\n");

    float sum = rainfall[0] + rainfall[1] + rainfall[2];
    float average = sum / 3;

    printf("The average rainfall is: %f", average);

    printf("\n\n");
    return 0;
}
  
```

This program outputs:

```
C:\Windows\system32\cmd.exe
Enter June's rainfall: 33
Enter July's rainfall: 27
Enter August's rainfall: 19

The average rainfall is: 26.333334

Pressione qualquer tecla para continuar. . . -
```

Figure 103: Rainfall Output.

4.2.9 Exercise 9

This program must take a base from the user, then output the powers from zero to eight for that base, as in the example:

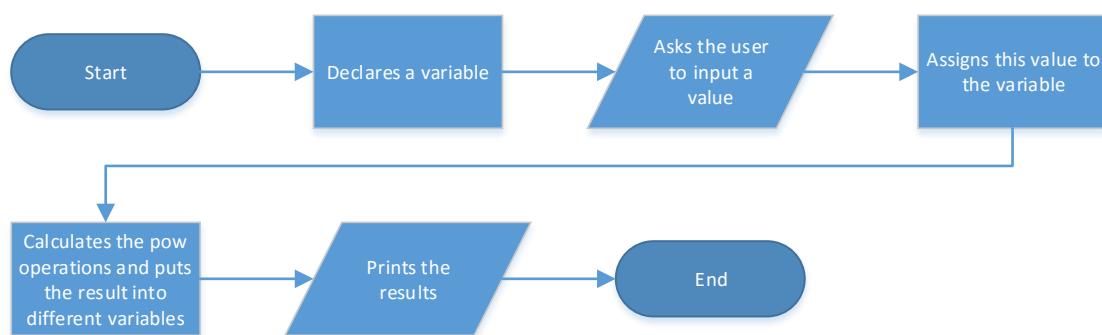
```
Please enter a base: 2

The powers of 2 are:

2 ^ 0 is 1
2 ^ 1 is 2
2 ^ 2 is 4
2 ^ 3 is 8
2 ^ 4 is 16
2 ^ 5 is 32
2 ^ 6 is 64
2 ^ 7 is 128
2 ^ 8 is 256
```

Figure 104: Powers Printer.

To develop this program, I wrote the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>
#include<math.h>

int main()
```

```
{  
    int base;  
    printf("Please enter a base: ");  
    scanf("%d", &base);  
    printf("\n\n");  
  
    int pow0 = pow(base, 0);  
    printf("%d ^ 0 is %d\n", base, pow0);  
  
    int pow1 = pow(base, 1);  
    printf("%d ^ 1 is %d\n", base, pow1);  
  
    int pow2 = pow(base, 2);  
    printf("%d ^ 2 is %d\n", base, pow2);  
  
    int pow3 = pow(base, 3);  
    printf("%d ^ 3 is %d\n", base, pow3);  
  
    int pow4 = pow(base, 4);  
    printf("%d ^ 4 is %d\n", base, pow4);  
  
    int pow5 = pow(base, 5);  
    printf("%d ^ 5 is %d\n", base, pow5);  
  
    int pow6 = pow(base, 6);  
    printf("%d ^ 6 is %d\n", base, pow6);  
  
    int pow7 = pow(base, 7);  
    printf("%d ^ 7 is %d\n", base, pow7);  
  
    int pow8 = pow(base, 8);  
    printf("%d ^ 8 is %d\n", base, pow8);  
  
    printf("\n\n");  
    return 0;  
}
```

This program outputs:



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window contains the following text:

```
Please enter a base: 7

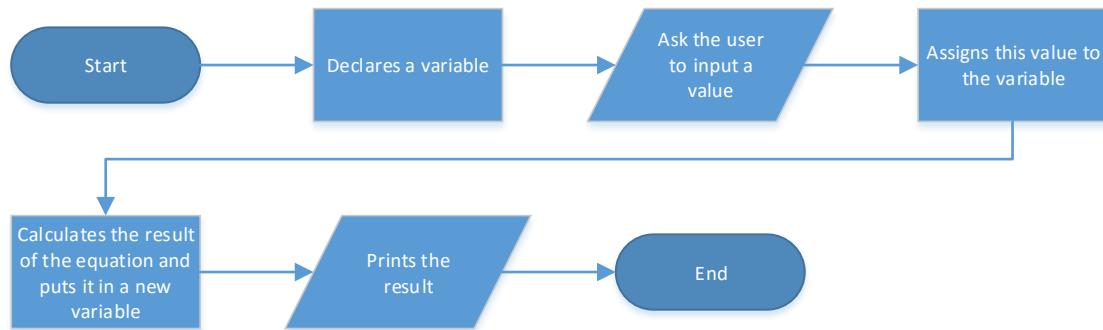
7 ^ 0 is 1
7 ^ 1 is 7
7 ^ 2 is 49
7 ^ 3 is 343
7 ^ 4 is 2401
7 ^ 5 is 16807
7 ^ 6 is 117649
7 ^ 7 is 823543
7 ^ 8 is 5764801

Pressione qualquer tecla para continuar. . . -
```

Figure 105: Powers Printer.

4.2.10 Exercise 10

This program must take a value from the user for the x variable, calculate the mathematical expression, then output the result. To write this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>
#include<math.h>

int main()
{
    float x;
    printf("Enter the x value: ");
    scanf("%f", &x);
    printf("\n\n");

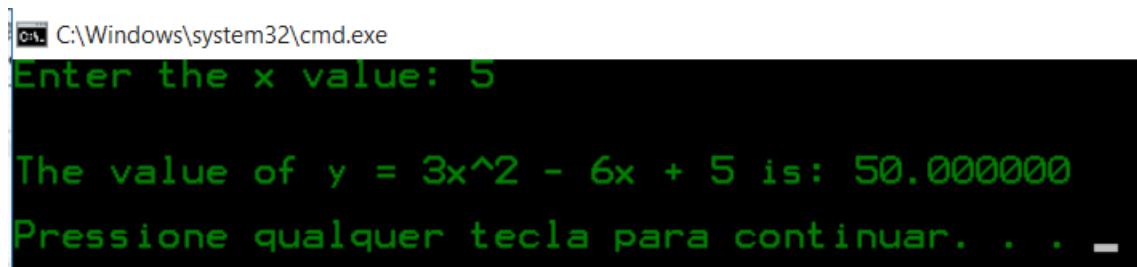
    float r1 = pow(x, 2);
    float r2 = 3 * r1;
    float r3 = 6 * x;
    float y = r2 - r3 + 5;

    printf("The value of y = 3x^2 - 6x + 5 is: %f", y);

    printf("\n\n");
    return 0;
}
    
```

The program calculates the y value for the expression $y = 3x^2 - 6x + 5$, x is a value that the user must input.

This program outputs:

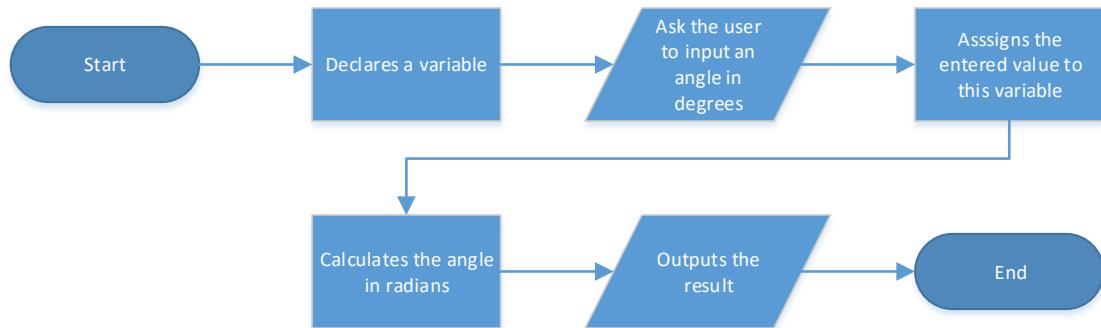


A screenshot of a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
Enter the x value: 5
The value of y = 3x^2 - 6x + 5 is: 50.000000
Pressione qualquer tecla para continuar. . .

Figure 106: Quadratic Evaluation Output.

4.2.11 Exercise 11

This program must take from the user an angle in degrees and outputs the equivalent in radians. To develop this program, I draw the following flowchart:



With this flowchart, I wrote the following code:

```

#include <stdio.h>
#include<math.h>

int main()
{
    float degrees;
    printf("Enter an angle in degrees: ");
    scanf("%f", &degrees);
    printf("\n\n");

    float radians = degrees * 0.017453293;
    printf("The angle in radians is: %f\n", radians);

    printf("\n\n");
    return 0;
}
  
```

To calculate the result, the program multiplies the user's input by 0.017453293.

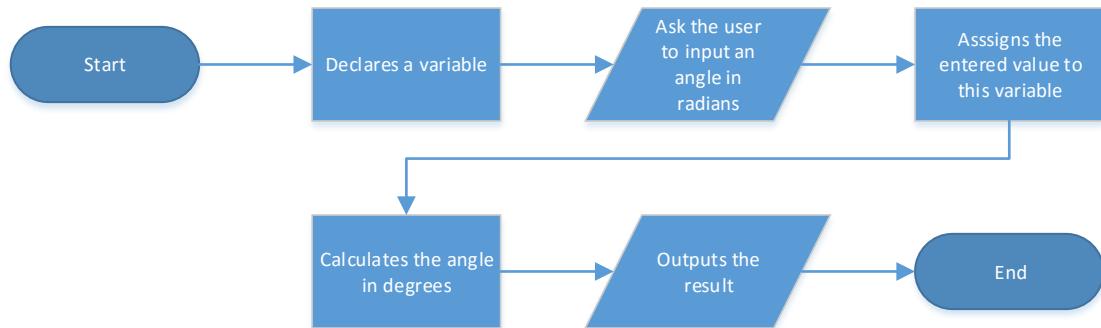
This program outputs:

```
C:\Windows\system32\cmd.exe
Enter an angle in degrees: 45
The angle in radians is: 0.785398
Pressione qualquer tecla para continuar. . .
```

Figure 107: Degrees to Radians.

4.2.12 Exercise 12

This program takes an angle in radians from the user and outputs the respective value in degrees. To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>
#include<math.h>

int main()
{
    float rad;
    printf("Enter an angle in radians: ");
    scanf("%f", &rad);
    printf("\n\n");

    float degrees = rad * 57.29577951;
    printf("The angle in degrees is: %f\n", degrees);

    printf("\n\n");
    return 0;
}
  
```

To calculate the answer, the program multiplies by 57.29577951 the user's input.

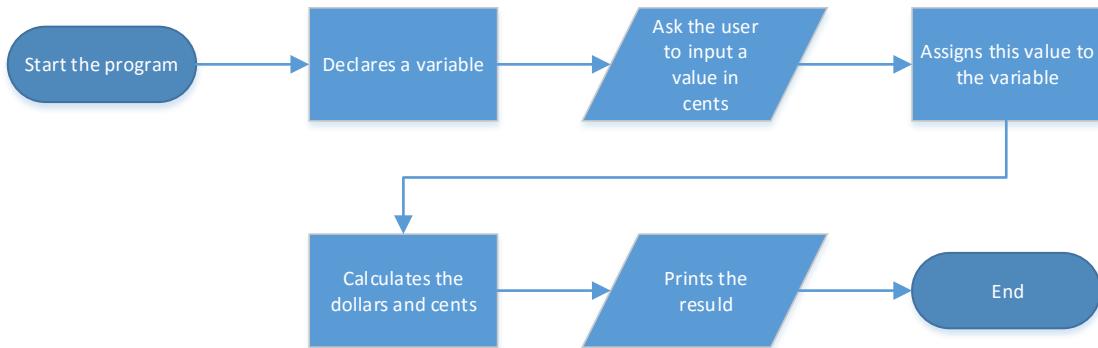
This program outputs:

```
C:\Windows\system32\cmd.exe
Enter an angle in radians: 6.28
The angle in degrees is: 359.817505
Pressione qualquer tecla para continuar. . .
```

Figure 108: Radians to Degrees Output.

4.2.13 Exercise 13

This program must take an input in cents, and give the number of dollars and cents. To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>
#include<math.h>

int main()
{
    int cents;
    printf("Enter the number of cents: ");
    scanf("%d", &cents);
    printf("\n\n");

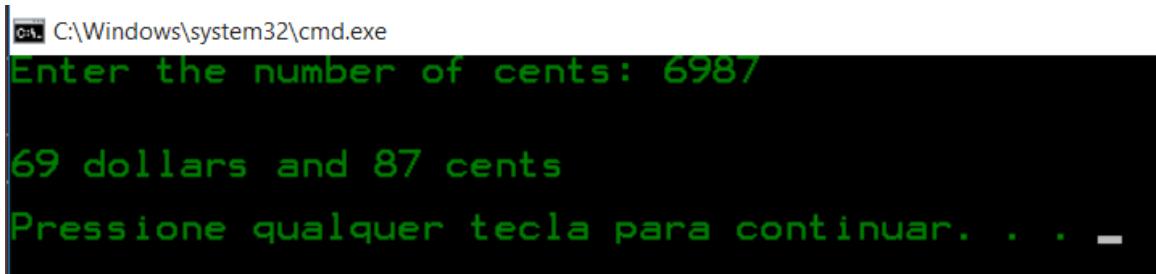
    int dollars = cents / 100;
    int cents_result = cents % 100;

    printf("%d dollars and %d cents", dollars, cents_result);

    printf("\n\n");
    return 0;
}
  
```

The program first takes the input from the user, then it calculates the cents divided by 100 with an int variable, calculates the rest of the operation and prints out both results.

This program outputs:

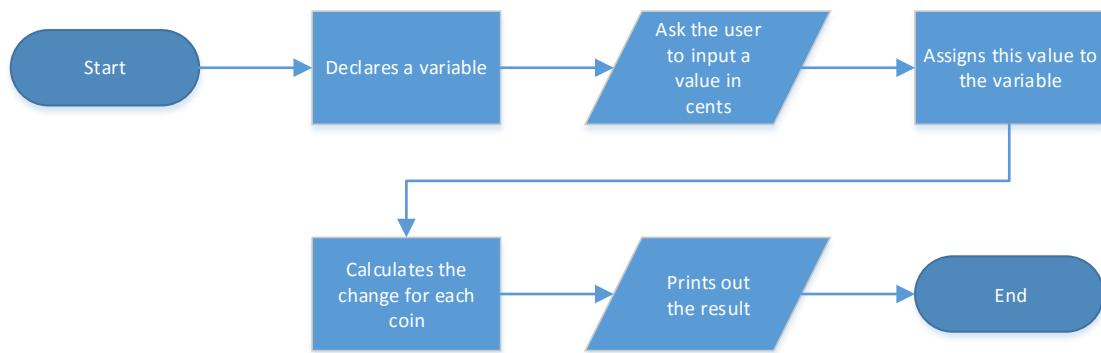


```
C:\Windows\system32\cmd.exe
Enter the number of cents: 6987
69 dollars and 87 cents
Pressione qualquer tecla para continuar. . . -
```

Figure 109: Number of Cents Output.

4.2.14 Exercise 14

This program must take an input from the user in cents, and calculates the change amount as two dollar, one dollar, 50 cent, 20 cent, 10 cent, 5 cent, 2 cent and 1 cent pieces. To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>
#include<math.h>

int main()
{
    int cents;
    printf("Enter the change in cents: ");
    scanf("%d", &cents);
    printf("\n\n");

    int s2 = cents / 200;
    cents %= 200;

    int s1 = cents / 100;
    cents %= 100;

    int s05 = cents / 50;
    cents %= 50;

    int s02 = cents / 20;
    cents %= 20;
  
```

```
int s01 = cents / 10;
cents %= 10;

int s005 = cents / 5;
cents %= 5;

int s002 = cents / 2;
cents %= 2;

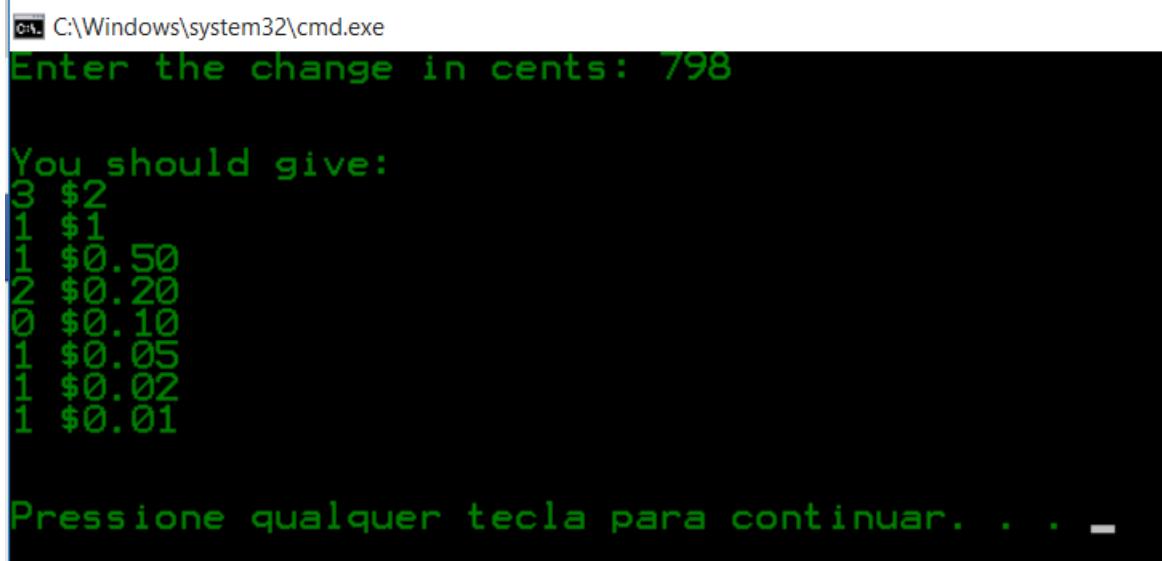
int s001 = cents;

printf("You should give:\n");
printf("%d $2\n", s2);
printf("%d $1\n", s1);
printf("%d $0.50\n", s05);
printf("%d $0.20\n", s02);
printf("%d $0.10\n", s01);
printf("%d $0.05\n", s005);
printf("%d $0.02\n", s002);
printf("%d $0.01\n", s001);

printf("\n\n");
return 0;
}
```

To calculate the change, the program takes the input from the user, calculates each piece separately dividing the piece by the number of pieces that are needed to get to the previous bigger piece, then it assigns the rest of the operation to the user input variable and repeats the process for all the other pieces of money.

This code outputs:



A screenshot of a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:

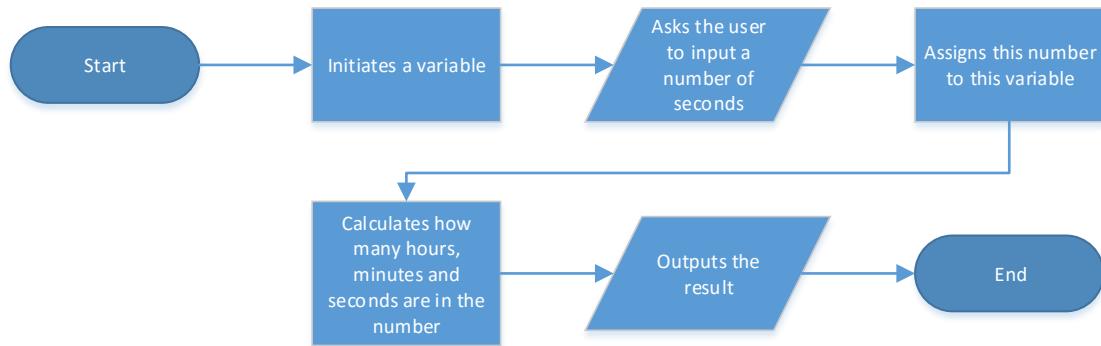
```
Enter the change in cents: 798
You should give:
3 $2
1 $1
1 $0.50
2 $0.20
0 $0.10
1 $0.05
1 $0.02
1 $0.01

Pressione qualquer tecla para continuar. . . -
```

Figure 110: Change Calculator Output.

4.2.15 Exercise 15

This program must take an input from the user in seconds and output the time in hours, minutes and seconds. To develop this code I draw the following flowchart.



With this flowchart, I wrote the following code:

```

#include <stdio.h>
#include<math.h>

int main()
{
    int seconds;
    printf("Enter a number in seconds: ");
    scanf("%d", &seconds);
    printf("\n\n");

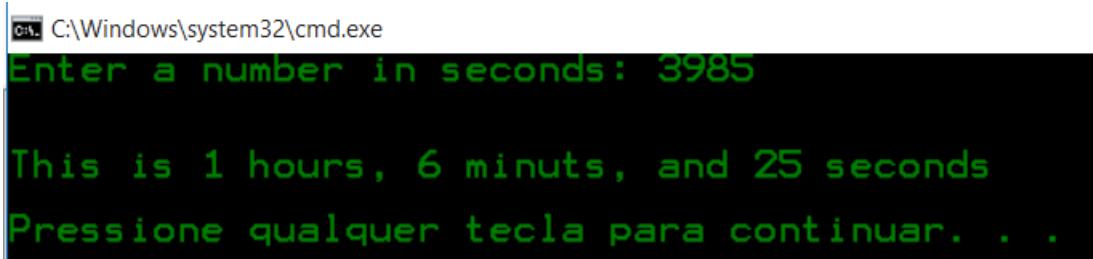
    int hours = seconds / 3600;
    int minutes = seconds / 60 - hours * 60;
    int seconds_result = seconds % 60;

    printf("This is %d hours, %d minutes, and %d seconds", hours,
minutes, seconds_result);

    printf("\n\n");
    return 0;
}
  
```

The logic used to design this code is the same as the previous two programs. It divides the seconds by the number of seconds that are in each element.

This program outputs:



```
C:\Windows\system32\cmd.exe
Enter a number in seconds: 3985
This is 1 hours, 6 minutes, and 25 seconds
Pressione qualquer tecla para continuar. . .
```

A screenshot of a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:

Enter a number in seconds: 3985

This is 1 hours, 6 minutes, and 25 seconds

Pressione qualquer tecla para continuar. . .

The text is displayed in white on a black background.

Figure 111: Seconds Converter Output.

4.2.16 Exercise 16

This program must output the following pattern to the console:

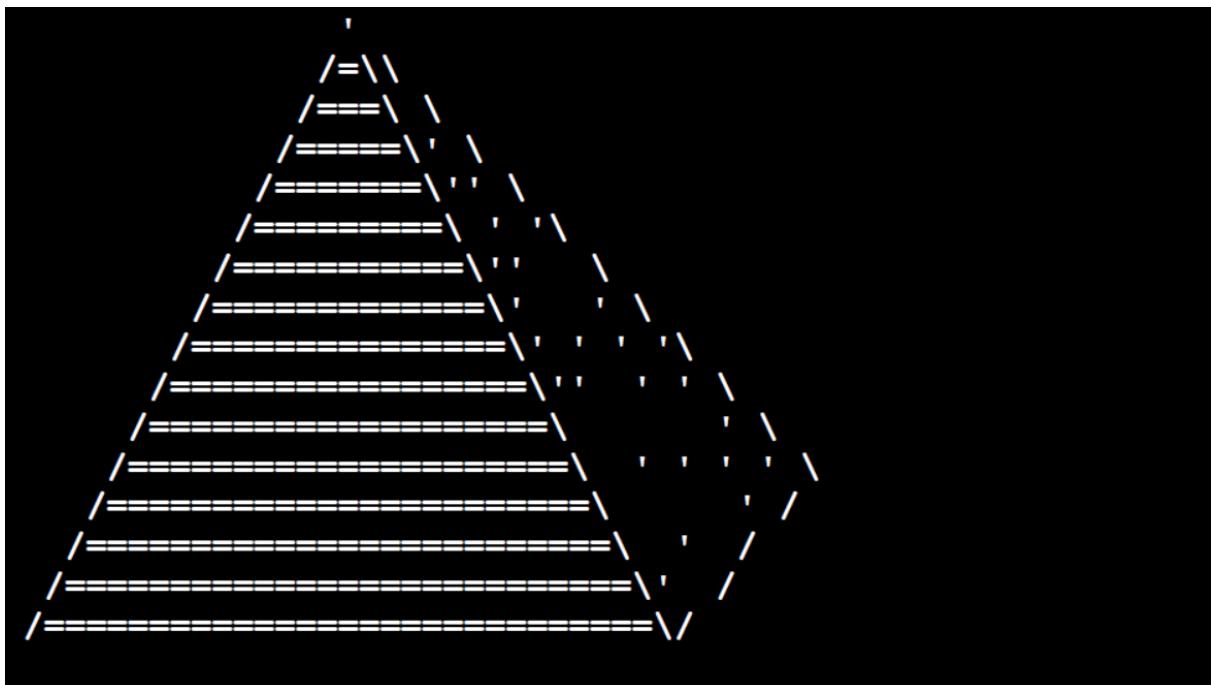


Figure 112: Pyramid.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>
#include<math.h>

int main()
{
```

This program outputs:

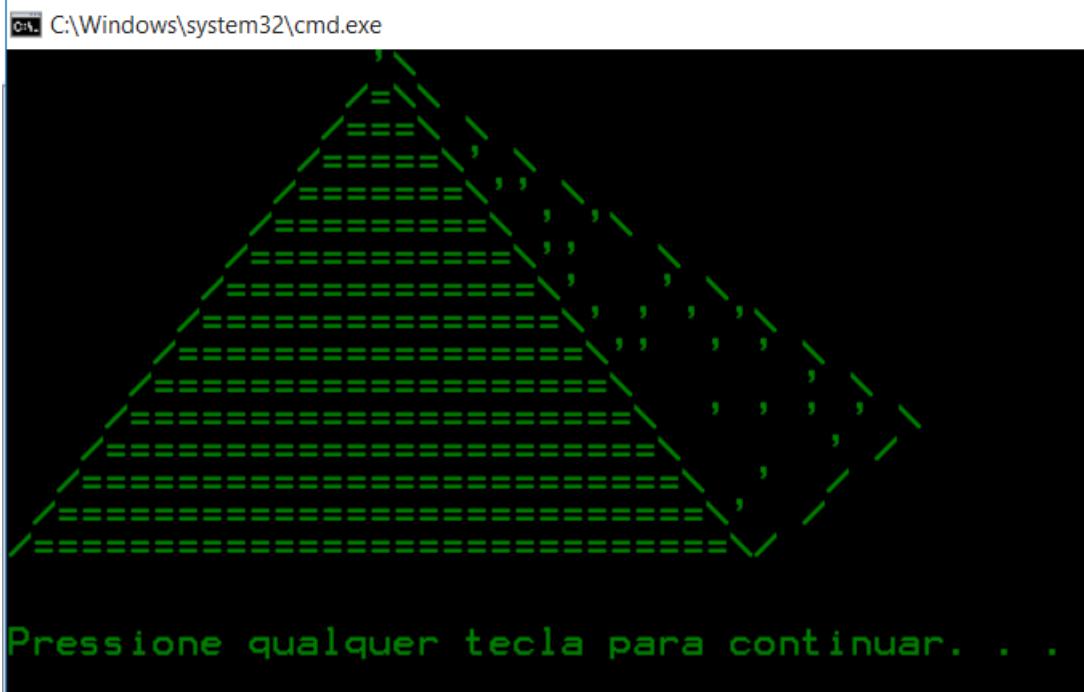


Figure 113: Pyramid Output.

5 Week Five

5.1 Lectures

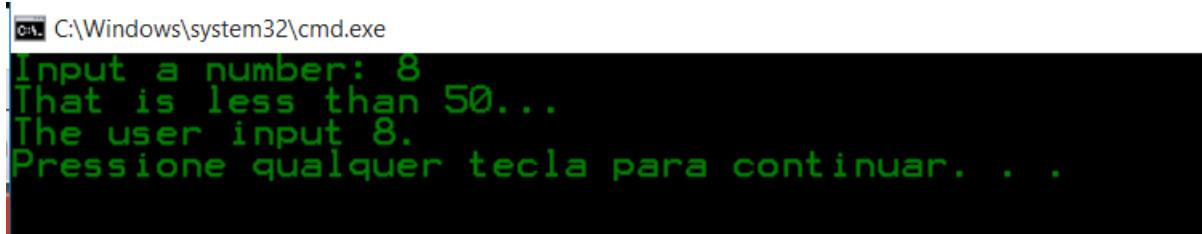
In this week we studied more debugging, and we saw a lot of examples of how to debug some errors that happens frequently.

We also reminded some topics of the previous weeks.

5.2 Examples

5.2.1 Example 1

This program asks the user to input a number, if the number is smaller than 50, it prints that the number is less then 50, and print the number.



The screenshot shows a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The user has typed 'Input a number: 8' and pressed Enter. The program then outputs 'That is less than 50...' followed by 'The user input 8.' and a prompt to press any key to continue. The text is in white on a black background.

Figure 114: Simple if.

This was the written code:

```
#include <stdio.h>

int main()
{
    int x = 0;

    printf("Input a number: ");

    scanf("%d", &x);

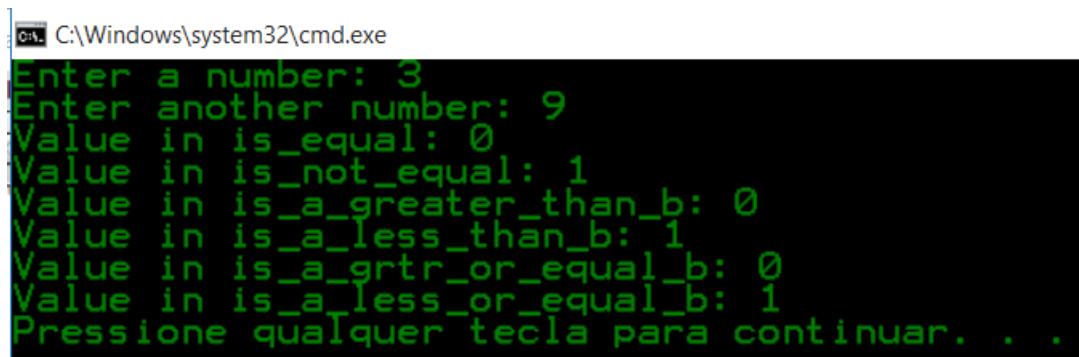
    if (x < 50)
    {
        printf("That is less than 50...\n");
    }

    printf("The user input %d.\n", x);

    return 0;
}
```

5.2.2 Example 2

The program takes from the user input from the user for two variables, and outputs if the variables are equal, not equal, which is greater than another, and greater and equal or less than equal.



```
C:\Windows\system32\cmd.exe
Enter a number: 3
Enter another number: 9
Value in is_equal: 0
Value in is_not_equal: 1
Value in is_a_greater_than_b: 0
Value in is_a_less_than_b: 1
Value in is_a_grtr_or_equal_b: 0
Value in is_a_less_or_equal_b: 1
Pressione qualquer tecla para continuar. . .
```

Figure 115: Relational Operators.

The written code was this:

```
#include <stdio.h>

int main()
{
    int a = 0;
    int b = 0;

    printf("Enter a number: ");
    scanf("%d", &a);

    printf("Enter another number: ");
    scanf("%d", &b);

    int is_equal = a == b;
    int is_not_equal = a != b;

    int is_a_greater_than_b = a > b;
    int is_a_less_than_b = a < b;
```

```
int is_a_grtr_or_equal_b = a >= b;
int is_a_less_or_equal_b = a <= b;

printf("Value in is_equal: ");
printf("%d\n", is_equal);

printf("Value in is_not_equal: ");
printf("%d\n", is_not_equal);

printf("Value in is_a_greater_than_b: ");
printf("%d\n", is_a_greater_than_b);

printf("Value in is_a_less_than_b: ");
printf("%d\n", is_a_less_than_b);

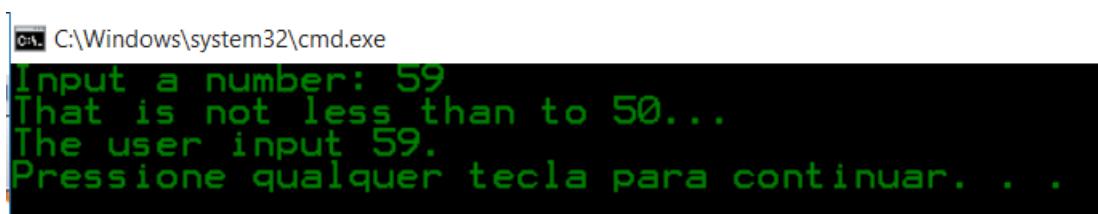
printf("Value in is_a_grtr_or_equal_b: ");
printf("%d\n", is_a_grtr_or_equal_b);

printf("Value in is_a_less_or_equal_b: ");
printf("%d\n", is_a_less_or_equal_b);

return 0;
}
```

5.2.3 Example 3

This program takes input from the user for a number, if the number is smaller than 50, it prints a message, if is not, it prints another message, then print the number.



```
C:\Windows\system32\cmd.exe
Input a number: 59
That is not less than to 50...
The user input 59.
Pressione qualquer tecla para continuar. . .
```

Figure 116: Simple if else.

The written code was this:

```
#include <stdio.h>

int main()
{
    int x = 0;

    printf("Input a number: ");

    scanf("%d", &x);

    if (x < 50)
    {
        printf("That is less than 50...\\n");
    }
    else
    {
        printf("That is not less than to 50...\\n");
    }

    printf("The user input %d.\\n", x);

    return 0;
}
```

5.2.4 Example 4

This program takes a number from the user and outputs if the number is smaller, bigger or equal to 50.



The screenshot shows a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. It displays the following text:
Input a number: 50
That is fifty...
The user input 50.
Pressione qualquer tecla para continuar. . . -

Figure 117: Simple if else if else.

This was the written code:

```
#include <stdio.h>

int main()
{
    int x = 0;

    printf("Input a number: ");

    scanf("%d", &x);

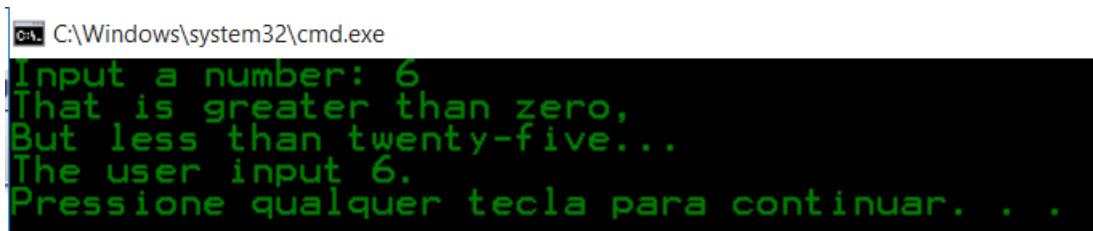
    if (x < 50)
    {
        printf("That is less than fifty...\n");
    }
    else if (x > 50)
    {
        printf("That is greater than fifty...\n");
    }
    else
    {
        printf("That is fifty...\n");
    }

    printf("The user input %d.\n", x);
}
```

```
    return 0;  
}
```

5.2.5 Example 5

This program takes a number from the user, says if the number is bigger than 100, 50, 25 and/or 0, equal to 0, or negative.



```
C:\Windows\system32\cmd.exe
Input a number: 6
That is greater than zero,
But less than twenty-five...
The user input 6.
Pressione qualquer tecla para continuar. . .
```

Figure 118: Simple Ladder.

The written code was this:

```
#include <stdio.h>

int main()
{
    int x = 0;

    printf("Input a number: ");

    scanf("%d", &x);

    if (x > 100)
    {
        printf("Greater than one-hundred...\n");
    }
    else if (x > 50)
    {
        printf("Greater than fifty,\n");
        printf("But less than one-hundred...\n");
    }
    else if (x > 25)
    {
        printf("That is greater than twenty-five,\n");
        printf("But less than fifty...\n");
    }
    else if (x > 0)
```

```
{  
    printf("That is greater than zero,\n");  
    printf("But less than twenty-five...\n");  
}  
else if (x == 0)  
{  
    printf("That is zero...\n");  
}  
else  
{  
    printf("That is negative...\n");  
}  
  
printf("The user input %d.\n", x);  
  
return 0;  
}
```

5.2.6 Example 6

Takes a number from the user and outputs if the number is positive, negative, or zero.



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
Input a whole number: 7
Input is a positive number.
Pressione qualquer tecla para continuar. . .

Figure 119: Whole Number Classifier.

The written code is this:

```
#include <stdio.h>

int main()
{
    int x;

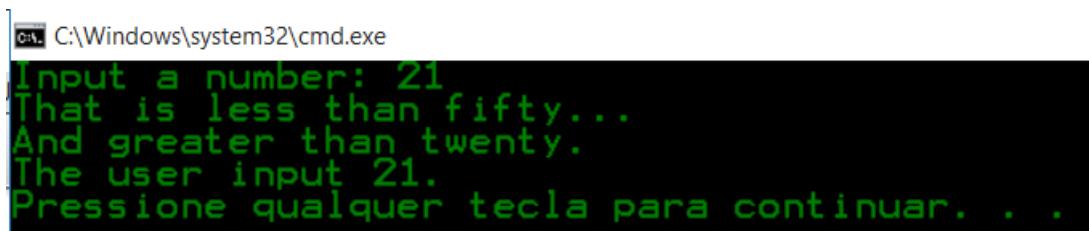
    printf("Input a whole number: ");
    scanf("%d", &x);

    if (x > 0)
    {
        printf("Input is a positive number.\n");
    }
    else if (x < 0)
    {
        printf("Input is a negative number.\n");
    }
    else
    {
        printf("Input is zero.\n");
    }

    return 0;
}
```

5.2.7 Example 7

This code takes a number as input from the user, outputs if the number is smaller than fifty and greater than twenty, or between twenty and fifty.



```
C:\Windows\system32\cmd.exe
Input a number: 21
That is less than fifty...
And greater than twenty.
The user input 21.
Pressione qualquer tecla para continuar. . .
```

Figure 120: Simple Nested if.

This was the written code:

```
#include <stdio.h>

int main()
{
    int x = 0;

    printf("Input a number: ");

    scanf("%d", &x);

    if (x < 50)
    {
        printf("That is less than fifty...\n");
        if (x > 20)
        {
            printf("And greater than twenty.\n");
        }
        else
        {
            printf("And less than or ");
            printf("equal to twenty.\n");
        }
    }

    printf("The user input %d.\n", x);
```

```
    return 0;  
}
```

5.2.8 Example 8

This program gives examples of how truth tables work.

```

C:\Windows\system32\cmd.exe
AND Truth Table:
a | b | a && b
---+---+-----
0 | 0 | 0
0 | 1 | 0
1 | 0 | 0
1 | 1 | 1

OR Truth Table:
a | b | a || b
---+---+-----
0 | 0 | 0
0 | 1 | 1
1 | 0 | 1
1 | 1 | 1

NOT Truth Table:
a | !a
---+-----
0 | 1
1 | 0

Pressione qualquer tecla para continuar. . .

```

Figure 121: Truth Tables.

The written code was this:

```

#include <stdio.h>

int main()
{
    printf("AND Truth Table:\n\n");
    printf(" a | b | a && b \n");

```

```
printf("----+---+-----\n");
printf(" 0 | 0 | %d \n", 0 && 0);
printf(" 0 | 1 | %d \n", 0 && 1);
printf(" 1 | 0 | %d \n", 1 && 0);
printf(" 1 | 1 | %d \n", 1 && 1);
printf("\n");

printf("OR Truth Table:\n\n");

printf(" a | b | a || b \n");
printf("----+---+-----\n");
printf(" 0 | 0 | %d \n", 0 || 0);
printf(" 0 | 1 | %d \n", 0 || 1);
printf(" 1 | 0 | %d \n", 1 || 0);
printf(" 1 | 1 | %d \n", 1 || 1);
printf("\n");

printf("NOT Truth Table:\n\n");

printf(" a | !a \n");
printf("----+---\n");
printf(" 0 | %d\n", !0);
printf(" 1 | %d\n", !1);
printf("\n");

return 0;
}
```

5.2.9 Example 9

This program takes from the user 1 and 0 inputs and does a series of truth operations with these numbers.

```
C:\Windows\system32\cmd.exe
Enter a (0 for false, 1 for true): 1
Enter b (0 for false, 1 for true): 0
Enter c (0 for false, 1 for true): 0
not_a is 0
not_b is 1
not_c is 1
a_and_b is 0
a_and_c is 0
b_and_c is 0
a_and_b_and_c is 0
a_or_b is 1
a_or_c is 1
b_or_c is 0
a_or_b_or_c is 1
a_xor_b is 1
a_xor_c is 1
Pressione qualquer tecla para continuar. . .
```

Figure 122: Logical Operators.

This was the written code:

```
#include <stdio.h>
int main()
{
    int a = 0;
    int b = 0;
    int c = 0;

    printf("Enter a (0 for false, 1 for true): ");
    scanf("%d", &a);
    printf("Enter b (0 for false, 1 for true): ");
    scanf("%d", &b);
    printf("Enter c (0 for false, 1 for true): ");
    scanf("%d", &c);
```

```
int not_a = !a;
int not_b = !b;
int not_c = !c;

int a_and_b = a && b;
int a_and_c = a && c;
int b_and_c = b && c;
int a_and_b_and_c = a && b && c;

int a_or_b = a || b;
int a_or_c = a || c;
int b_or_c = b || c;
int a_or_b_or_c = a || b || c;

int a_xor_b = (a || b) && !(a && b);
int a_xor_c = (a || c) && !(a && c);

printf("not_a is %d\n", not_a);
printf("not_b is %d\n", not_b);
printf("not_c is %d\n", not_c);

printf("a_and_b is %d\n", a_and_b);
printf("a_and_c is %d\n", a_and_c);
printf("b_and_c is %d\n", b_and_c);
printf("a_and_b_and_c is %d\n", a_and_b_and_c);

printf("a_or_b is %d\n", a_or_b);
printf("a_or_c is %d\n", a_or_c);
printf("b_or_c is %d\n", b_or_c);
printf("a_or_b_or_c is %d\n", a_or_b_or_c);

printf("a_xor_b is %d\n", a_xor_b);
printf("a_xor_c is %d\n", a_xor_c);
return 0;
}
```

5.2.10 Example 10

This example shows how switch case works in C programming, it takes an input from the user from 0 to 8 and gives a different output for each case.

```
C:\Windows\system32\cmd.exe
Input an odd number between 0 and 8: 5
User input FIVE
Pressione qualquer tecla para continuar. . .
```

Figure 123: Simple switch.

The written code was this:

```
#include <stdio.h>
int main()
{
    int number;

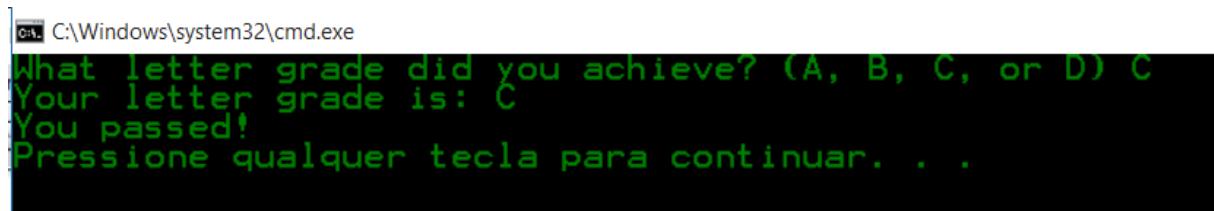
    printf("Input an odd number between 0 and 8: ");
    scanf("%d", &number);

    switch (number)
    {
        case 1:
        {
            printf("User input ONE\n");
            break;
        }
        case 3:
        {
            printf("User input THREE\n");
            break;
        }
        case 5:
        {
            printf("User input FIVE\n");
            break;
        }
        case 7:
        {
            printf("User input SEVEN\n");
            break;
        }
    }
}
```

```
default:  
{  
    printf("User input does not match ");  
    printf("the requirements!\n\n");  
    printf("Should be 1, 3, 5, or 7.\n");  
    break;  
}  
}  
  
return 0;  
}
```

5.2.11 Example 11

This example takes a char from the user and displays different outputs for each letter, according to AUT's grades, the program displays if the user succeeded or failed the paper.



```
C:\Windows\system32\cmd.exe
What letter grade did you achieve? (A, B, C, or D) C
Your letter grade is: C
You passed!
Pressione qualquer tecla para continuar. . .
```

Figure 124: Result Checker.

This was the written code:

```
#include <stdio.h>

int main()
{
    char grade = 0;

    printf("What letter grade did you achieve? ");
    printf("(A, B, C, or D) ");
    scanf("%c", &grade);
    printf("Your letter grade is: %c\n", grade);

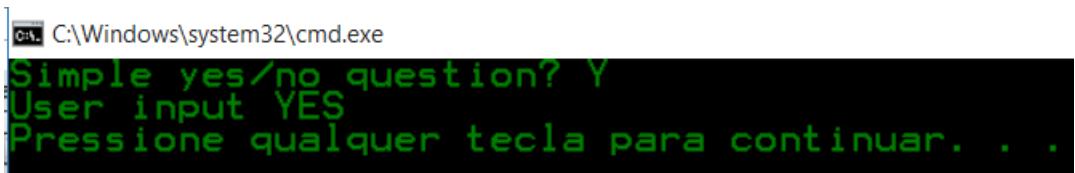
    switch (grade)
    {
        case 'A':
        {
            printf("Excellent result!\n");
            break;
        }
        case 'B':
        {
            printf("Well done!\n");
            break;
        }
        case 'C':
        {
            printf("You passed!\n");
        }
    }
}
```

```
        break;
    }
case 'D':
{
    printf("Better study more next time!\n");
    break;
}
default:
{
    printf("Invalid grade entered!!!\n");
    break;
}
}

return 0;
}
```

5.2.12 Example 12

This example asks a simple yes or no question, but it considers both lower and upper case letters for the answer using the switch fall through.



The screenshot shows a Windows command prompt window titled 'cmd' with the path 'C:\Windows\system32\cmd.exe'. The window displays the following text:
Simple yes/no question? Y
User input YES
Pressione qualquer tecla para continuar. . .

Figure 125: Simple switch with Fall Through.

This was the written code:

```
#include <stdio.h>

int main()
{
    char input;

    printf("Simple yes/no question? ");
    scanf("%c", &input);

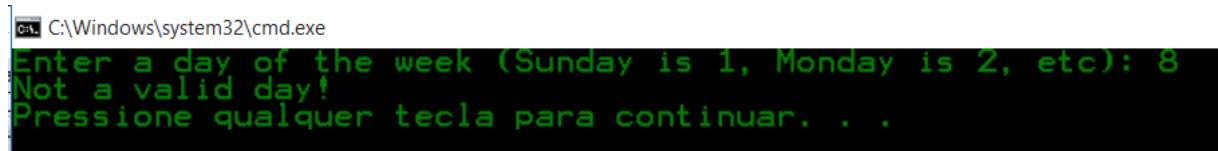
    switch (input)
    {
        case 'Y': //Fall Through.
        case 'y':
        {
            printf("User input YES\n");
            break;
        }
        case 'N': //Fall Through.
        case 'n':
        {
            printf("User input NO\n");
            break;
        }
        default:
        {
            printf("User input does not match ");
        }
    }
}
```

```
    printf("the requirements!\n\n");
    printf("Should be Y, y, N, or n.\n");
    break;
}
}

return 0;
}
```

5.2.13 Example 13

Using switch, the program takes an input from the user for a day of the week as a number and says if the day is a week day or a weekend, or if the user input a invalid day.



```
C:\Windows\system32\cmd.exe
Enter a day of the week (Sunday is 1, Monday is 2, etc): 8
Not a valid day!
Pressione qualquer tecla para continuar. . .
```

Figure 126: Day of the Week.

This was the written code:

```
#include <stdio.h>

int main()
{
    int day;

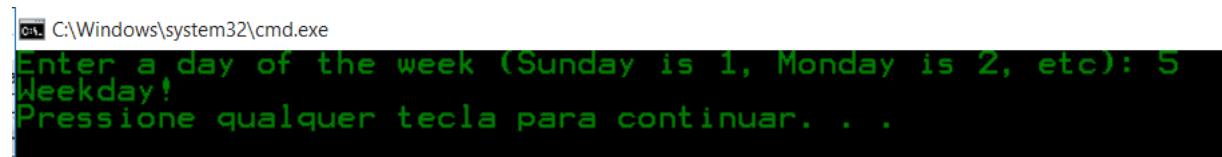
    printf("Enter a day of the week ");
    printf("(Sunday is 1, Monday is 2, etc): ");
    scanf("%d", &day);

    switch (day)
    {
        case 1: //Sunday, fall through.
        case 7: //Saturday
        {
            printf("Weekend!\n");
            break;
        }
        case 2: //Monday, fall through.
        case 3: //Tuesday, fall through.
        case 4: //Wednesday, fall through.
        case 5: //Thursday, fall Through.
        case 6: //Friday
        {
            printf("Weekday!\n");
            break;
        }
    }
}
```

```
    default:  
    {  
        printf("Not a valid day!\n");  
        break;  
    }  
}  
  
return 0;  
}
```

5.2.14 Example 14

This program has the same function as the previous example, but it uses a nested if else to print the result.



```
C:\Windows\system32\cmd.exe
Enter a day of the week (Sunday is 1, Monday is 2, etc): 5
Weekday!
Pressione qualquer tecla para continuar. . .
```

Figure 127: Day of the Week Nested if else.

This was the written code:

```
#include <stdio.h>

int main()
{
    int day;

    printf("Enter a day of the week ");
    printf("(Sunday is 1, Monday is 2, etc): ");
    scanf("%d", &day);

    if (1 <= day && day <= 7)
    {
        if (1 == day || 7 == day)
        {
            printf("Weekend!\n");
        }
        else
        {
            printf("Weekday!\n");
        }
    }
    else
    {
        printf("Not a valid day!\n");
    }

    return 0;
}
```

14885857

Reporting Journal #4

}

5.2.15 Example 15

This program takes input from the user for an mathematical operation and two numbers.

```
C:\Windows\system32\cmd.exe
Math Operations:
[a] Add
[s] Subtract
[m] Multiply
[d] Divide

Choose an operation:
m

Enter the left operand:
3

Enter the right operand:
8
3 * 8 is 24
Pressione qualquer tecla para continuar. . .
```

Figure 128: switch for Math Operation.

This was the written code:

```
#include <stdio.h>
int main()
{
    char operation = 0;
    int left = 0;
    int right = 0;

    printf("Math Operations: \n");
    printf(" [a] Add \n");
    printf(" [s] Subtract \n");
    printf(" [m] Multiply \n");
    printf(" [d] Divide \n");
    printf("\nChoose an operation: \n");
    scanf(" %c", &operation);
    printf("\nEnter the left operand: \n");
    scanf(" %d", &left);
    printf("\nEnter the right operand: \n");
```

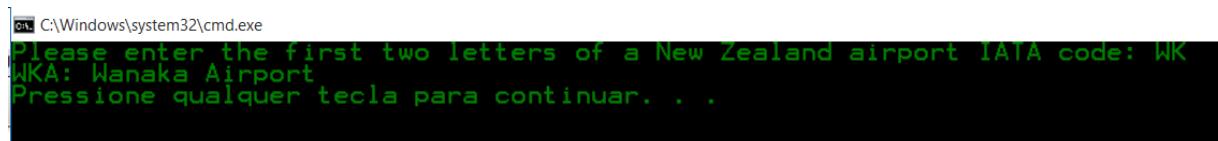
```
scanf(" %d", &right);

switch (operation)
{
    case 'a':
    {
        printf("%d + %d is %d\n", left, right, left +
right);
        break;
    }
    case 'm':
    {
        printf("%d * %d is %d\n", left, right, left *
right);
        break;
    }
    case 'd':
    {
        printf("%d / %d is %d\n", left, right, left /
right);
        break;
    }
    case 's':
    {
        printf("%d - %d is %d\n", left, right, left -
right);
        break;
    }
    default:
    {
        printf("Unknown operation!\n");
        break;
    }
}

return 0;
}
```

5.2.16 Example 16

This example shows how a nested switch works. The user input two chars, the switch takes the first char, and the second char goes to the nested switch.



The screenshot shows a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. It displays the following text:
Please enter the first two letters of a New Zealand airport IATA code: WK
WKA: Wanaka Airport
Pressione qualquer tecla para continuar. . .

Figure 129: Nested switch.

This was the written code:

```
#include <stdio.h>
int main()
{
    char iata[2];
    printf("Please enter the first two letters of ");
    printf("a New Zealand airport IATA code: ");
    scanf("%c%c", &iata[0], &iata[1]);

    switch (iata[0])
    {
        case 'P':
        {
            switch (iata[1])
            {
                case 'M':
                {
                    printf("PMR: Palmerston North");
                }
                break;
                case 'P':
                {
                    printf("PPQ: Kapiti Coast Airport");
                }
                break;
                case 'C':
                {
                    printf("PCN: Picton Aerodrome");
                }
            }
        }
    }
}
```

```
        }
        break;
    }
    break;
}
case 'W':
{
    switch (iata[1])
    {
        case 'K':
        {
            printf("WKA: Wanaka Airport");
        }
        break;
        case 'A':
        {
            printf("WAG: Wanganui Airport");
        }
        break;
        case 'L':
        {
            printf("WLG: Wellington International
Airport");
        }
        break;
    }
    break;
}
default:
{
    printf("Unknown IATA code!");
    break;
}
}

printf("\n");

return 0;
}
```

5.3 Exercises

5.3.1 Exercise 1

In this exercise the student should develop a program that asks the user to input two real numbers, output the division of the first number by the second number.

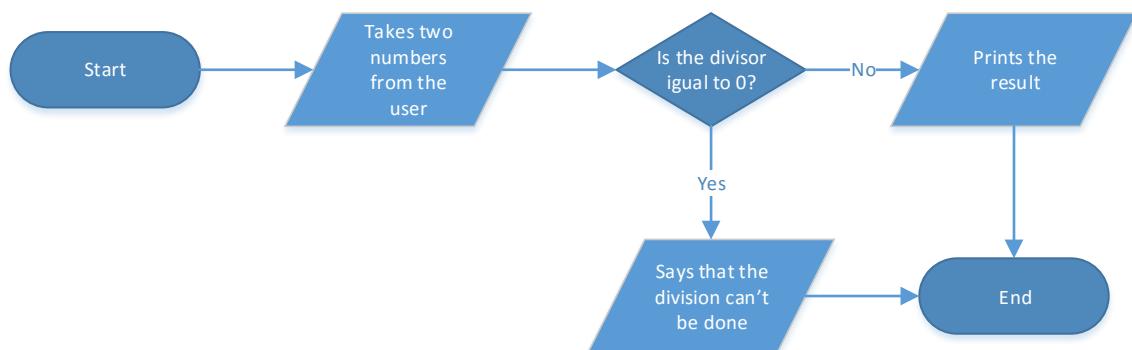
```
Enter the numerator: 5
Enter the denominator: 7

Trying to calculate: 5 / 7

The result is: 0.714286
```

Figure 130: Divide by Zero.

To write this code, I draw the following flowchart:



Using this flowchart, I wrote the following program:

```
#include <stdio.h>

int main()
{
```

```

double numerator;
printf("Enter the numerator: ");
scanf("%f", &numerator);

double denominator;
printf("Enter the denominator: ");
scanf("%f", &denominator);

printf("\n\n");

printf("Trying to calculate : %f / %f", numerator,
denominator);
printf("\n\n");

double result;

if (denominator != 0)
{
    result = numerator / denominator;

    printf("The result is: %f", result);
}
else
{
    printf("Unable to calculate due to divide by zero!");
}

printf("\n\n");
return 0;
}

```

But the output was not as expected:

```

C:\Windows\system32\cmd.exe
Enter the numerator: 6
Enter the denominator: 3
Trying to calculate : -9255960452106777587750189883038321855178746716337109586673664.000000 / -925596044252868062836321362356
41632658284422740165914450198528.000000
The result is: 1.000000
Pressione qualquer tecla para continuar. . .

```

Figure 131: Wrong output.

While debugging, I noticed that the variable was not taking the correct input. I had two options, change the variable type, or the modifier. I tried changing the modifier to %lf, but the

problem persisted. I tried cleaning and rebuilding the solution, but it did not work, to make the program work, I had to restart the computer. This is the final code:

```
#include <stdio.h>

int main()
{
    float numerator;
    printf("Enter the numerator: ");
    scanf("%f", &numerator);

    float denominator;
    printf("Enter the denominator: ");
    scanf("%f", &denominator);

    printf("\n\n");

    printf("Trying to calculate : %f / %f", numerator,
denominator);
    printf("\n\n");

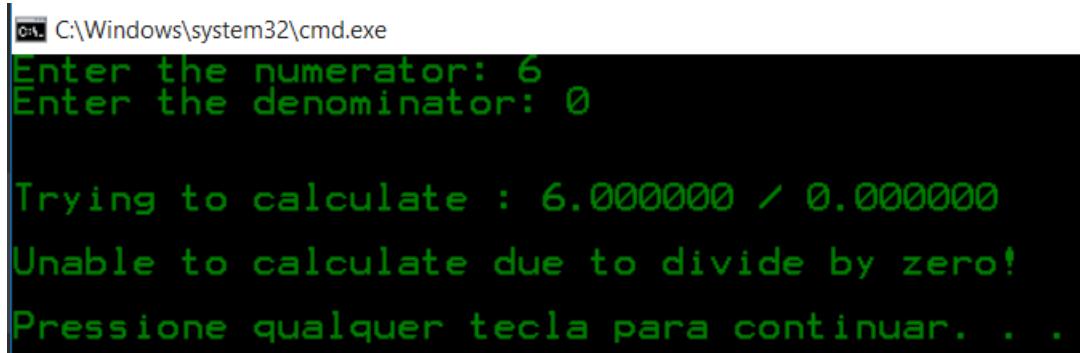
    float result;

    if (denominator != 0)
    {
        result = numerator / denominator;

        printf("The result is: %f", result);
    }
    else
    {
        printf("Unable to calculate due to divide by zero!");
    }

    printf("\n\n");
    return 0;
}
```

This program outputs:



A screenshot of a Windows Command Prompt window titled "C:\Windows\system32\cmd.exe". The window contains the following text:
Enter the numerator: 6
Enter the denominator: 0

Trying to calculate : 6.000000 / 0.000000
Unable to calculate due to divide by zero!
Pressione qualquer tecla para continuar. . .

Figure 132: Divide by Zero Output.

5.3.2 Exercise 2

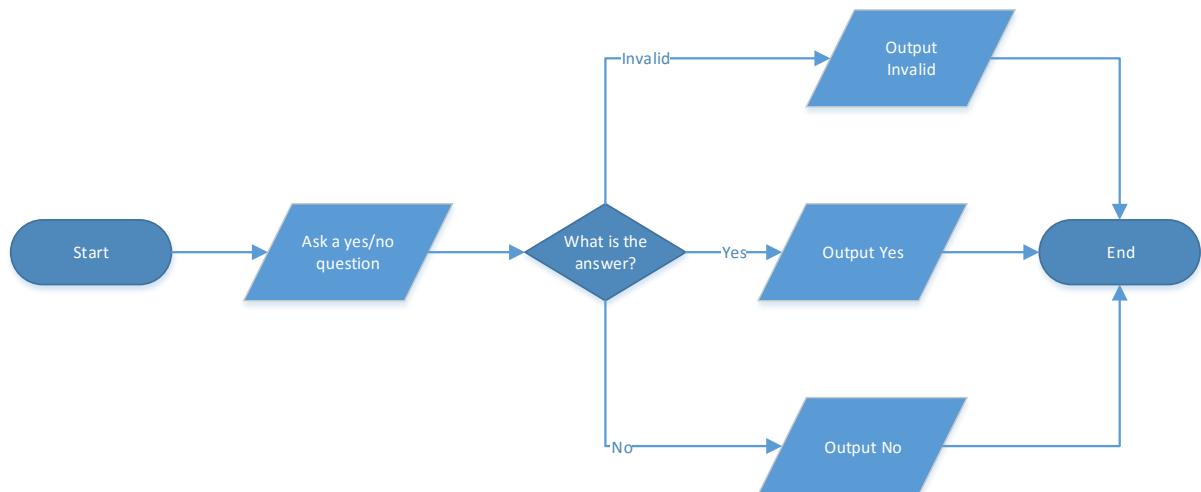
In this example, the program asks a yes/no question, takes a single letter as a response from the user, both upper and lower case letters must be valid.

```
Question (Y/N)? n
```

```
User response: No
```

Figure 133: Single Letter Response.

To write this code, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    char answer;
    printf("Question (Y/N)?");
    scanf("%c", &answer);

    if (answer == 'y' || answer == 'Y')
```

```
{  
    printf("User response: Yes");  
}  
else if (answer == 'n' || answer == 'N')  
{  
    printf("User response: No");  
}  
else  
{  
    printf("User response: Invalid input!");  
}  
  
printf("\n\n");  
  
return 0;  
}
```

This code outputs:



```
C:\Windows\system32\cmd.exe  
Question (Y/N)?n  
User response: No  
Pressione qualquer tecla para continuar. . .
```

Figure 134: Single Letter Response Output.

5.3.3 Exercise 3

The student should develop a program that uses a sequence and selection to ask the user to input a birthday month as a whole month number, and outputs, according to the New Zealand season, in which season was the person born.

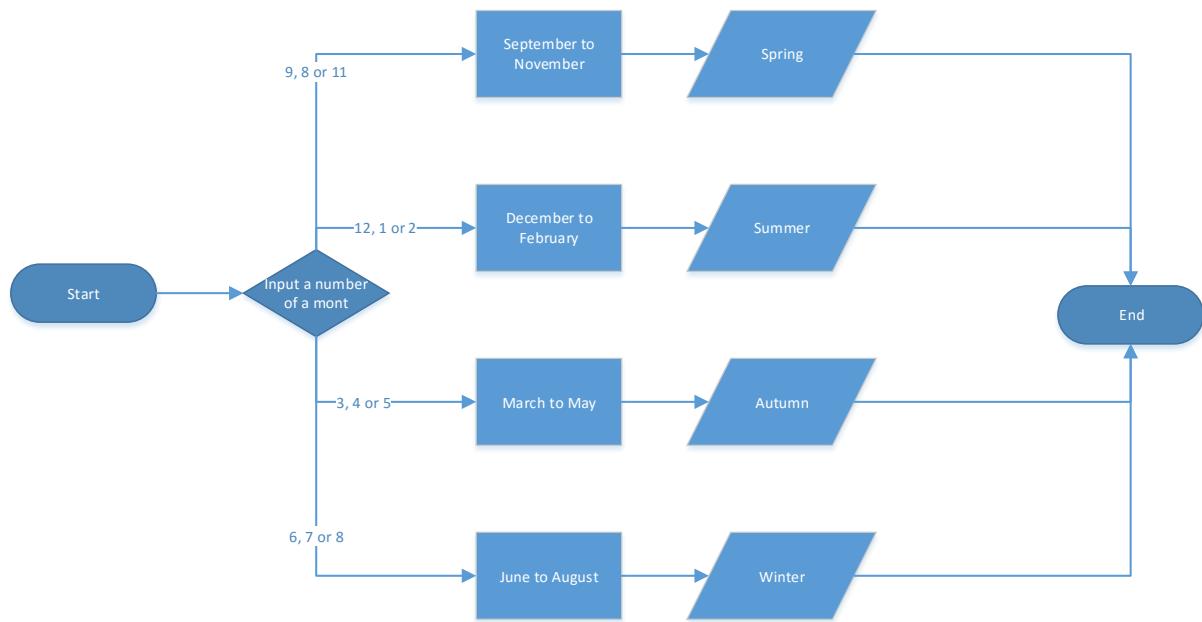
```
Key: January is 1
      February is 2
      March is 3
      April is 4
      May is 5
      June is 6
      July is 7
      August is 8
      September is 9
      October is 10
      November is 11
      December is 12
```

```
What month were you born in? 8
```

```
August in New Zealand is in Winter.
```

Figure 135: Birthday Season.

To write this code, I draw the following flowchart



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

int main()
{
    printf("Key: January is 1\n");
    printf(" February is 2\n");
    printf(" March is 3\n");
    printf(" April is 4\n");
    printf(" May is 5\n");
    printf(" June is 6\n");
    printf(" July is 7\n");
    printf(" August is 8\n");
    printf(" September is 9\n");
    printf(" October is 10\n");
    printf(" November is 11\n");
    printf(" December is 12\n");

    printf("\n");

    int month;
    printf("What month were you born in?");
    scanf("%d", &month);
  
```

```
printf("\n");

switch (month)
{
    case 1:
    {
        printf("January in New Zealand is in Summer.");
        break;
    }
    case 2:
    {
        printf("February in New Zealand is in Summer.");
        break;
    }
    case 3:
    {
        printf("March in New Zealand is in Autumn.");
        break;
    }
    case 4:
    {
        printf("April in New Zealand is in Autumn.");
        break;
    }
    case 5:
    {
        printf("May in New Zealand is in Autumn.");
        break;
    }
    case 6:
    {
        printf("June in New Zealand is in Winter.");
        break;
    }
    case 7:
    {
        printf("July in New Zealand is in Winter.");
        break;
    }
    case 8:
    {
        printf("August in New Zealand is in Winter.");
        break;
    }
    case 9:
    {
        printf("September in New Zealand is in Spring.");
    }
}
```

```
        break;
    }
case 10:
{
    printf("October in New Zealand is in Spring.");
    break;
}
case 11:
{
    printf("November in New Zealand is in Spring.");
    break;
}
case 12:
{
    printf("December in New Zealand is in Summer.");
    break;
}
default:
{
    printf("This is not a valid number.");
    break;
}
}

printf("\n\n");

return 0;
}
```

This code outputs:

```
C:\Windows\system32\cmd.exe
Key: January is 1
      February is 2
      March is 3
      April is 4
      May is 5
      June is 6
      July is 7
      August is 8
      September is 9
      October is 10
      November is 11
      December is 12

What month were you born in?4
April in New Zealand is in Autumn.

Pressione qualquer tecla para continuar. . .
```

Figure 136: Birthday Season Output.

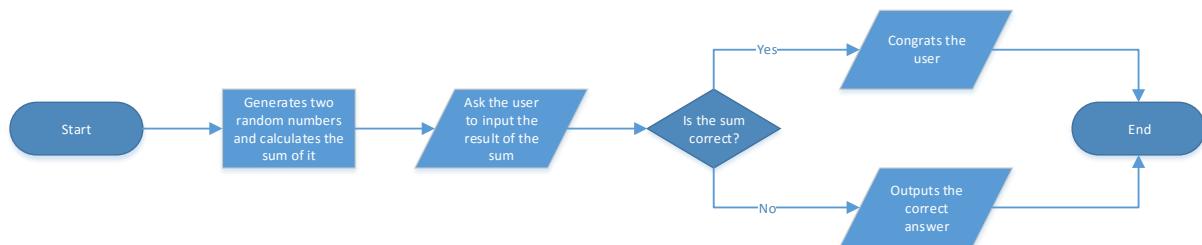
5.3.4 Exercise 4

This program should generate two whole numbers between 1 and 100, and ask the user to input the sum of the two numbers, check if the answer is correct and congrats the user or provide the correct result.

```
What is 24 + 56? 70
Wrong! 24 + 56 is 80
```

Figure 137: Math Quiz.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
    srand(time(0));

    int first = (rand() % 100) + 1;
    int second = (rand() % 100) + 2;

    int sum = first + second;

    int user;
```

```
printf("What is %d + %d? ", first, second);
scanf("%d", &user);

if (user == sum)
{
    printf("Correct!");
}
else
{
    printf("Wrong! %d + %d is %d", first, second, sum);
}

printf("\n\n");

return 0;
}
```

This code outputs:



The screenshot shows a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
What is 74 + 95? 12
Wrong! 74 + 95 is 169
Pressione qualquer tecla para continuar. . .

Figure 138: Math Quiz Output.

5.3.5 Exercise 5

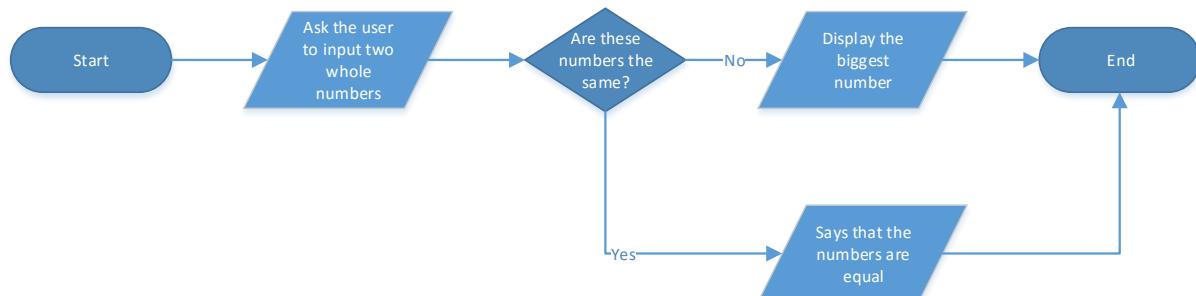
This program takes two numbers as input from the user and outputs the biggest number, or if the numbers are equal.

```
Please enter a number: 25
Please enter another number: 17

25 is bigger than 17.
```

Figure 139: Biggest Number of Two.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    int first;
    printf("Please enter a number: ");
    scanf("%d", &first);

    int second;
    printf("Please enter another number: ");
    scanf("%d", &second);

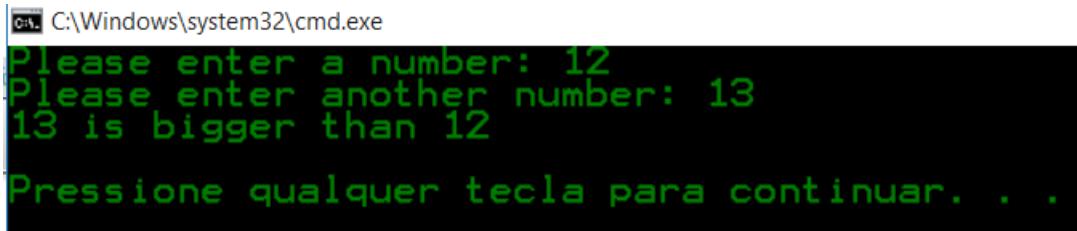
    if (first == second)
    {
```

```
        printf("The same number (%d) was entered twice.", first);
    }
else if (first > second)
{
    printf("%d is bigger than %d", first, second);
}
else
{
    printf("%d is bigger than %d", second, first);
}

printf("\n\n");

return 0;
}
```

This program outputs:



```
C:\Windows\system32\cmd.exe
Please enter a number: 12
Please enter another number: 13
13 is bigger than 12
Pressione qualquer tecla para continuar. . .
```

Figure 140: Biggest Number of Two Output.

5.3.6 Exercise 6

This program should take a whole number from the user to represent a purchase price at a retail store in dollars, apply the discount and output the final payable total.

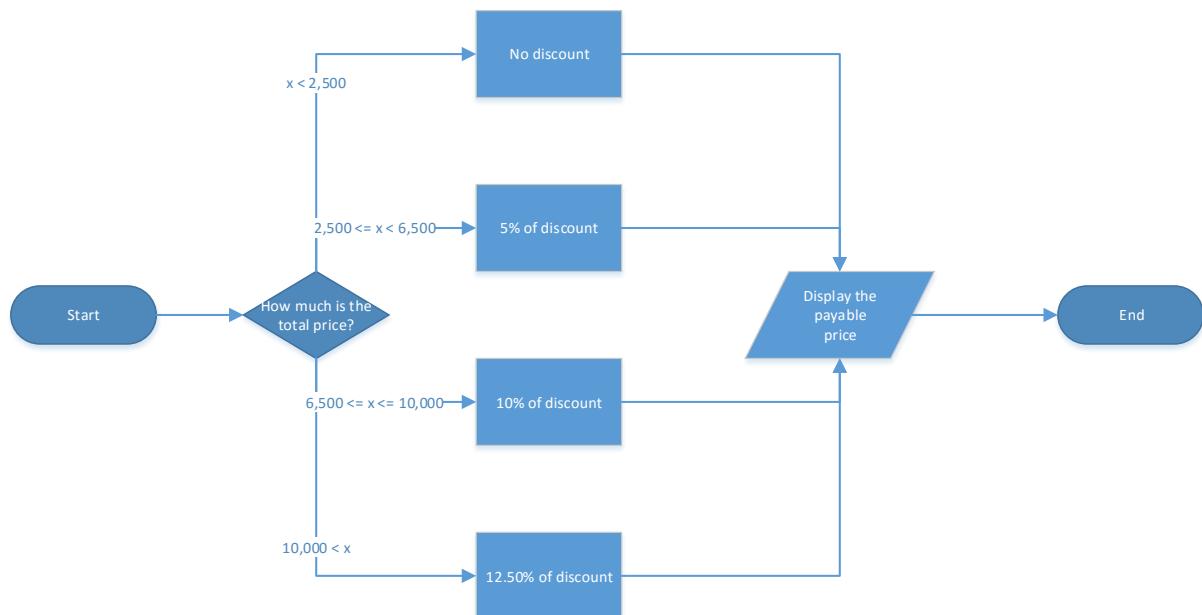
Purchase Amount:	Discount Percentage:
$x < 2500$	0.00%
$2500 \leq x < 6500$	5.00%
$6500 \leq x \leq 10000$	10.00%
$x > 10000$	12.50%

Figure 84: Discount Table.

```
Input the total purchase price: 5000
Discount is: 250
Payable Total is: 4750
```

Figure 141: Discount Percentage.

To develop this program, I wrote the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    int purchase;
    printf("Input the total purchase price: ");
    scanf("%d", &purchase);

    int discount;
    int total;

    if (purchase < 2500)
    {
        printf("Discount is: 0\n");
        printf("Payable Total is: %d", purchase);
    }
    else if (purchase >= 2500 && purchase < 6500)
    {
        discount = purchase * 0.05;
        total = purchase - discount;

        printf("Discount is: %d\n", discount);
        printf("Payable Total is: %d", total);
    }
    else if (purchase >= 6500 && purchase <= 10000)
    {
        discount = purchase * 0.1;
        total = purchase - discount;

        printf("Discount is: %d\n", discount);
        printf("Payable Total is: %d", total);
    }
    else if (purchase > 10000)
    {
        discount = purchase * 0.125;
        total = purchase - discount;

        printf("Discount is: %d\n", discount);
        printf("Payable Total is: %d", total);
    }

    printf("\n\n");
}
```

```
    return 0;  
}
```

This code has the following output:

The screenshot shows a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
Input the total purchase price: 8395
Discount is: 839
Payable Total is: 7556
Pressione qualquer tecla para continuar. . .

Figure 142: Discount Percentage Output.

5.3.7 Exercise 7

This program should take a percentage score from the user and output the equivalent letter grade result.

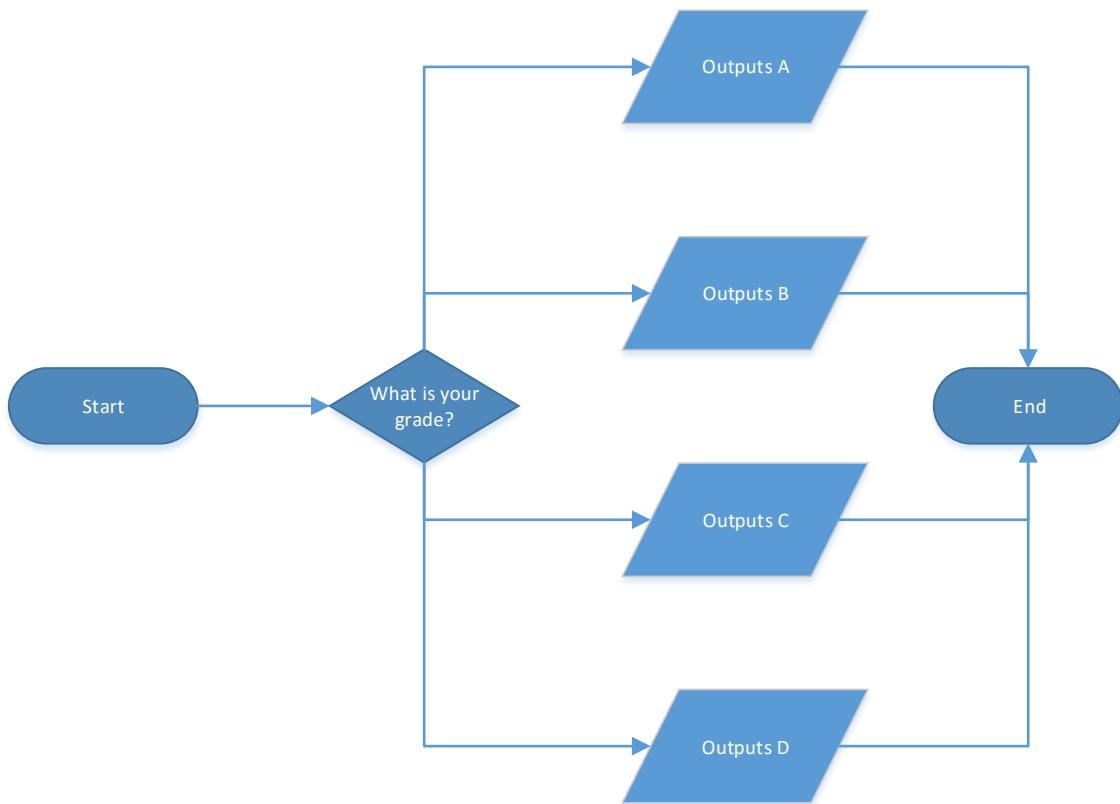
<i>Grade Boundaries:</i>			
<i>A Range:</i>	$A+ \geq 90\%$	$A \geq 85\%$	$A- \geq 80\%$
<i>B Range:</i>	$B+ \geq 75\%$	$B \geq 70\%$	$B- \geq 65\%$
<i>C Range:</i>	$C+ \geq 60\%$	$C \geq 55\%$	$C- \geq 50\%$
<i>D Range:</i>		$D < 50\%$	

Figure 143: Grades equivalency.

```
Please enter your Practical Test 1 percentage: 71.25
71.250000% is a 'B'
```

Figure 144: Letter Grade.

To write this code, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

int main()
{
    float grade;
    printf("Please enter your Practical Test 1 percentage: ");
    scanf("%f", &grade);

    if (grade < 50)
    {
        printf("%f is a 'D'", grade);
    }
    else if (grade >= 50 && grade < 55)
    {
        printf("%f is a 'C-'", grade);
    }
    else if (grade >= 55 && grade < 60)
    {
  
```

```
        printf("%f is a 'C'", grade);
    }
else if (grade >= 60 && grade < 65)
{
    printf("%f is a 'C+', grade);
}
else if (grade >= 65 && grade < 70)
{
    printf("%f is a 'B-', grade);
}
else if (grade >= 70 && grade < 75)
{
    printf("%f is a 'B'", grade);
}
else if (grade >= 75 && grade < 80)
{
    printf("%f is a 'B+', grade);
}
else if (grade >= 80 && grade < 85)
{
    printf("%f is a 'A-", );
}
else if (grade >= 85 && grade < 90)
{
    printf("%f is a 'A'");
}
else if (grade >= 90)
{
    printf("%f is a 'A+'");
}

printf("\n\n");

return 0;
}
```

This program outputs:

```
C:\Windows\system32\cmd.exe
Please enter your Practical Test 1 percentage: 75
75.000000 is a 'B+'
Pressione qualquer tecla para continuar. . .
```

Figure 145: Letter Grade Output.

5.3.8 Exercise 8

This program should ask the user to input a single character and output the type of the character, according to the table:

<i>ASCII Classification:</i>	<i>Low:</i>	<i>High:</i>
Non-Printable	0	31
Space	32	32
Symbol	33	47
Digit	48	57
Symbol	58	64
Uppercase	65	90
Symbol	91	96
Lowercase	97	122
Symbol	123	126
Non-Printable	127	127

Figure 146: ASCII Classification Table.

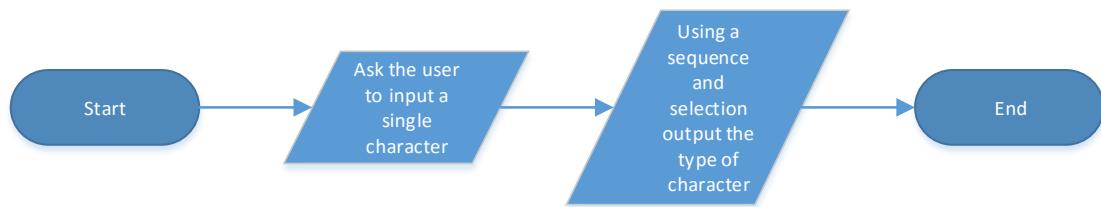
Output example:

```
Input a character: x
```

```
Input is lowercase.
```

Figure 147: ASCII Classifier.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

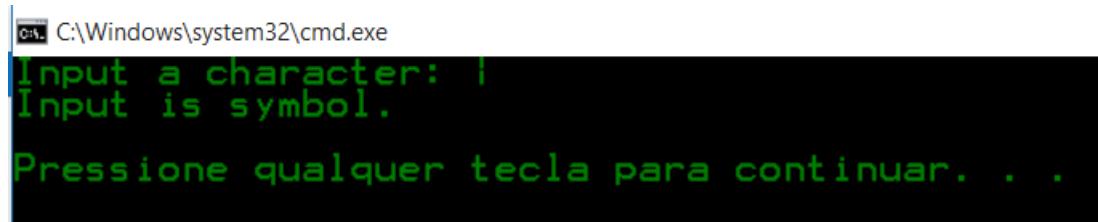
int main()
{
    char character;
    printf("Input a character: ");
    scanf("%c", &character);

    if (character <= 31 || character == 127)
    {
        printf("Input is non-printable.");
    }
    else if (character == 32)
    {
        printf("Input is space.");
    }
    else if (character >= 48 && character <= 57)
    {
        printf("Input is digit.");
    }
    else if (character >= 65 && character <= 90)
    {
        printf("Input is uppercase.");
    }
    else if (character >= 97 && character <= 122)
    {
        printf("Input is lowercase.");
    }
    else if ((character >= 33 && character <= 47) || (character >= 58 && character <= 64) || (character >= 91 && character <= 96) || (character >= 123 && character <= 126))
    {
        printf("Input is symbol.");
    }

    printf("\n\n");
}
  
```

```
    return 0;  
}
```

This program outputs:



A screenshot of a Windows command-line interface (cmd.exe) window. The window title is 'C:\Windows\system32\cmd.exe'. The text inside the window reads:
Input a character: I
Input is symbol.
Pressione qualquer tecla para continuar. . .

Figure 148: ASCII Classifier Output.

5.3.9 Exercise 9

This program takes three numbers as inputs from the user, then compute and output the numbers in order from the biggest to the smallest, as in the example:

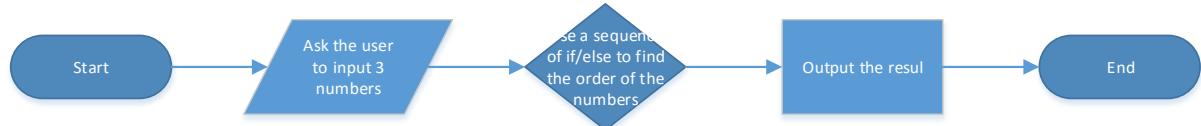
```
Please enter a number: 32
Please enter another number: 15
Please enter a third number: 28

The numbers, from biggest to smallest, are:

32 then 28 then 15
```

Figure 149: Biggest Number of Three.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    int first;
    printf("Please enter a number: ");
    scanf("%d", &first);

    int second;
    printf("Please enter another number: ");
    scanf("%d", &second);

    int third;
    printf("Please enter a third number: ");
    scanf("%d", &third);
```

```
printf("\n");

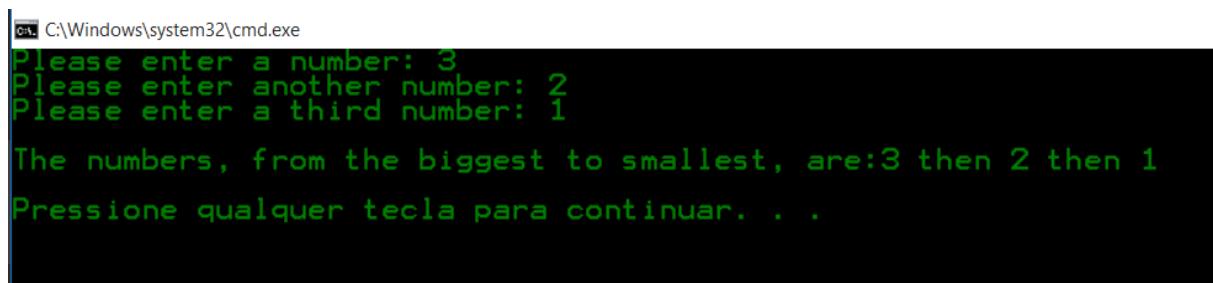
printf("The numbers, from the biggest to smallest, are:");

if (first > second && second > third)
{
    printf("%d then %d then %d", first, second, third);
}
else if (first > third && third > second)
{
    printf("%d then %d then %d", first, third, second);
}
else if (second > first && first > third)
{
    printf("%d then %d then %d", second, first, second);
}
else if (second > third && third > first)
{
    printf("%d then %d then %d", second, third, first);
}
else if (third > first && first > second)
{
    printf("%d then %d then %d", third, first, second);
}
else if (third > second && second > first)
{
    printf("%d then %d then %d", third, second, first);
}

printf("\n\n");

return 0;
}
```

This code outputs:



```
C:\Windows\system32\cmd.exe
Please enter a number: 3
Please enter another number: 2
Please enter a third number: 1
The numbers, from the biggest to smallest, are:3 then 2 then 1
Pressione qualquer tecla para continuar. . .
```

Figure 150: Biggest Number of Three Output.

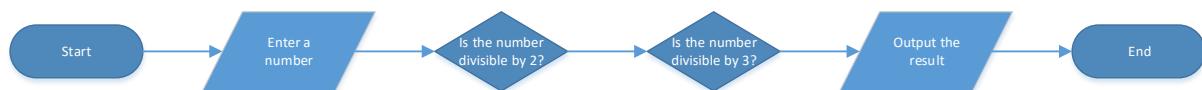
5.3.10 Exercise 10

This program should take a whole number from the user and output if this number is wholly divisible by 2 and/or 3.

```
Please input a whole number: 19
19 is not divisible by 2 or 3.
```

Figure 151: Divisible Question.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    int input;
    printf("Please input a whole number: ");
    scanf("%d", &input);
    printf("\n");

    int rest_2 = input % 2;
    int rest_3 = input % 3;

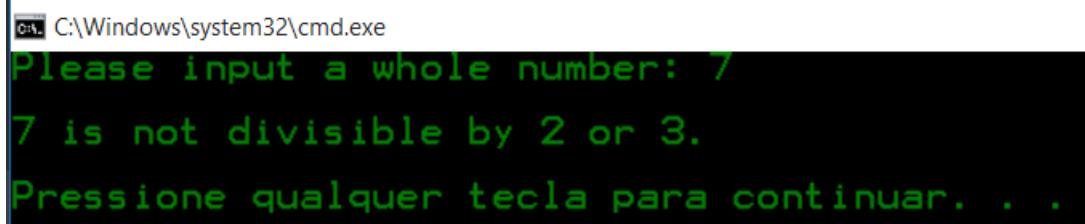
    if (rest_2 != 0 && rest_3 != 0)
    {
        printf("%d is not divisible by 2 or 3.", input);
    }
    else if (rest_2 == 0 && rest_3 != 0)
    {
        printf("%d is divisible by 2, but not by 3.", input);
    }
}
```

```
else if (rest_2 != 0 && rest_3 == 0)
{
    printf("%d is divisible by 3, but not by 2.", input);
}
else if (rest_2 == 0 && rest_3 == 0)
{
    printf("%d is divisible by 2 and 3.", input);
}

printf("\n\n");

return 0;
}
```

This program outputs:



```
C:\Windows\system32\cmd.exe
Please input a whole number: 7
7 is not divisible by 2 or 3.
Pressione qualquer tecla para continuar. . .
```

Figure 152: Divisible Question Output.

5.3.11 Exercise 11

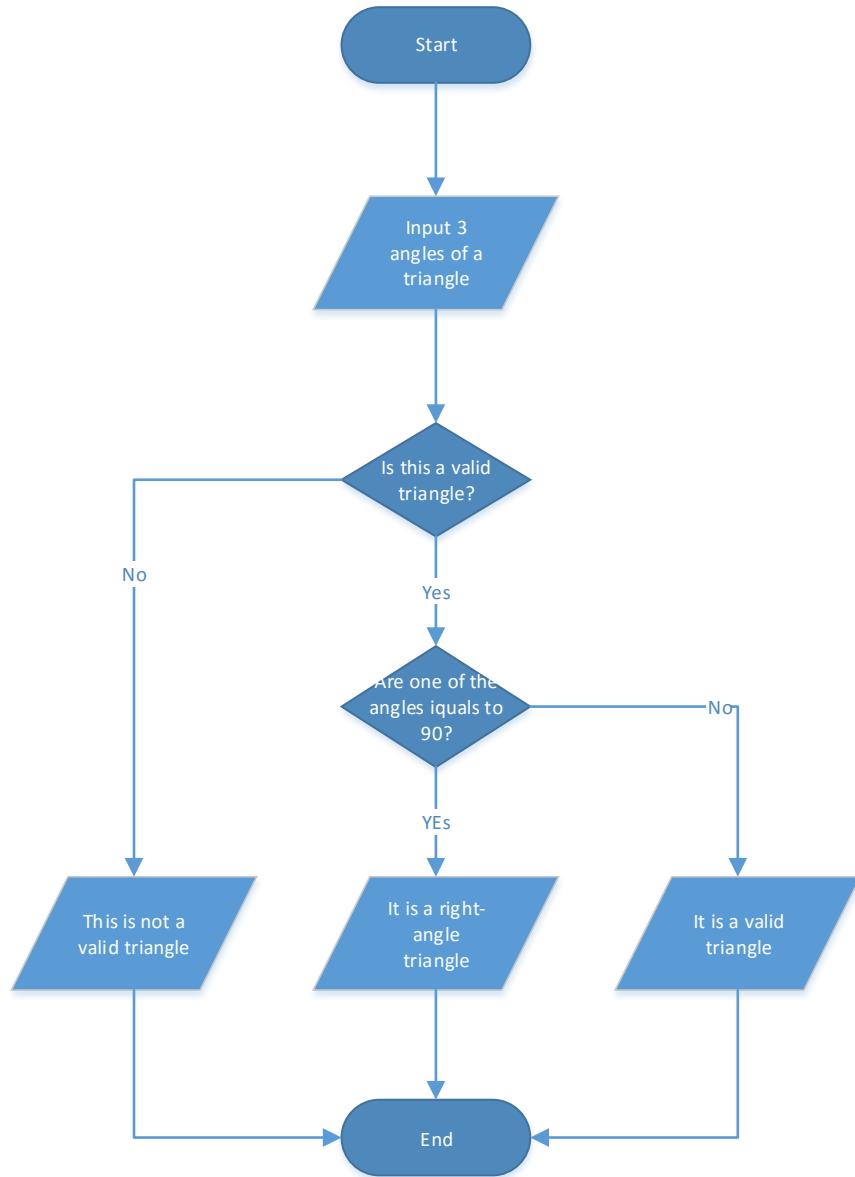
This program asks the user to input three angles of a triangle and outputs if the three angles create a valid triangle. If it is a valid triangle, checks if it is a right-angle triangle.

```
Enter the first angle (in degrees) : 85
Enter the second angle (in degrees) : 50
Enter the third angle (in degrees) : 45

85, 50, and 45 is a valid triangle.
```

Figure 153: Triangle Angle Checker.

To develop this program, I draw the following flowchart.



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

int main()
{
    int first;
    printf("Enter the first angle (in degrees): ");
    scanf("%d", &first);
  
```

```
int second;
printf("Enter the second angle (in degrees): ");
scanf("%d", &second);

int third;
printf("Enter the third angle (in degrees): ");
scanf("%d", &third);

printf("\n");

if (first + second + third == 180 && first != 0 && second != 0
&& third != 0)
{
    if (first == 90 || second == 90 || third == 90)
    {
        printf("%d, %d, and %d is a valid right-angle
triangle.", first, second, third);
    }
    else
    {
        printf("%d, %d, and %d is a valid triangle.", first,
second, third);
    }
}
else
{
    printf("%d, %d, and %d is not a valid triangle.", first,
second, third);
}

printf("\n\n");

return 0;
}
```

This code outputs:

```
C:\Windows\system32\cmd.exe
Enter the first angle (in degrees): 90
Enter the second angle (in degrees): 45
Enter the third angle (in degrees): 45
90, 45, and 45 is a valid right-angle triangle.
Pressione qualquer tecla para continuar. . .
```

Figure 154: Triangle Angle Checker.

5.3.12 Exercise 12

This program asks the user to input the humidity and temperature and gives a suggestion of an activity, as indicated by the table:

<i>Temperature:</i>	<i>Humidity:</i>	<i>Activity:</i>
Warm	Dry	Play tennis
Warm	Humid	Go swimming
Cold	Dry	Study Programming 1
Cold	Humid	Read a book

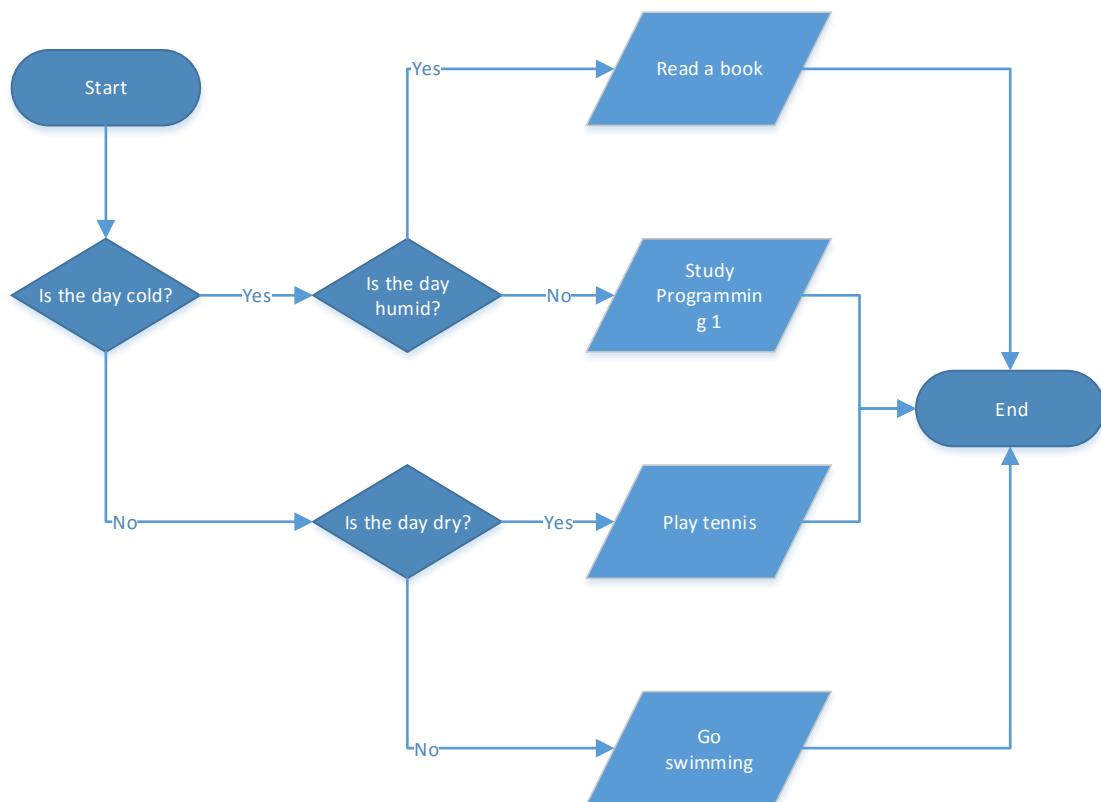
Figure 155: Activity table.

```
Is the temperature warm or cold (w/c)? w
Is it dry or humid (d/h)? d

You should play tennis.
```

Figure 156: Activity Decider.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

int main()
{
    char temperature;
    printf("Is the temperature warm or cold (w/c) ? ");
    scanf("%c", &temperature);

    char humidity;
    printf("Is it dry or humid (d/h) ? ");
    scanf("%c", &humidity);

    printf("\n\n");

    if (temperature == 'w')
    {
        if (humidity == 'd')
        {
  
```

```
        printf("You should play tennis.");
    }
    else if (humidity == 'h')
    {
        printf("You should go swimming.");
    }
    else
    {
        printf("I'm sorry Dave, I'm afraid I can't do
that.");
    }
}
else if (temperature == 'c')
{
    if (humidity == 'd')
    {
        printf("You should study Programming 1.");
    }
    else if (humidity == 'h')
    {
        printf("You should read a book.");
    }
    else
    {
        printf("I'm sorry Dave, I'm afraid I can't do
that.");
    }
}
else
{
    printf("I'm sorry Dave, I'm afraid I can't do that.");
}

printf("\n\n");

return 0;
}
```

But this code did not give me the expected output.

```
C:\Windows\system32\cmd.exe
Is the temperature warm or cold (w/c)? c
Is it dry or humid (d/h)?
I'm sorry Dave, I'm afraid I can't do that.
Pressione qualquer tecla para continuar. . .
```

Figure 157: Activity Decider Wrong Output.

Debugging this issue, I noticed that the program was taking “ENTER” as a char as well, so I put a sacrifice scanf. The final code was this:

```
#include <stdio.h>

int main()
{
    char temperature;
    printf("Is the temperature warm or cold (w/c)? ");
    scanf("%c", &temperature);

    char sacrifice;
    scanf("%c", &sacrifice);

    char humidity;
    printf("Is it dry or humid (d/h)? ");
    scanf("%c", &humidity);

    printf("\n");

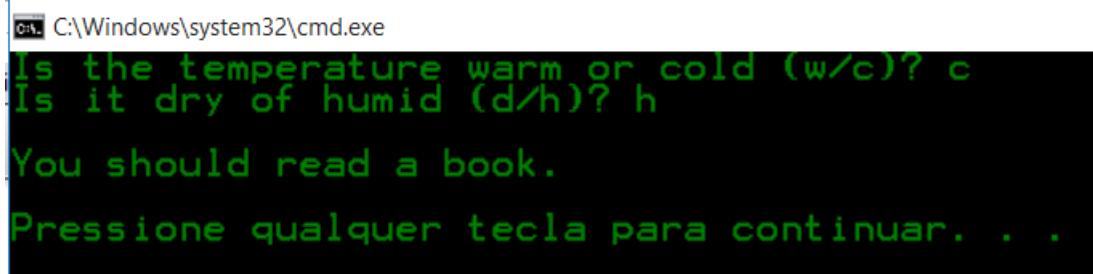
    if (temperature == 'w')
    {
        if (humidity == 'd')
        {
            printf("You should play tennis.");
        }
        else if (humidity == 'h')
        {
            printf("You should go swimming.");
        }
        else
        {
            printf("I'm sorry Dave, I'm afraid I can't do
that.");
        }
    }
}
```

```
        }
    }
else if (temperature == 'c')
{
    if (humidity == 'd')
    {
        printf("You should study Programming 1.");
    }
    else if (humidity == 'h')
    {
        printf("You should read a book.");
    }
    else
    {
        printf("I'm sorry Dave, I'm afraid I can't do
that.");
    }
}
else
{
    printf("I'm sorry Dave, I'm afraid I can't do that.");
}

printf("\n\n");

return 0;
}
```

This program outputs:



C:\Windows\system32\cmd.exe
Is the temperature warm or cold (w/c)? c
Is it dry or humid (d/h)? h
You should read a book.
Pressione qualquer tecla para continuar. . .

Figure 158: Activity Decider Output.

5.3.13 Exercise 13

This program generates a random number between 1 and 10 and gives three chances for the user to guess the number. If the guess is correct, the program outputs how many guesses the user needed to guess the number, else, the program tells the user which was the secret number.

```
I'm thinking of a number between 1 and 10...
You have three chances to guess the number...

What is your first guess? 10
Incorrect!

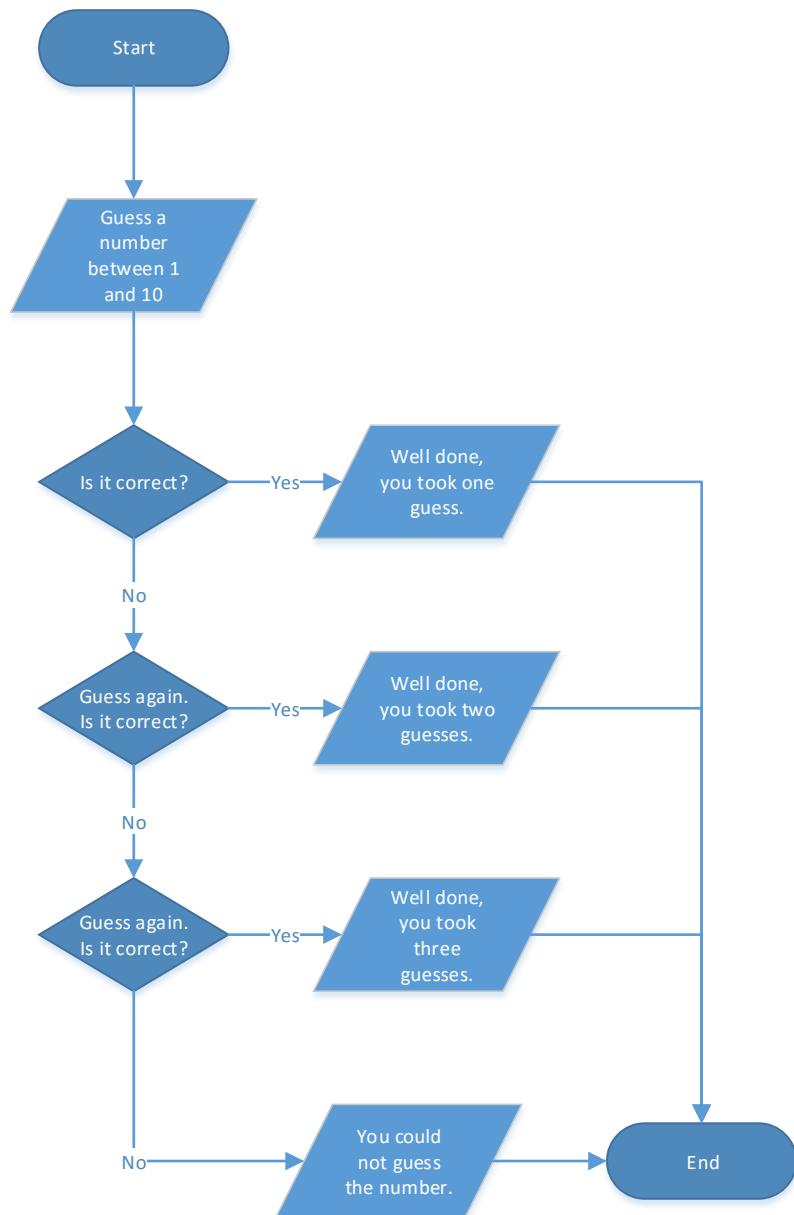
What is your second guess? 6
Incorrect!

What is your third guess? 3
Incorrect!

You could not guess my number! It was 7.
```

Figure 159: Simple Guessing Game.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
  
```

```
 srand(time(0));
int number = (rand() % 10) + 1;

printf("I'm thinking of a number between 1 and 10...\n");
printf("You have three chances to guess the number...");
printf("\n\n");

int guess;
printf("What is your first guess? ");
scanf("%d", &guess);

if (guess == number)
{
    printf("Correct!");
    printf("\n\n");
    printf("Well done, %d was my number! You took one
guess!", number);
}
else
{
    printf("Incorrect!");
    printf("\n\n");

    printf("What is your second guess? ");
    scanf("%d", &guess);
    if (guess == number)
    {
        printf("Correct!");
        printf("\n\n");
        printf("Well done, %d was my number! You took one
guess!", number);
    }
    else
    {
        printf("Incorrect!");
        printf("\n\n");

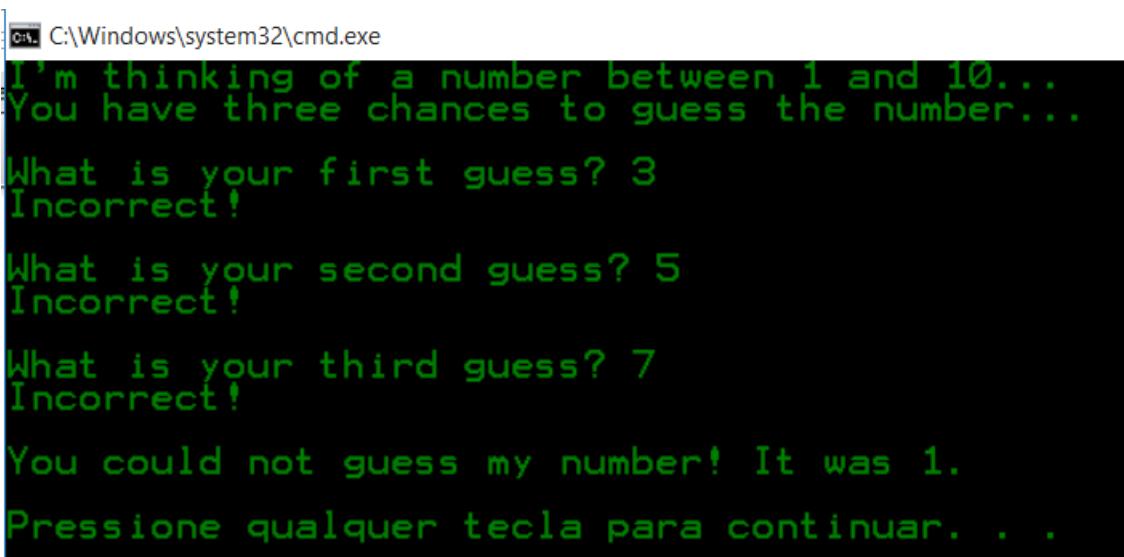
        printf("What is your third guess? ");
        scanf("%d", &guess);
        if (guess == number)
        {
            printf("Correct!");
            printf("\n\n");
            printf("Well done, %d was my number! You took
one guess!", number);
        }
        else
    }
}
```

```
        {
            printf("Incorrect!");
            printf("\n\n");
            printf("You could not guess my number! It was
%d.", number);
        }
    }

printf("\n\n");

return 0;
}
```

This code outputs:



C:\Windows\system32\cmd.exe

```
I'm thinking of a number between 1 and 10...
You have three chances to guess the number...

What is your first guess? 3
Incorrect!

What is your second guess? 5
Incorrect!

What is your third guess? 7
Incorrect!

You could not guess my number! It was 1.

Pressione qualquer tecla para continuar. . .
```

Figure 160: Simple Guessing Game Output.

To check if the program was working correctly, I opened the program with the debugger and tried a variety of answers.

5.3.14 Exercise 14

This program asks the user to input the length of three sides of a triangle, then classifies the triangle according to this table:

Type of triangle:	Properties:
Equilateral triangle	All sides are equal length
Isosceles triangle	Two sides are equal length
Scalene triangle	None of the sides are equal length

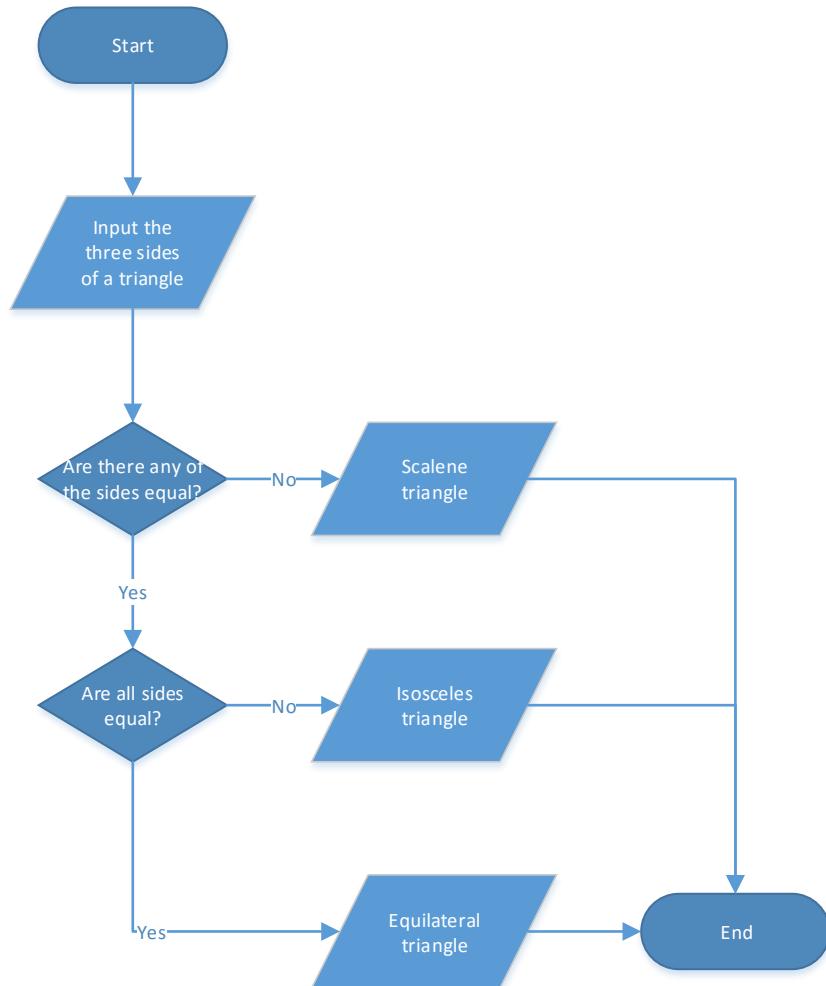
Figure 161: Triangle Table.

```
Enter the first side length: 40
Enter the second side length: 40
Enter the third side length: 40

This is an equilateral triangle.
```

Figure 162: Triangle Side Checker.

To write this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

int main()
{
    int first;
    printf("Enter the first side length: ");
    scanf("%d", &first);

    int second;
    printf("Enter the second side length: ");
    scanf("%d", &second);

    int third;
  
```

```
printf("Enter the third side length: ");
scanf("%d", &third);

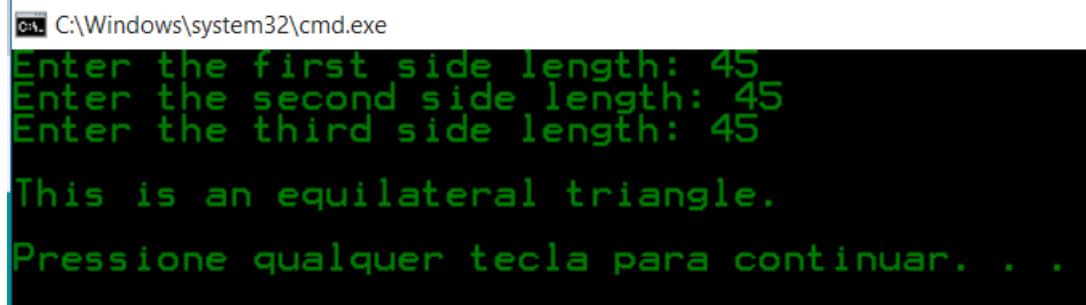
printf("\n");

if (first == second && second == third)
{
    printf("This is an equilateral triangle.");
}
else if (first == second || first == third || second == third)
{
    printf("This is an isosceles triangle.");
}
else
{
    printf("This is a scalene triangle.");
}

printf("\n\n");

return 0;
}
```

This program outputs:



```
C:\Windows\system32\cmd.exe
Enter the first side length: 45
Enter the second side length: 45
Enter the third side length: 45
This is an equilateral triangle.
Pressione qualquer tecla para continuar. . .
```

Figure 163: Triangle Side Checker Output.

5.3.15 Exercise 15

The program takes input for the three letters of a quadratic equation, if the discriminant is positive, outputs two real numbers; if it is zero, outputs the root; if it is negative, outputs two complex numbers.

```
Given a quadratic in the form: ax^2 + bx + c = 0
```

```
Enter the value of a: 5  
Enter the value of b: 6  
Enter the value of c: 1
```

```
Solving the quadratic:
```

```
5.000000x^2 + 6.000000x + 1.000000 = 0
```

```
Two real roots found...
```

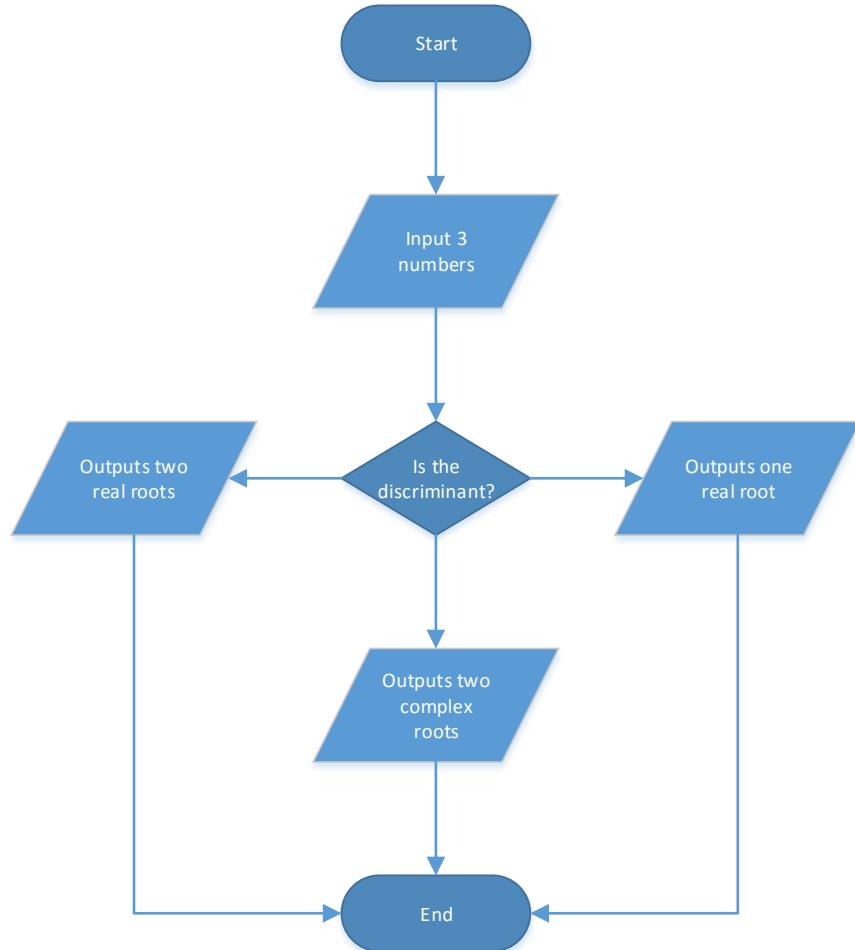
```
x is: -0.200000
```

```
Or...
```

```
x is: -1.000000
```

Figure 164: Finding Quadratic Root.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

int main()
{
    printf("Given a quadratic in the form: ax^2 + bx + c = 0");
    printf("\n\n");

    float a;
    printf("Enter the value of a: ");
    scanf("%f", &a);

    float b;
    printf("Enter the value of b: ");
    scanf("%f", &b);
  
```

```
float c;
printf("Enter the value of c: ");
scanf("%f", &c);

printf("\n\n");

printf("Solving the quadratic: ");
printf("\n\n");

printf("  %fx^2 + %fx + %f = 0", a, b, c);
printf("\n\n");

float discriminant = pow(b, 2) - 4 * a * c;

float result1;
float result2;

if (discriminant > 0)
{
    result1 = (-b + sqrt(discriminant)) / (2 * a);
    result2 = (-b - sqrt(discriminant)) / (2 * a);

    printf("Two real roots found...");
    printf("\n\n");

    printf("x is: %f: ", result1);
    printf("\n\n");

    printf("Or... ");
    printf("\n\n");
}

printf("x is: %f", result2);
}
else if (discriminant == 0)
{
    result1 = (-b) / (2 * a);

    printf("One real root found...");
    printf("\n\n");

    printf("x is: %f: ", result1);
    printf("\n\n");
}
else if (discriminant < 0)
{
    result1 = (-b) / (2 * a);
```

```
result2 = sqrt(pow(b, 2) - 4 * a * c);

printf("Two complex roots found...");
printf("\n\n");

printf("x is: %f + %fi: ", result1, result2);
printf("\n\n");

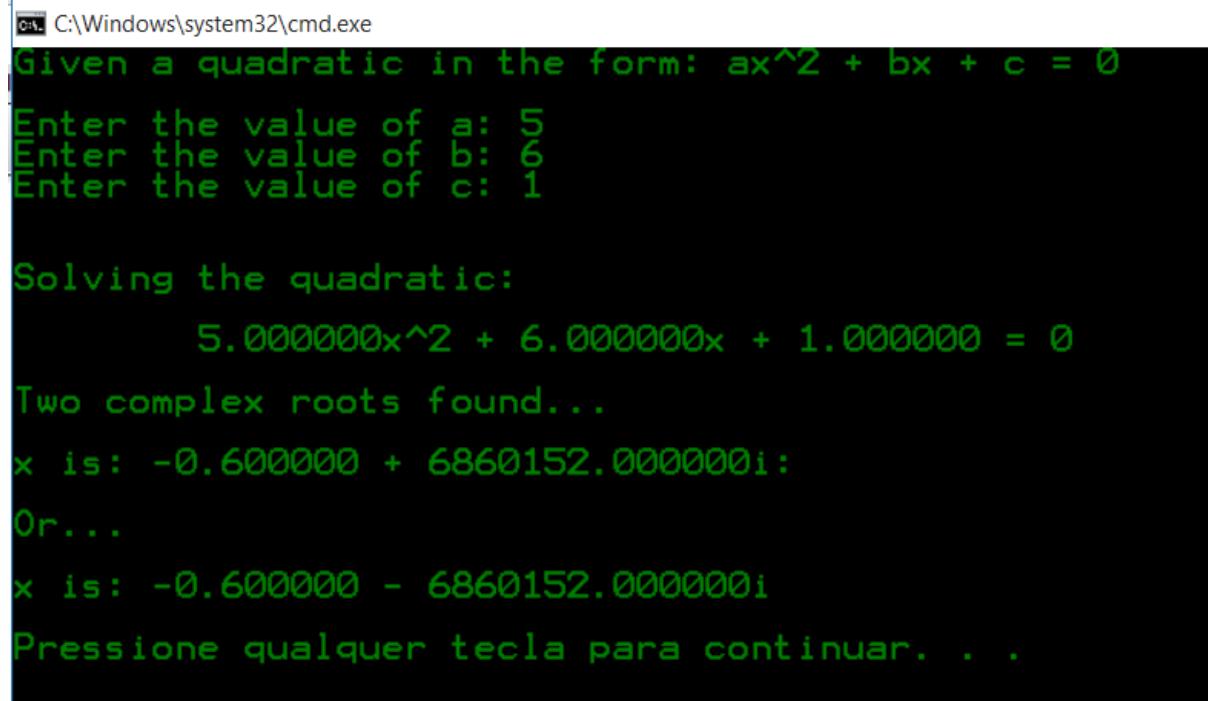
printf("Or...");
printf("\n\n");

printf("x is: %f - %fi", result1, result2);
}

printf("\n\n");

return 0;
}
```

But the output was not as expected:



C:\Windows\system32\cmd.exe

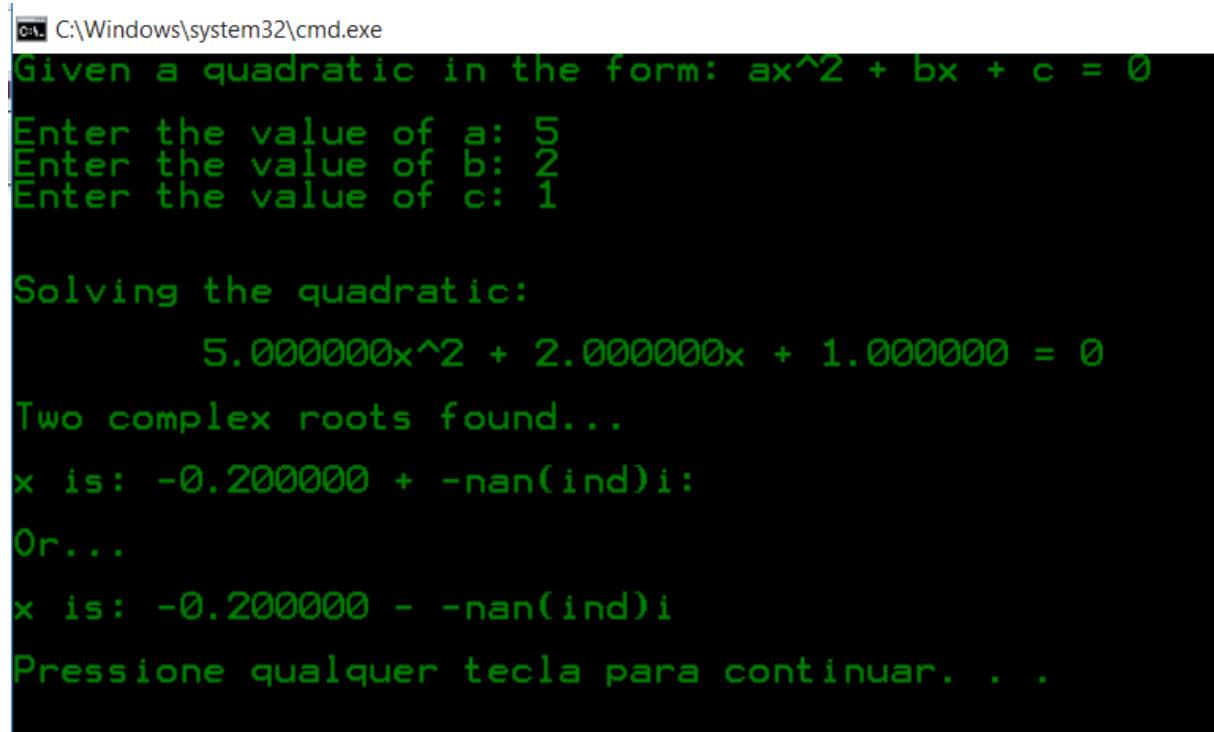
```
Given a quadratic in the form: ax^2 + bx + c = 0
Enter the value of a: 5
Enter the value of b: 6
Enter the value of c: 1

Solving the quadratic:
5.000000x^2 + 6.000000x + 1.000000 = 0
Two complex roots found...
x is: -0.600000 + 6860152.000000i:
Or...
x is: -0.600000 - 6860152.000000i
Pressione qualquer tecla para continuar. . .
```

Figure 165: Finding Quadratic Root Wrong Output.

I decided to include a function to print the result of the discriminant and debug it, I found out that there was something wrong in the discriminant function in my code. It took me a long time, and a lot of different testes, to realize that the problem was that I had forgotten to include the math library.

But after solving that, I noticed a problem with the complex function:



C:\Windows\system32\cmd.exe
Given a quadratic in the form: ax^2 + bx + c = 0
Enter the value of a: 5
Enter the value of b: 2
Enter the value of c: 1

Solving the quadratic:
5.000000x^2 + 2.000000x + 1.000000 = 0
Two complex roots found...
x is: -0.200000 + -nan(ind)i:
Or...
x is: -0.200000 - -nan(ind)i
Pressione qualquer tecla para continuar. . .

Figure 166: Finding Quadratic Root Another Wrong Output.

To solve this problem, I multiplied the operation of the complex square root by -1 , turning it in a positive number. My final code was:

```
#include <stdio.h>
#include <math.h>

int main()
{
    printf("Given a quadratic in the form: ax^2 + bx + c = 0");
    Page 267 of 688
Paludo
```

```
printf("\n\n");

float a;
printf("Enter the value of a: ");
scanf("%f", &a);

float b;
printf("Enter the value of b: ");
scanf("%f", &b);

float c;
printf("Enter the value of c: ");
scanf("%f", &c);

printf("\n\n");

printf("Solving the quadratic: ");
printf("\n\n");

printf(" %fx^2 + %fx + %f = 0", a, b, c);
printf("\n\n");

float discriminant = pow(b, 2) -(4 * a * c);

float result1;
float result2;

if (discriminant > 0)
{
    result1 = (-b + sqrt(discriminant)) / (2 * a);
    result2 = (-b - sqrt(discriminant)) / (2 * a);

    printf("Two real roots found... ");
    printf("\n\n");

    printf("x is: %f: ", result1);
    printf("\n\n");

    printf("Or... ");
    printf("\n\n");
}

printf("x is: %f", result2);
}
else if (discriminant == 0)
{
    result1 = (-b) / (2 * a);
```

```
printf("One real root found...") ;
printf("\n\n");

printf("x is: %f: ", result1) ;
printf("\n\n");
}

else if (discriminant < 0)
{
    result1 = (-b) / (2 * a);
    result2 = sqrt((pow(b, 2) - 4 * a * c) * -1);

    printf("Two complex roots found...") ;
    printf("\n\n");

    printf("x is: %f + %fi: ", result1, result2) ;
    printf("\n\n");

    printf("x is: %f - %fi", result1, result2);
}

printf("\n\n");
return 0;
}
```

This program outputs:

```
C:\Windows\system32\cmd.exe
Given a quadratic in the form: ax^2 + bx + c = 0
Enter the value of a: 49
Enter the value of b: 87
Enter the value of c: 55

Solving the quadratic:
    49.000000x^2 + 87.000000x + 55.000000 = 0
Two complex roots found...
x is: -0.887755 + 56.665688i:
Or...
x is: -0.887755 - 56.665688i
Pressione qualquer tecla para continuar. . .
```

Figure 169: Finding Quadratic Root Output.

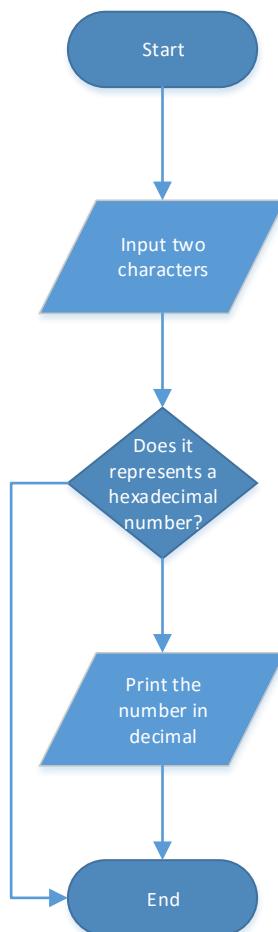
5.3.16 Exercise 16

This program should take two characters from the user, print the corresponding value of these characters when interpreted as a two-digit hexadecimal number.

```
Input a two-digit hexadecimal number: 0A  
0A (Base-16) is 10 (Base-10)
```

Figure 170: char to Hexadecimal.

To write this program, I draw the following flowchart:



It was extremely hard for me to figure out how to write this code, I asked a friend to send me a code that he had written, but it was too complex for me to understand. Researching on the internet, I found a code really easy to understand (Ahmad, n. d.):

```
/*
* Title : Convert Hexadecimal to Decimal(Hexadecimal
to Decimal.c)
* Program Description : Write a C Program to Convert
* Hexadecimal Number to Decimal Number.
* Author : robustprogramming.com
* Interface : Console
* IDE : Code::Blocks 13.12
* Operating System : Windows 8.1
*/
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

unsigned long convertToDecimal(char hex[]);

int main()
{
    char hex[9];// 8 characters for 32-bit Hexadecimal Number and
one for '\0'.
    unsigned long decimalNumber;
    printf(" C Program to Convert Hexadecimal Number to Decimal
Number \n");
    printf(" Enter 32-bit Hexadecimal Number : ");
    scanf("%s", hex);

    decimalNumber = convertToDecimal(hex);
    printf(" Decimal Number is %u \n", decimalNumber);

    printf(" Press enter to continue... \n");
    fflush(stdin);
    getchar();
    return 0;
}

unsigned long convertToDecimal(char hex[])
{
    char *hexString;
    int length = 0;
```

```

const int base = 16; // Base of Hexadecimal Number
unsigned long decimalNumber = 0;
int i;
// Find length of Hexadecimal Number
for (hexString = hex; *hexString != '\0'; hexString++)
{
    length++;
}
// Find Hexadecimal Number
hexString = hex;
for (i = 0; *hexString != '\0' && i < length; i++, hexString++)
{
    // Compare *hexString with ASCII values
    if (*hexString >= 48 && *hexString <= 57) // is *hexString
Between 0-9
    {
        decimalNumber += (((int)(*hexString)) - 48) * pow(base,
length - i - 1);
    }
    else if ((*hexString >= 65 && *hexString <= 70)) // is
*hexString Between A-F
    {
        decimalNumber += (((int)(*hexString)) - 55) * pow(base,
length - i - 1);
    }
    else if ((*hexString >= 97 && *hexString <= 102) // is
*hexString Between a-f
    {
        decimalNumber += (((int)(*hexString)) - 87) * pow(base,
length - i - 1);
    }
    else
    {
        printf(" Invalid Hexadecimal Number \n");

        printf(" Press enter to continue... \n");
        fflush(stdin);
        getchar();
        return 0;
        exit(0);
    }
}
return decimalNumber;
}

```

Using my flowchart, and this code, I wrote my code. This program has high efficiency variables and a very well structured programming using concepts of Object Oriented
 Paludo

Programming (OOB), creating method for more complex functions and returning the result of this method, I decided to take the same strategy. It took me really long to understand how this code works, I asked help from a lot of friends, researched on books and on the internet. Then, I wrote the following code, this code is stored in two files, one for the main method, and another file for the convert method:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

unsigned long convert_method(char hex[]);

int main()
{
    char hex[3];
    printf("Input a two-digit hexadecimal number: ");
    scanf("%2s", hex);

    printf("\n\n");

    unsigned long decimal_number = convert_method(hex);
    printf("%s (Base-16) is %u (Base10)", hex, decimal_number);

    printf("\n\n");

    return 0;
}
unsigned long convert_method(char hex[])
{
    char *hex_string;
    static int LENGTH = 2;
    const int BASE = 16;
    unsigned long decimal_number = 0;

    hex_string = hex;

    for (int i = 0; i < 2; i++, hex_string++)
    {
        if (*hex_string >= 48 && *hex_string <= 57)
```

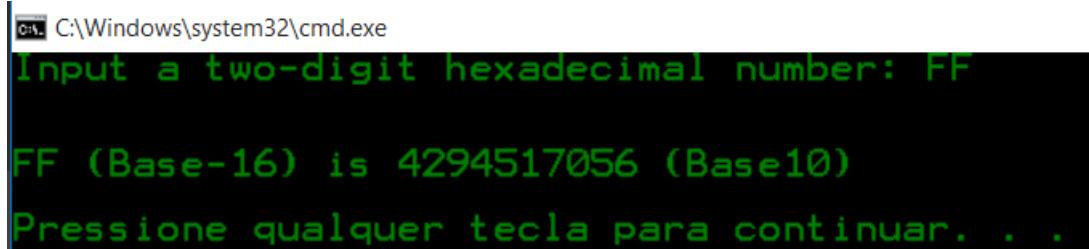
```

    {
        decimal_number += (((int)(*hex_string)) - 48) *
pow(BASE, LENGTH - i - 1);
    }
    else if (*hex_string >= 65 && *hex_string <= 70)
    {
        decimal_number += (((int)(*hex_string)) - 55) *
pow(BASE, LENGTH - i - 1);
    }
    else if (*hex_string >= 97 && *hex_string <= 102)
    {
        decimal_number += (((int)(*hex_string)) - 87) *
pow(BASE, LENGTH - i - 1);
    }
    else
    {
        printf("Input is not a hexadecimal number!");

        return 0;
    }
}
return decimal_number;
}

```

But this output did not have the expected output, it outputs:



The screenshot shows a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The user has typed 'Input a two-digit hexadecimal number: FF' and pressed Enter. The program then outputs 'FF (Base-16) is 4294517056 (Base10)' followed by a prompt to press any key to continue.

Figure 171: char to Hexadecimal Unexpected Output.

To fix this problem, I decided to take the code from the internet, and make small changes to better understand it, until I can write my own code with similar efficiency.

Doing a test, I related that Ahmad's (n. d.) code does work properly:

```
C:\Windows\system32\cmd.exe
C Program to Convert Hexadecimal Number to Decimal Number
Enter 32-bit Hexadecimal Number : FF
Decimal Number is 255
Press enter to continue...
Pressione qualquer tecla para continuar. . .
```

Figure 172: Ahmad's Code Output.

I realized that the problem might be because I was using two different files without using all the recommended OOB programming methods, and this was causing the program to have an unexpected output. This problem can be solved either putting the two methods in the same file, or importing all the libraries in both files, I decided to take the second option. The written code is the following one:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main()
{
    char hex[3];
    printf("Input a two-digit hexadecimal number: ");
    scanf("%2s", hex);

    printf("\n");

    unsigned long decimal_number = convert_method(hex);
    printf("%s (Base-16) is %u (Base10)", hex, decimal_number);

    printf("\n\n");

    return 0;
}

unsigned long convert_method(char hex[])
{
    char *hex_string = hex;
    static int LENGTH = 2;
    static const int BASE = 16;
    unsigned long decimal_number = 0;
```

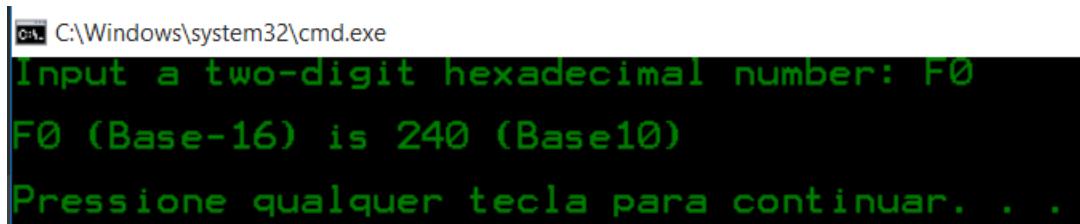
```

for (int i = 0; i < 2; i++, hex_string++)
{
    if (*hex_string >= 48 && *hex_string <= 57)
    {
        decimal_number += (((int)(*hex_string)) - 48) *
pow(BASE, LENGTH - i - 1);
    }
    else if (*hex_string >= 65 && *hex_string <= 70)
    {
        decimal_number += (((int)(*hex_string)) - 55) *
pow(BASE, LENGTH - i - 1);
    }
    else
    {
        printf("Input is not a hexadecimal number!");

        return 0;
    }
}
return decimal_number;
}

```

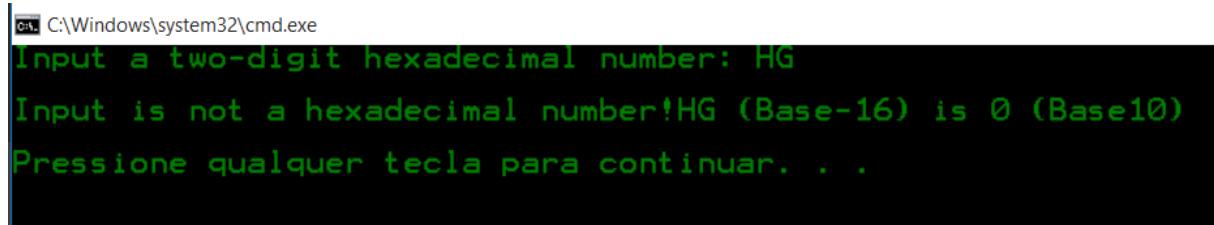
This code has the correct output when the user inputs a valid hexadecimal number, with correct characters:



C:\Windows\system32\cmd.exe
Input a two-digit hexadecimal number: F0
F0 (Base-16) is 240 (Base10)
Pressione qualquer tecla para continuar. . .

Figure 173: char to Hexadecimal Output.

But when the input is not valid, the output is not exactly as expected:



C:\Windows\system32\cmd.exe
Input a two-digit hexadecimal number: HG
Input is not a hexadecimal number!HG (Base-16) is 0 (Base10)
Pressione qualquer tecla para continuar. . .

Figure 174: char to Hexadecimal Non-Hexadecimal Wrong Output.

To fix this, I took the structure responsible for stopping the program out from the converting method, and put it in the main method, with an if/else function inside a for function. The for function does the operation for the two characters, and the inside function proceeds the program if the character is a valid hexadecimal number, and stops the program and outputs a message if is not. The written code was this:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main()
{
    char hex[3];
    printf("Input a two-digit hexadecimal number: ");
    scanf("%2s", hex);

    printf("\n");

    unsigned long decimal_number = convert_method(hex);

    for (int n = 0; n < 2; n++)
    {
        if ((hex[n] >= 48 && hex[n] <= 57) || (hex[n] >= 65 &&
hex[n] <= 70))
        {
        }
        else
        {
            printf("Input is not a hexadecimal number!");

            printf("\n\n");
            return 0;
        }
    }

    printf("%s (Base-16) is %u (Base10)", hex, decimal_number);

    printf("\n\n");
    return 0;
}
```

```

unsigned long convert_method(char hex[])
{
    char *hex_string = hex;
    static int LENGTH = 2;
    static const int BASE = 16;
    unsigned long decimal_number = 0;

    for (int i = 0; i < 2; i++, hex_string++)
    {
        if (*hex_string >= 48 && *hex_string <= 57)
        {
            decimal_number += (((int)(*hex_string)) - 48) *
pow(BASE, LENGTH - i - 1);
        }
        else if (*hex_string >= 65 && *hex_string <= 70)
        {
            decimal_number += (((int)(*hex_string)) - 55) *
pow(BASE, LENGTH - i - 1);
        }
    }
    return decimal_number;
}

```

This code has the following output:

```

C:\Windows\system32\cmd.exe
Input a two-digit hexadecimal number: F0
F0 (Base-16) is 240 (Base10)F0 (Base-16) is 240 (Base10)
Pressione qualquer tecla para continuar. . .

```

Figure 175: char to Hexadecimal Non-Hexadecimal Output.

Talking to a friend, I was advised that the approach I took to stop the program in case of a wrong input was not the best one to take, then I wrote the following code:

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main()
{
    char hex[3];
    printf("Input a two-digit hexadecimal number: ");
    scanf("%2s", hex);

```

```

printf("\n");

static int LENGTH = 2;
for (int n = 0; n < LENGTH; n++)
{
    if ((hex[n] < 48) || (hex[n] > 57 && hex[n] < 65) ||
(hex[n] > 70))
    {
        printf("Input is not a hexadecimal number!");
        printf("\n\n");

        return 0;
    }
}

unsigned long decimal_number = convert_method(hex);
printf("%s (Base-16) is %u (Base10)", hex, decimal_number);

printf("\n\n");

return 0;
}
unsigned long convert_method(char hex[])
{
    char *hex_string = hex;
    static int LENGTH = 2;
    static const int BASE = 16;
    unsigned long decimal_number = 0;

    for (int i = 0; i < LENGTH; i++, hex_string++)
    {
        if (*hex_string >= 48 && *hex_string <= 57)
        {
            decimal_number += (((int)(*hex_string)) - 48) *
pow(BASE, LENGTH - i - 1);
        }
        else if (*hex_string >= 65 && *hex_string <= 70)
        {
            decimal_number += (((int)(*hex_string)) - 55) *
pow(BASE, LENGTH - i - 1);
        }
    }
    return decimal_number;
}

```

This code has the same output, but is better structured. I decided to go ahead and try something else. As I only have two characters, I will try to take out the for structure:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main()
{
    char hex[3];
    printf("Input a two-digit hexadecimal number: ");
    scanf("%2s", hex);

    printf("\n");

    if (((hex[0] | hex[1]) < 48) || ((hex[0] | hex[1]) > 57 &&
(hex[0] | hex[1]) < 65) || ((hex[0] | hex[1]) > 70))
    {
        printf("Input is not a hexadecimal number!");
        printf("\n\n");

        return 0;
    }

    unsigned long decimal_number = convert_method(hex);
    printf("%s (Base-16) is %u (Base10)", hex, decimal_number);

    printf("\n\n");

    return 0;
}

unsigned long convert_method(char hex[])
{
    char *hex_string = hex;
    static int LENGTH = 2;
    static const int BASE = 16;
    unsigned long decimal_number = 0;

    for (int i = 0; i < LENGTH; i++, hex_string++)
    {
        if (*hex_string >= 48 && *hex_string <= 57)
```

```
{  
    decimal_number += (((int)(*hex_string)) - 48) *  
pow(BASE, LENGTH - i - 1);  
}  
else if (*hex_string >= 65 && *hex_string <= 70)  
{  
    decimal_number += (((int)(*hex_string)) - 55) *  
pow(BASE, LENGTH - i - 1);  
}  
}  
return decimal_number;  
}
```

6 Week Six

6.1 Lectures

This week we studied repetition methods, these methods are used to save the programmer re-typing the same sequence a lot of times. A selection function will take an action if something is true, a repetition function will take an action while something is true, it can happen only once or more than one time. The difference between the while loop and the do while loop is that the while loop may not be executed, whereas the do while loop will run at least one time.

We also saw an nested if inside a while that stops the program in a situation using break. The continue keyword can be used instead of an empty statement, just to make the code clear.

The goto keyword is used to return or go forward to a point in the code that has a label, it is not recommended to use it because it makes the code hard to understand and it will break the logical flow of the program.

The for loop repeats a statement, take an action, and keeps running until it reaches the condition.

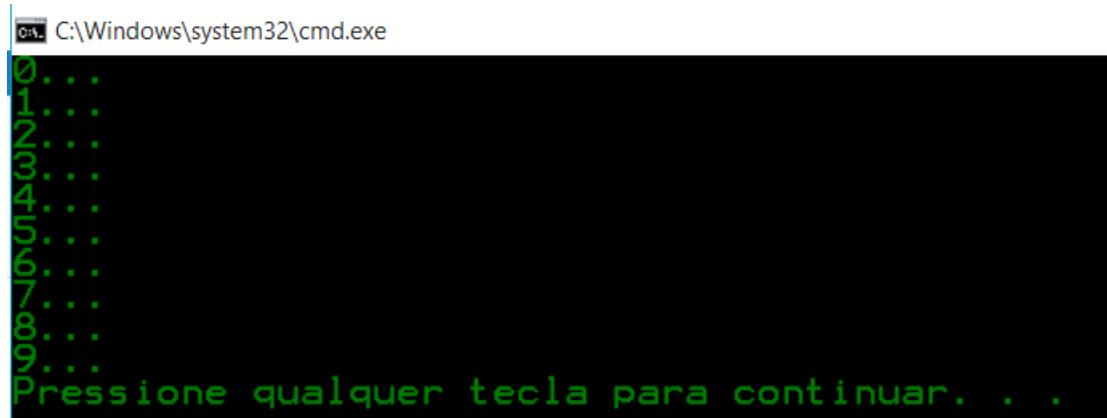
Bitwise operators are extremely important in programming, they operate using binary logic. They are: and, or, exclusive or, not.

The enumeration classes are used to enumerate lists with determined values. They are very useful to make the code easy to understand.

6.2 Examples

6.2.1 Example 1

This example is an example of the use of simple while loops. The program prints the number that the counter holds and adds one to the counter, repeat until counter equal to ten. The program outputs:



```
C:\Windows\system32\cmd.exe
0...
1...
2...
3...
4...
5...
6...
7...
8...
9...
Pressione qualquer tecla para continuar. . .
```

Figure 176: Simple while Loop.

This was the written code:

```
#include <stdio.h>

int main()
{
    int counter = 0;

    while (counter < 10)
    {
        printf("%d... \n", counter);

        ++counter;
    }
}
```

```
    return 0;  
}
```

6.2.2 Example 2

This program demonstrates how the do while loops work. The action is taken at least once. The program outputs:



```
C:\Windows\system32\cmd.exe
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, Pressione qualquer tecla para continuar. . .
```

Figure 177: Simple do while Loop.

This was the written code:

```
#include <stdio.h>

int main()
{
    int counter = 10;

    do
    {
        printf("%d, ", counter);

        --counter;
    }
    while (counter > 0);

    return 0;
}
```

6.2.3 Example 3

This program gives an example of how the for structures work. The program starts a variable as zero, and adds one to this variable until it reaches ten. The program outputs the number that the variable holds every repetition, outputting:



The screenshot shows a Windows command prompt window titled 'cmd C:\Windows\system32\cmd.exe'. The window contains the text: '0... 1... 2... 3... 4... 5... 6... 7... 8... 9... Pressione qualquer tecla para continuar. . .' (Press any key to continue. .).

Figure 178: Simple for Loop.

This was the written code:

```
#include <stdio.h>

int main()
{
    for (int i = 0; i < 10; ++i)
    {
        printf("%d... ", i);
    }

    return 0;
}
```

6.2.4 Example 4

This code uses an if statement nested inside a for structure to print the ASCII table printing ten elements per line with its respective position numbers. The program outputs:



```

32 ! 33 " 34 # 35 $ 36 % 37 & 38 ' 39 ( 40 ) 41
* 42 + 43 , 44 - 45 . 46 / 47 0 48 1 49 2 50 3 51
4 52 53 6 54 7 55 8 56 9 57 : 58 ; 59 < 60 = 61
> 62 ? 63 @ 64 A 65 B 66 C 67 D 68 E 69 F 70 G 71
H 72 I 73 J 74 K 75 L 76 M 77 N 78 O 79 P 80 Q 81
R 82 S 83 T 84 U 85 V 86 W 87 X 88 Y 89 Z 90 [ 91
\ 92 ] 93 ^ 94 _ 95 ` 96 a 97 b 98 c 99 d 100 e 101
f 102 g 103 h 104 i 105 j 106 k 107 l 108 m 109 n 110 o 111
p 112 q 113 r 114 s 115 t 116 u 117 v 118 w 119 x 120 y 121
z 122 { 123 | 124 } 125 ~ 126 ^ 127 Pressione qualquer tecla para continuar...

```

Figure 179: ASCII Table Printer.

This was the written code:

```

#include <stdio.h>

int main()
{
    int column_count = 0;
    unsigned char ascii = 0;

    for (ascii = ' '; ascii < 128; ++ascii)
    {
        printf("%c ", ascii);

        if (ascii < 100)
        {
            printf(" ");
        }

        printf("%d ", ascii);

        ++column_count;

        if (column_count >= 10)
        {
            printf("\n");
        }
    }
}

```

```
    column_count = 0;
}
}

return 0;
}
```

6.2.5 Example 5

This program starts an array with ten elements, uses a for loop to ask the user to input ten elements and assign each element to the proper position in the array. The program, then, ask the user to input a number, if the number is in the array, its position is returned, if the number is not in the array, the program returns another message. This code outputs:

```
C:\Windows\system32\cmd.exe
Please input 10 numbers...
Number at index 0: 03
Number at index 1: 5
Number at index 2: 6
Number at index 3: 78
Number at index 4: 95
Number at index 5: 113
Number at index 6: 321
Number at index 7: 56
Number at index 8: 45
Number at index 9: 87

What value are you searching for? 3
3 was found at index 0
Pressione qualquer tecla para continuar. . .
```

Figure 180: Loop Array Fill and Search Not Found.

```
C:\Windows\system32\cmd.exe
Please input 10 numbers...
Number at index 0: 5897
Number at index 1: 13
Number at index 2: 21
Number at index 3: 34
Number at index 4: 78
Number at index 5: 48
Number at index 6: 57
Number at index 7: 56
Number at index 8: 03
Number at index 9: 3

What value are you searching for? 13
13 was found at index 1
Pressione qualquer tecla para continuar. . .
```

Figure 181: Loop Array Fill and Search Number Found.

This was the written code:

```
#include <stdio.h>

int main()
{
    int data[10];

    // Fill the array with user input:
    printf("Please input 10 numbers...\n");

    for (int i = 0; i < 10; ++i)
    {
        printf("Number at index %d: ", i);

        scanf("%d", &data[i]);
    }

    // Search the array for a value:
    printf("\nWhat value are you searching for? ");
    int search_value = 0;
    scanf("%d", &search_value);

    int found_at = -1;
    for (int index = 0; index < 10 && found_at == -1; ++index)
    {
        if (search_value == data[index])
```

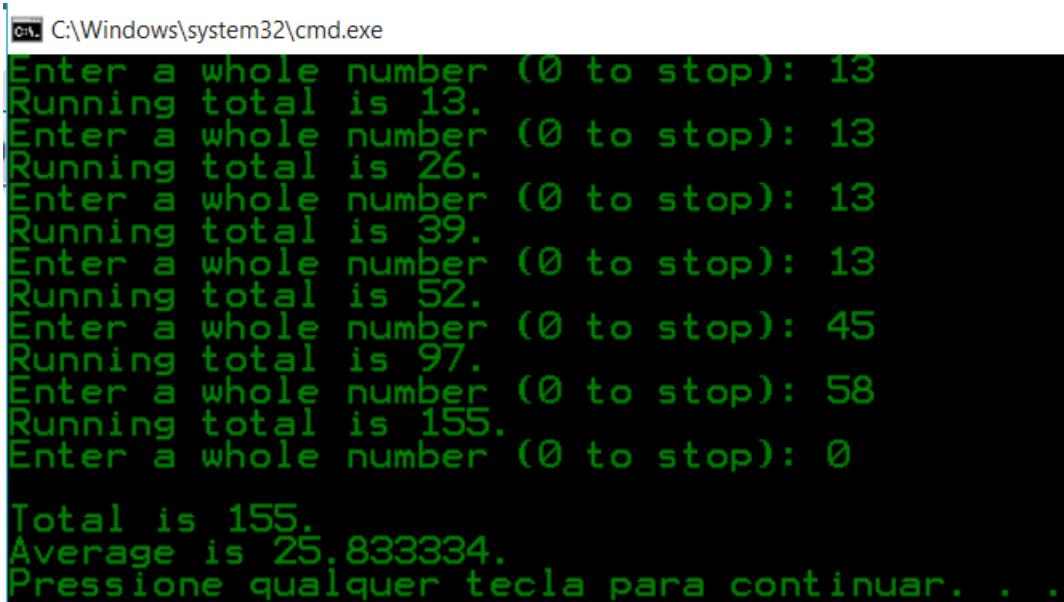
```
        {
            found_at = index;
        }
    }

    if (found_at != -1)
    {
        printf("%d was found at", data[found_at]);
        printf(" index %d\n", found_at);
    }
    else
    {
        printf("%d was not found!\n", search_value);
    }

    return 0;
}
```

6.2.6 Example 6

This code gives an example of how the break keyword works. The program creates an infinite while loop, asks the user to input a number, if the number is 0, it stops the loop, otherwise, it adds it to a variable, in the end, the program sums all the numbers, output the result and gives the result of the average as well. It outputs:



```
C:\Windows\system32\cmd.exe
Enter a whole number (0 to stop): 13
Running total is 13.
Enter a whole number (0 to stop): 13
Running total is 26.
Enter a whole number (0 to stop): 13
Running total is 39.
Enter a whole number (0 to stop): 13
Running total is 52.
Enter a whole number (0 to stop): 45
Running total is 97.
Enter a whole number (0 to stop): 58
Running total is 155.
Enter a whole number (0 to stop): 0

Total is 155.
Average is 25.833334.
Pressione qualquer tecla para continuar. . .
```

Figure 182: Loop with break.

This was the written code:

```
#include <stdio.h>

int main()
{
    int sum = 0;
    int count = 0;

    while (1)
    {
```

```
int input = 0;

printf("Enter a whole number (0 to stop): ");
scanf("%d", &input);

if (0 == input)
{
    break;
}

sum += input;
++count;

printf("Running total is %d.\n", sum);
}

printf("\nTotal is %d.\n", sum);

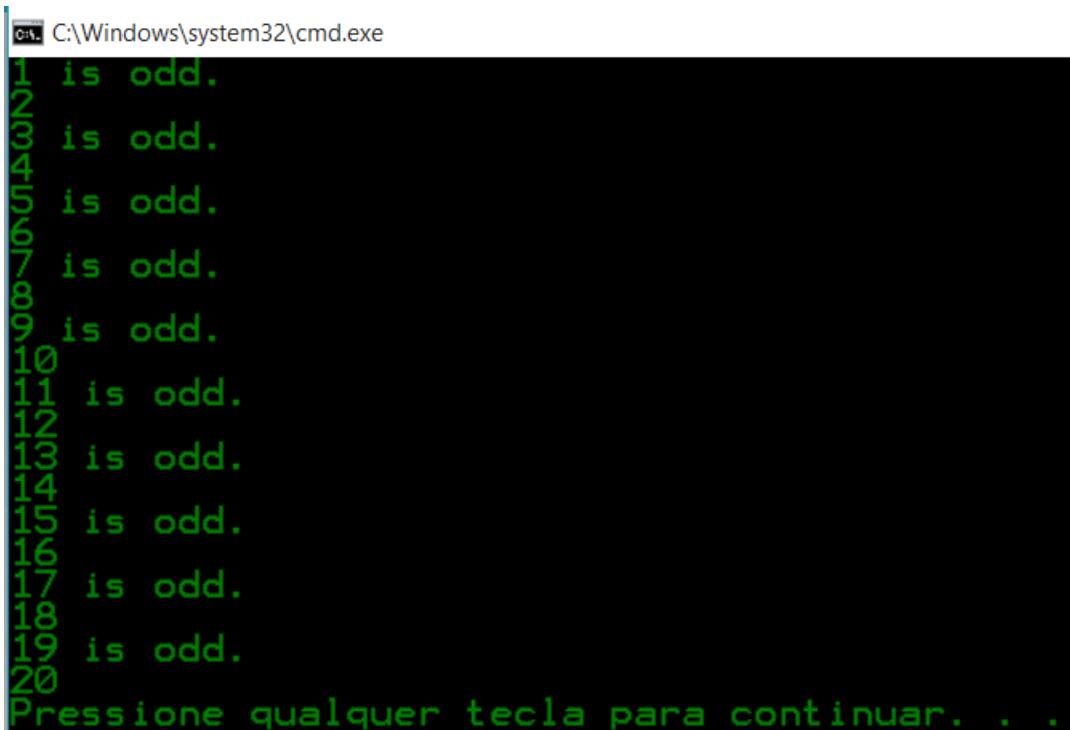
float real_sum = sum;
float average = real_sum / count;

printf("Average is %f.\n", average);

return 0;
}
```

6.2.7 Example 7

The program initializes and declares a variable as zero, and prints the value of the variable until it achieves twenty, checks if the number is odd or not, if true, it says that it is an odd number, as shown in the output.



C:\Windows\system32\cmd.exe

```
1 is odd.
2
3 is odd.
4
5 is odd.
6
7 is odd.
8
9 is odd.
10
11 is odd.
12
13 is odd.
14
15 is odd.
16
17 is odd.
18
19 is odd.
20
Pressione qualquer tecla para continuar. . .
```

Figure 183: Loop with continue.

This was the written code:

```
#include <stdio.h>

int main()
{
    int number = 0;

    while (number < 20)
    {
```

```
++number;

printf("%d", number);

if (number % 2 == 0)
{
    printf("\n");

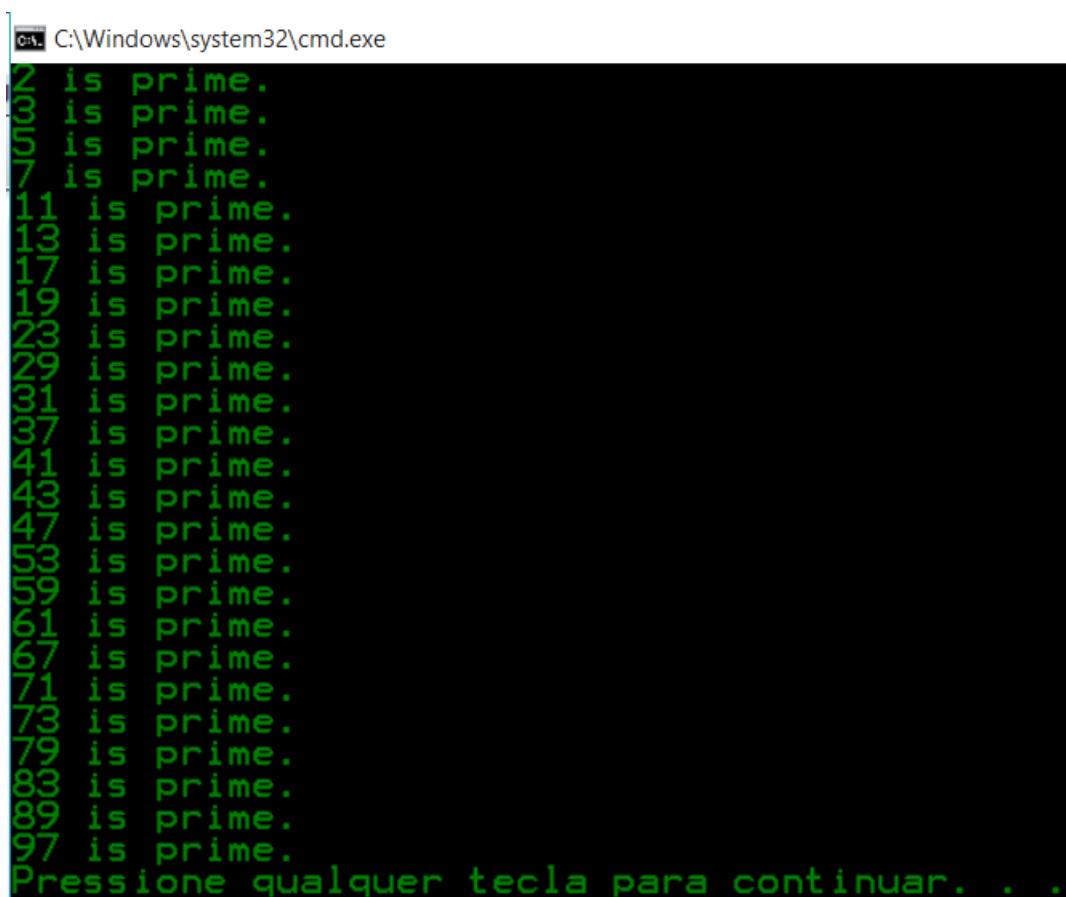
    continue;
}

printf(" is odd.\n");
}

return 0;
}
```

6.2.8 Example 8

This program uses nested structures and prints all the prime numbers between 2 and 100. The program outputs:



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the output of a program that generates prime numbers from 2 to 97. Each prime number is followed by the text 'is prime.'. The output is as follows:

```
2 is prime.
3 is prime.
5 is prime.
7 is prime.
11 is prime.
13 is prime.
17 is prime.
19 is prime.
23 is prime.
29 is prime.
31 is prime.
37 is prime.
41 is prime.
43 is prime.
47 is prime.
53 is prime.
59 is prime.
61 is prime.
67 is prime.
71 is prime.
73 is prime.
79 is prime.
83 is prime.
89 is prime.
97 is prime.
Pressione qualquer tecla para continuar. . .
```

Figure 184: Nested for Loops Prime Number Generator.

This was the written code:

```
#include <stdio.h>

int main()
{
    for (int i = 2; i < 100; ++i)
```

```
{  
    int k = 0;  
  
    for (k = 2; k <= (i / k); ++k)  
    {  
        if ((i % k) == 0)  
        {  
            break;  
        }  
    }  
  
    if (k >(i / k))  
    {  
        printf("%d is prime.\n", i);  
    }  
}  
  
return 0;  
}
```

6.3 Exercises

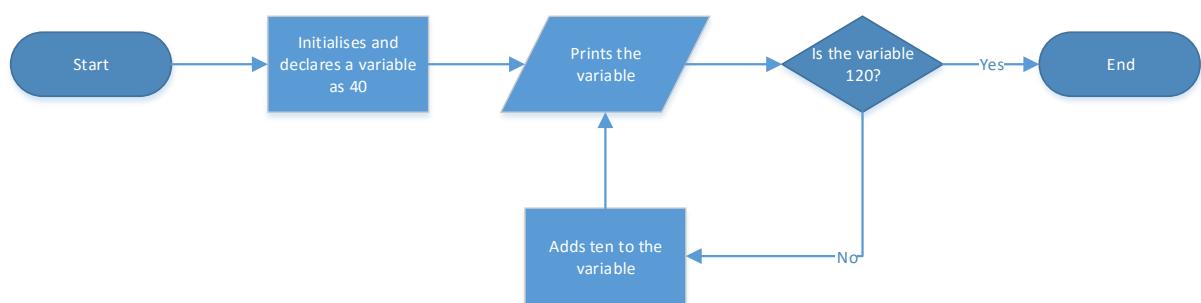
6.3.1 Exercise 1

This program should use a loop to count from 40 to 120 in steps of ten, as in the example:

```
40
50
60
70
80
90
100
110
120
```

Figure 185: Simple Count Up Loop.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

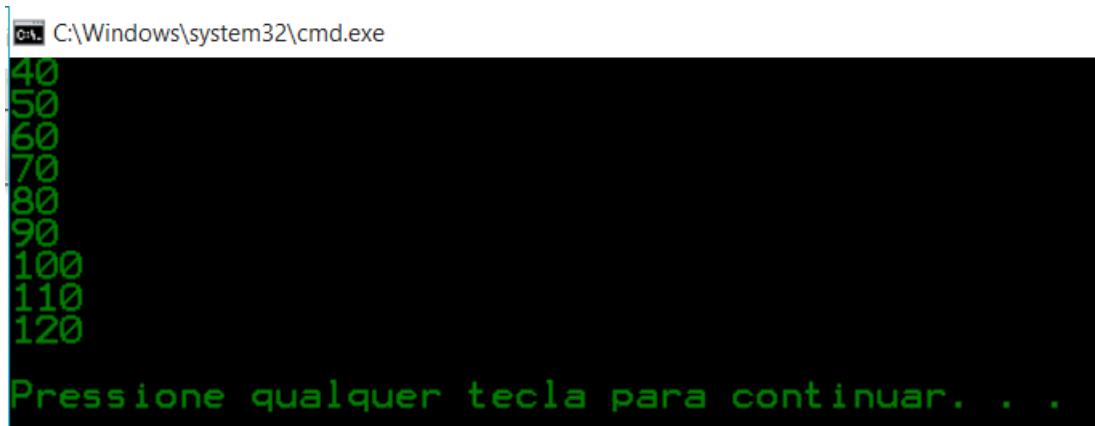
```
#include <stdio.h>

int main()
```

Paludo

```
{  
    for (int i = 40; i <= 120; i += 10)  
    {  
        printf("%d\n", i);  
    }  
  
    printf("\n");  
}  
return 0;  
}
```

This code outputs:



```
C:\Windows\system32\cmd.exe  
40  
50  
60  
70  
80  
90  
100  
110  
120  
Pressione qualquer tecla para continuar. . .
```

Figure 186: Simple Count Up Loop Output.

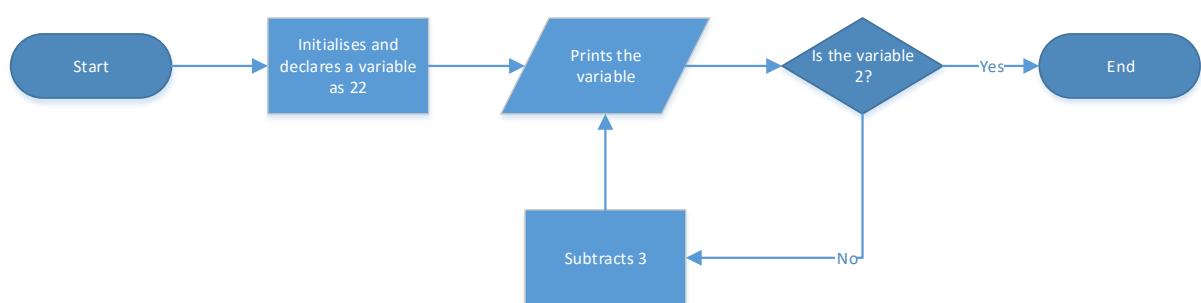
6.3.2 Exercise 2

This program is very similar to the previous program. It initializes a variable as 22 and counts down in steps of three, as in the example:

```
22
20
18
16
14
12
10
8
6
4
2
```

Figure 187: Simple Count Down Loop.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    for (int i = 22; i >= 2; i -= 2)
```

```
{  
    printf("%d\n", i);  
}  
  
printf("\n");  
  
return 0;  
}
```

This code outputs:



A screenshot of a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The window displays a series of integers from 22 down to 2, each on a new line. After the numbers, there is a message in Portuguese: 'Pressione qualquer tecla para continuar. . .' (Press any key to continue. .).

```
22  
20  
18  
16  
14  
12  
10  
8  
6  
4  
2  
Pressione qualquer tecla para continuar. . .
```

Figure 188: Simple Count Down Loop Output.

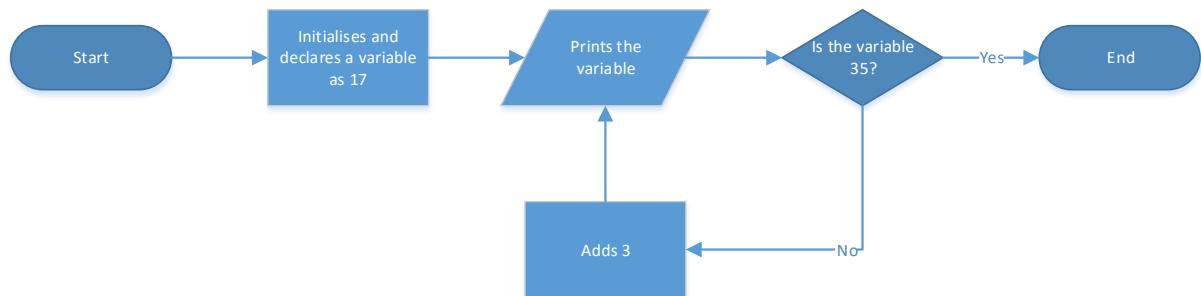
6.3.3 Exercise 3

Similar to the previous exercises, this program should declare and initialize a variable as 17 and count up to 35 in steps of three, as in the example:

```
17
20
23
26
29
32
35
```

Figure 189: Another Simple Count Up Loop.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

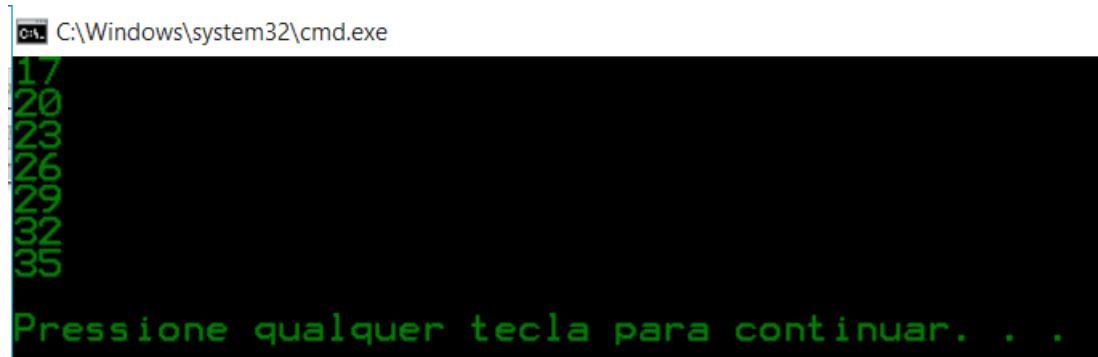
```
#include <stdio.h>

int main()
{
    for (int i = 17; i >= 35; i += 3)
    {
        printf("%d\n", i);
    }

    printf("\n");
}
```

```
    return 0;  
}
```

This code outputs:



A screenshot of a Windows Command Prompt window titled "C:\Windows\system32\cmd.exe". The window displays the following text:
17
20
23
26
29
32
35

Pressione qualquer tecla para continuar. . .

Figure 190: Another Simple Count Up Output.

6.3.4 Exercise 4

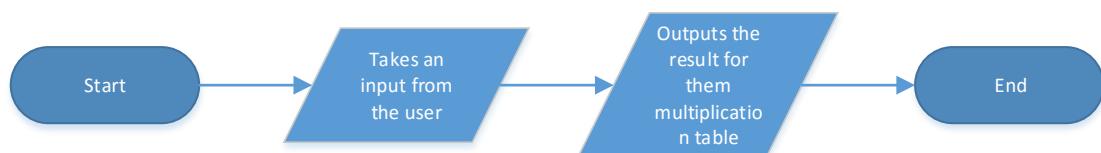
This program uses a loop structure to output the multiplication table from 0 to 14 for a number that the user input, as in the example:

```
Enter a whole number: 7

The 7 Times Table:
-----
0 x 7 = 0
1 x 7 = 7
2 x 7 = 14
3 x 7 = 21
4 x 7 = 28
5 x 7 = 35
6 x 7 = 42
7 x 7 = 49
8 x 7 = 56
9 x 7 = 63
10 x 7 = 70
11 x 7 = 77
12 x 7 = 84
13 x 7 = 91
14 x 7 = 98
```

Figure 191: Multiplication Table.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    int number;
    printf("Enter a whole number: ");
    scanf("%d", &number);

    printf("\n\n");

    printf("The %d Times Table:\n", number);
    printf("-----");

    printf("\n\n");

    int result = 0;

    for(int i = 0; i <= 14; i++)
    {
        result = number * i;
        printf("%d x %d = %d\n", i, number, result);
    }

    printf("\n");

    return 0;
}
```

This code outputs:

The screenshot shows a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The user has entered the number '3' and the program has output the 'The 3 Times Table' from 0 to 14. The output is as follows:

```
Enter a whole number: 3
The 3 Times Table:
-----
0 x 3 = 0
1 x 3 = 3
2 x 3 = 6
3 x 3 = 9
4 x 3 = 12
5 x 3 = 15
6 x 3 = 18
7 x 3 = 21
8 x 3 = 24
9 x 3 = 27
10 x 3 = 30
11 x 3 = 33
12 x 3 = 36
13 x 3 = 39
14 x 3 = 42
Pressione qualquer tecla para continuar. . .
```

Figure 192: Multiplication Table Output.

6.3.5 Exercise 5

This program asks the user to input a starting, stopping and step size number, using a for loop, the program outputs all the steps, as shown:

```
Starting number: 10
Stopping number: 35
Step size: 3
```

```
Using a for loop:
```

```
Starting at 10...
```

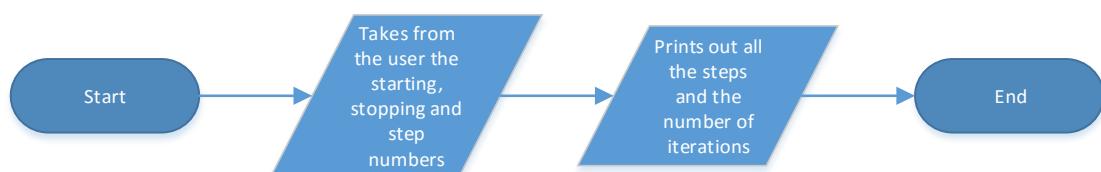
```
10...
13...
16...
19...
22...
25...
28...
31...
34...
```

```
Stopping at 35...
```

```
This loop did 9 iterations.
```

Figure 193: Stepping with a for Loop.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    int starting;
    printf("Starting number:");
    scanf("%d", &starting);

    int stopping;
    printf("Stopping number:");
    scanf("%d", &stopping);

    int size;
    printf("Step size:");
    scanf("%d", &size);

    printf("\n");

    printf("Using a for loop:");
    printf("\n\n");

    printf("Starting at %d...", starting);
    printf("\n\n");

    int count = 0;

    for (starting; starting <= stopping; starting += size)
    {
        count++;

        printf("%d...\n", starting);
    }

    printf("\n");

    printf("Stopping at %d", stopping);
    printf("\n\n");

    printf("This loop did %d iterations.", count);

    printf("\n\n");

    return 0;
}
```

}

This code outputs:

```
C:\Windows\system32\cmd.exe
Starting number:5
Stopping number:13
Step size:3

Using a for loop:

Starting at 5...
5...
8...
11...

Stopping at 13
This loop did 3 iterations.

Pressione qualquer tecla para continuar. . .
```

Figure 194: Stepping with a for Loop Output.

6.3.6 Exercise 6

This exercises gives a pseudo code and asks the student to develop the program, this is the pseudo code:

```

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.
  
```

Figure 195: Pseudo Code Loop.

Using this pseudo code, I wrote the following code:

```

#include <stdio.h>

int main()
{
    /*
    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.
    */
    int result = 0;

    for (int n = 0; n < 10; n++)
    {
        if (n > 4)
        {
            printf("%d\n", n);
        }
    }
}
  
```

```
    }
    else
    {
        result = 9 - n;
        printf("%d\n", result);
    }
}

printf("\n\n");

return 0;
}
```

This program outputs:



The screenshot shows a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
9
8
7
6
5
5
6
7
8
9

Pressione qualquer tecla para continuar. . .

Figure 196: Pseudo Code Loop Output.

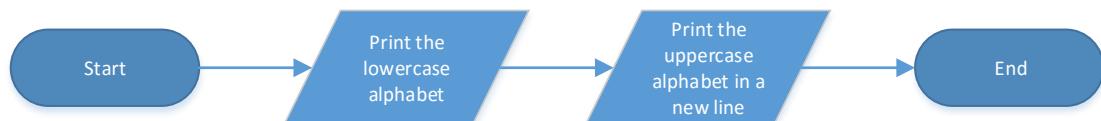
6.3.7 Exercise 7

This program should use loops to output all the lowercase and upper case letters of the alphabet separated by a space, as shown in the example:

```
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
```

Figure 197: ASCII Alphabet Printer.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    for (int i = 97; i <= 122; i++)
    {
        printf("%c ", i);
    }

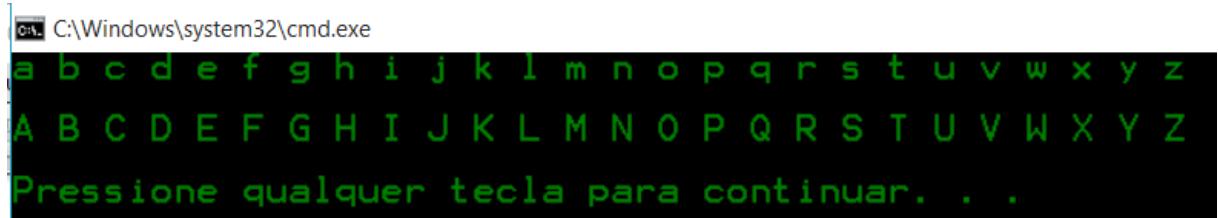
    printf("\n\n");

    for (int n = 65; n <= 90; n++)
    {
        printf("%c ", n);
    }

    printf("\n\n");
}
```

```
    return 0;  
}
```

This code outputs:



A screenshot of a Windows command prompt window titled 'cmd C:\Windows\system32\cmd.exe'. The window displays the following text:
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
Pressione qualquer tecla para continuar. . .

Figure 198: ASCII Alphabet Printer Output.

6.3.8 Exercise 8

This program takes input from the user, and stops when the user enter 0. Then the program outputs the lowest and highest numbers entered.

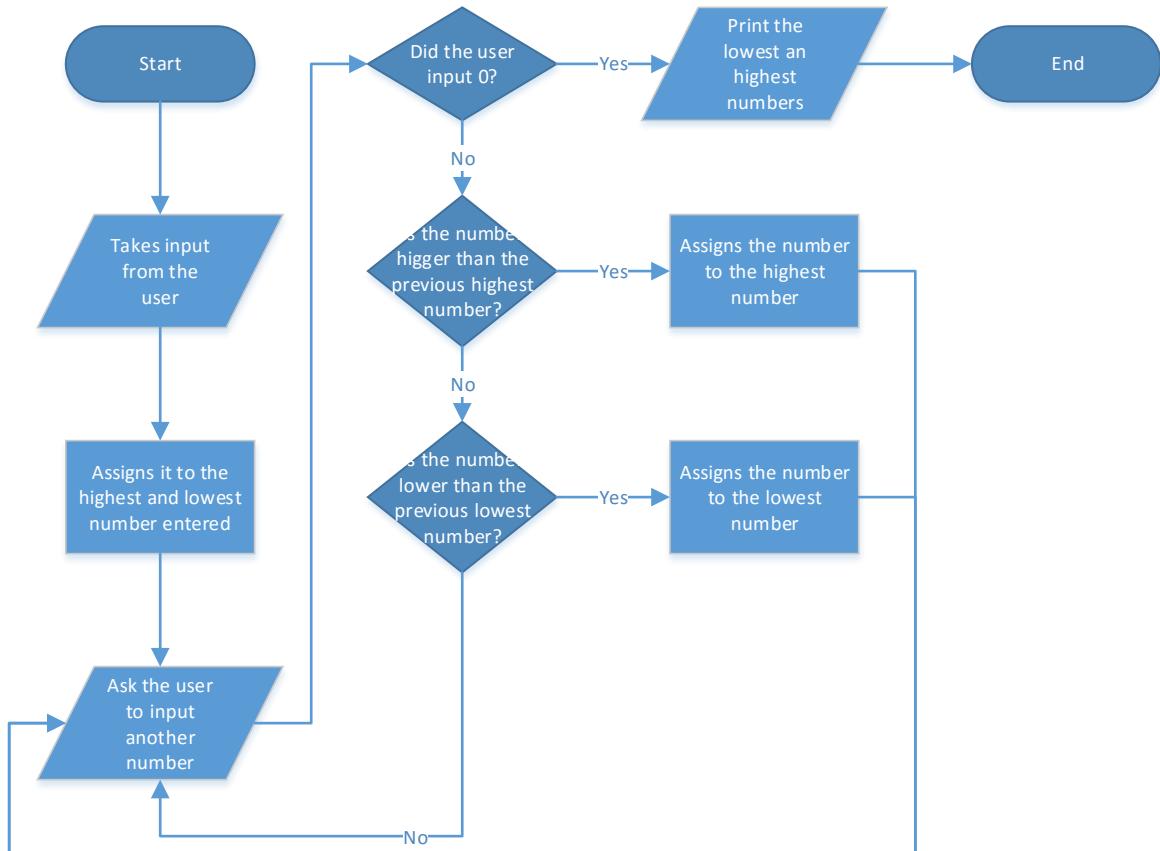
```
Enter a number (0 to stop): 55
Enter a number (0 to stop): 62
Enter a number (0 to stop): 17
Enter a number (0 to stop): 41
Enter a number (0 to stop): 89
Enter a number (0 to stop): 42
Enter a number (0 to stop): 0

The lowest number entered was: 17

The highest number entered was: 89
```

Figure 199: Lowest and Highest Input.

To develop this program I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

int main()
{
    int number;
    printf("Enter a number (0 to stop):");
    scanf("%d", &number);

    int lowest = number;
    int highest = number;

    while (number != 0)
    {
        printf("Enter a number (0 to stop):");
        scanf("%d", &number);

        if (number > highest)
            highest = number;
        else if (number < lowest)
            lowest = number;
    }
}
  
```

```
if (number > highest)
{
    highest = number;
}
else if (number < lowest && number != 0)
{
lowest = number;
}
}

printf("\n");

printf("The lowest number entered was: %d", lowest);
printf("\n\n");

printf("The highest number entered was: %d", highest);
printf("\n\n");

return 0;
}
```

This code outputs:

The screenshot shows a Windows Command Prompt window titled 'cmd C:\Windows\system32\cmd.exe'. The window displays the following text:

```
Enter a number (0 to stop):35
Enter a number (0 to stop):13
Enter a number (0 to stop):18
Enter a number (0 to stop):17
Enter a number (0 to stop):5
Enter a number (0 to stop):10
Enter a number (0 to stop):15
Enter a number (0 to stop):21
Enter a number (0 to stop):0

The lowest number entered was: 5
The highest number entered was: 35
Pressione qualquer tecla para continuar. . .
```

Figure 200: Lowest and Highest Input Output.

6.3.9 Exercise 9

This program takes seven inputs from the user, these inputs are the time that the user took to complete the run, the program then output the best time.

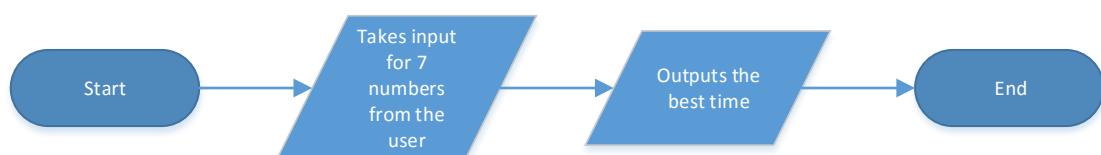
```
Enter your daily run time in minutes.
```

```
Day 1: 44
Day 2: 48
Day 3: 51
Day 4: 38
Day 5: 39
Day 6: 42
Day 7: 46
```

```
Your best time this week was 38 minutes.
```

Figure 201: Daily Run Personal Best.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    printf("Enter your daily run time in minutes.");
    printf("\n\n");

    int time;
    printf("Day 1: ");

```

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

int best = time;

for (int i = 2; i <= 7; i++)
{
    printf("Day %d: ", i);
    scanf("%d", &time);

    if (time < best)
    {
        best = time;
    }
}

printf("\n");

printf("Your best time this week was %d minutes.", best);

printf("\n\n");

return 0;
}
```

This code outputs:

```
C:\Windows\system32\cmd.exe
Enter your daily run time in minutes.

Day 1: 13
Day 2: 8
Day 3: 15
Day 4: 17
Day 5: 9
Day 6: 12
Day 7: 10

Your best time this week was 8 minutes.

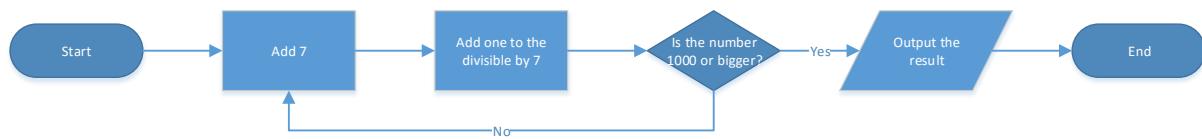
Pressione qualquer tecla para continuar. . .
```

Figure 202: Daily Run Personal Best Output.

6.3.10 Exercise 10

This program counts how many numbers between 1 and 1000 are divisible by seven, with no remainder and outputs the result.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

int main()
{
    int seven = 0;

    for (int i = 7; i <= 1000; i += 7)
    {
        seven++;
    }

    printf("There are %d numbers divisible by 7 between 1 and
1000.", seven);

    printf("\n\n");

    return 0;
}
  
```

This code outputs:

```

C:\Windows\system32\cmd.exe
There are 142 numbers divisible by 7 between 1 and 1000.
Pressione qualquer tecla para continuar. . .
  
```

Figure 203: How Many Divisible by Seven.

6.3.11 Exercise 11

This program takes characters from the user to input a character and gives the kind of character that the user input, to stop the program, the user should press Ctrl+C.

```
Press Ctrl-C to quit!
-----
Enter a character: a
[a] lowercase

Enter a character: Z
[Z] uppercase

Enter a character: 7
[7] digit

Enter a character: !
[!] something else

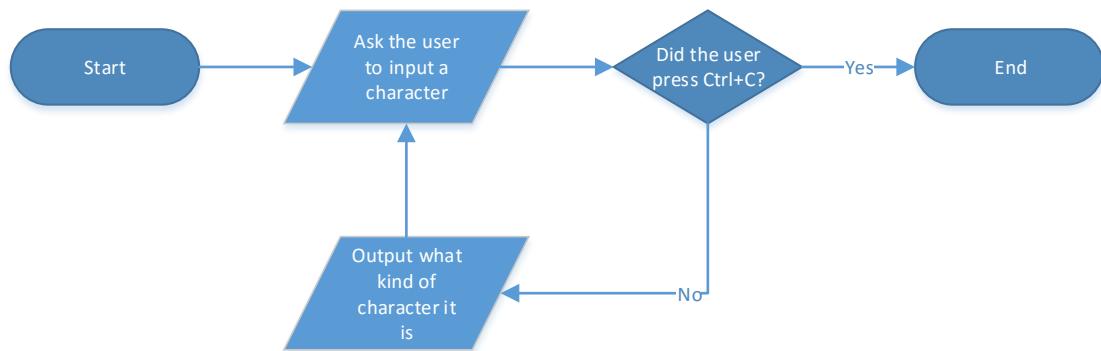
Enter a character: 2
[2] digit

Enter a character: S
[S] uppercase

Enter a character:
```

Figure 204: Infinite Character Input Loop.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

int main()
{
    printf("Press Ctrl-C to quit!\n");
    printf("-----");
    printf("\n\n");

    char character;

    while (1)
    {
        printf("Enter a character: ");
        scanf(" %c", &character);

        if (character >= 48 && character <= 57)
        {
            printf("[%c] digit");
            printf("\n\n");
        }
        else if (character >= 65 && character <= 90)
        {
            printf("[%c] lowercase");
            printf("\n\n");
        }
        else if (character >= 97 && character <= 122)
        {
            printf("[%c] uppercase");
            printf("\n\n");
        }
        else
        {
    
```

```
        printf("[%c] something else");
        printf("\n\n");
    }
}

printf("\n\n");

return 0;
}
```

But this code is not compiling, it gives me an error, but I could not find out what exactly is causing the error with Visual Studio. I also tried CodeBlocks, but it doesn't give me what is causing the error as well. I figured out that I forgot to put the & symbol in the scanf variable, this is the written code:

```
#include <stdio.h>

int main()
{
    printf("Press Ctrl-C to quit!\n");
    printf("-----");
    printf("\n\n");

    char character;

    while (1)
    {
        printf("Enter a character: ");
        scanf(" %c", &character);

        if (character >= 48 && character <= 57)
        {
            printf("[%c] digit");
            printf("\n\n");
        }
        else if (character >= 65 && character <= 90)
        {
            printf("[%c] uppercase");
            printf("\n\n");
        }
        else if (character >= 97 && character <= 122)
        {
            printf("[%c] lowercase");
            printf("\n\n");
        }
    }
}
```

```
    }
    else
    {
        printf("[%c] something else");
        printf("\n\n");
    }
}

printf("\n\n");

return 0;
}
```

This code compiles, but it does not do what it is supposed to do.



A screenshot of a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:

```
Press Ctrl-C to quit!
-----
Enter a character: d
[F] something else
Enter a character: a
[F] something else
Enter a character: b
[F] something else
Enter a character: ^CPressione qualquer tecla para continuar. . .
```

Figure 205: Infinite Character Input Loop Wrong Output.

The program is always returning [F].

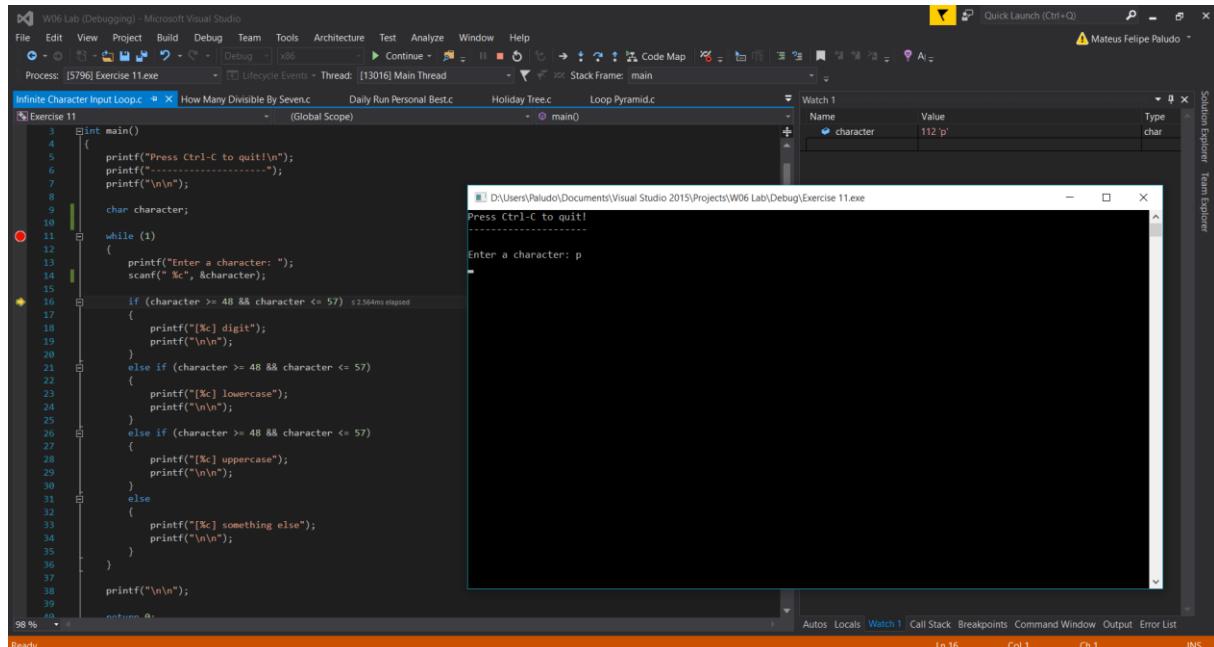


Figure 206: Debugging the problem.

The program is assigning correctly the char to the respective variable. I realized that I forgot to specify which variable the program should read. This was the written code:

```
#include <stdio.h>

int main()
{
    printf("Press Ctrl-C to quit!\n");
    printf("-----");
    printf("\n\n");

    char character;

    while (1)
    {
        printf("Enter a character: ");
        scanf(" %c", &character);

        if (character >= 48 && character <= 57)
        {
            printf("[%c] digit", character);
            printf("\n\n");
        }
    }
}
```

```

else if (character >= 48 && character <= 57)
{
    printf("[%c] lowercase", character);
    printf("\n\n");
}
else if (character >= 65 && character <= 90)
{
    printf("[%c] uppercase", character);
    printf("\n\n");
}
else
{
    printf("[%c] something else", character);
    printf("\n\n");
}

printf("\n\n");

return 0;
}

```

But the program displays the same thing. I cleaned the solution, the program was returning the correct variable, but it still do not giving the correct output, this happened because I forgot to change the character numbers.



The screenshot shows a Windows command prompt window titled 'cmd' with the path 'C:\Windows\system32\cmd.exe'. It displays the following text:

```

Press Ctrl-C to quit!
-----
Enter a character: a
[a] something else

Enter a character: B
[B] something else

Enter a character: 2
[2] digit

Enter a character: 3
[3] digit

Enter a character: ^CPressione qualquer tecla para continuar. . .

```

Figure 207: Infinite Character Input Loop, another wrong output.

After fixing the problem, this was my code:

```
#include <stdio.h>

int main()
{
    printf("Press Ctrl-C to quit!\n");
    printf("-----");
    printf("\n\n");

    char character;

    while (1)
    {
        printf("Enter a character: ");
        scanf(" %c", &character);

        if (character >= 48 && character <= 57)
        {
            printf("[%c] digit", character);
            printf("\n\n");
        }
        else if (character >= 97 && character <= 122)
        {
            printf("[%c] lowercase", character);
            printf("\n\n");
        }
        else if (character >= 65 && character <= 90)
        {
            printf("[%c] uppercase", character);
            printf("\n\n");
        }
        else
        {
            printf("[%c] something else", character);
            printf("\n\n");
        }
    }

    printf("\n\n");

    return 0;
}
```

This program outputs:

```
C:\Windows\system32\cmd.exe
Press Ctrl-C to quit!
-----
Enter a character: p
[p] lowercase

Enter a character: P
[P] uppercase

Enter a character: 0
[0] digit

Enter a character: ^
[~] something else

Enter a character: ^CPressione qualquer tecla para continuar. . .
```

Figure 208: Infinite Character Input Loop Output.

After solving this problem, I went to rest, because I could not think correctly and it was taking me a long time to solve simple problems.

6.3.12 Exercise 12

This program takes an input from the user for the base and powers this number to another number that the user input. It was advised to don't use the function pow (). The program should also output how the operation was done, as shown in the example below:

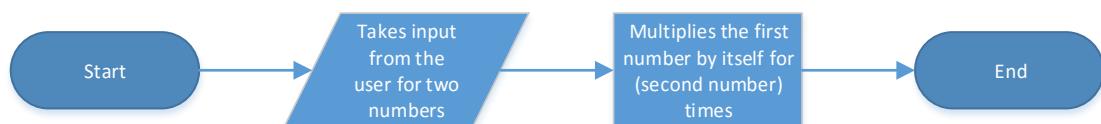
```
Enter the base: 2
Enter the power: 5

2 ^ 5 is the same as...

2 * 2 * 2 * 2 * 2 which is 32
```

Figure 209: Power Using a Loop.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    int base;
    printf("Enter the base: ");
    scanf("%d", &base);

    int pow;
    printf("Enter the power: ");
    scanf("%d", &pow);

    printf("\n");
```

```

printf("%d ^ %d is the same as...", base, pow);

printf("%d ", base);

int result;

for (int i = 0; i < pow; base = base * base)
{
    printf("* %d ", base);
}

printf(" which is %d", result);

printf("\n\n");

return 0;
}

```

This code caused an infinite loop:

The screenshot shows a Windows Command Prompt window with the title 'C:\Windows\System32\cmd.exe'. The command 'Enter the base: 3' has been typed, followed by 'Enter the power: 9'. The output is a very long string of digits, starting with '81' and followed by approximately 1000 '1's. The string is so long that it wraps around the screen multiple times.

Figure 210: Power Using a Loop Infinite Loop.

I realized that I was multiplying the base by itself and assigning the result to the base, and repeating this operation, what was causing the number to be extremely big. I thought about initializing the result variable as zero and changing the operation to: `result += base * base`,

but this is not the correct approach to take as well. This was the code I wrote after correcting some mistakes:

```
#include <stdio.h>

int main()
{
    int base;
    printf("Enter the base: ");
    scanf("%d", &base);

    int pow;
    printf("Enter the power: ");
    scanf("%d", &pow);

    printf("\n");

    printf("%d ^ %d is the same as...", base, pow);
    printf("\n\n");

    printf("%d ", base);

    int result = base;

    for (int i = 0; i < pow; result = result * base)
    {
        printf("* %d ", base);
    }

    printf(" which is %d", result);

    printf("\n\n");

    return 0;
}
```

But this also caused an infinite loop.

The screenshot shows a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The user has entered 'Enter the base: 2' and 'Enter the power: 4'. The program then prints '2 ^ 4 is the same as...'. Instead of calculating the power, it enters an infinite loop where it repeatedly prints the character '2' on a new line, filling the screen with a pattern of '2's.

Figure 211: Power Using a Loop, infinite loop again.

I realized that I forgot to increase the variable i in each operation. After fixing this problem, I run the program again with the following code:

```
#include <stdio.h>

int main()
{
    int base;
    printf("Enter the base: ");
    scanf("%d", &base);

    int pow;
    printf("Enter the power: ");
    scanf("%d", &pow);

    printf("\n");

    printf("%d ^ %d is the same as...", base, pow);
    printf("\n\n");

    printf("%d ", base);

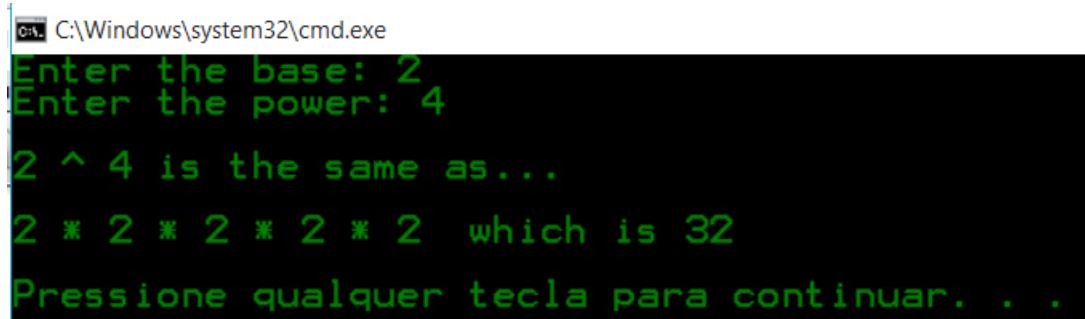
    int result = base;

    for (int i = 0; i < pow; i++)
    {
        printf("* %d ", base);
        result = result * base;
    }

    printf(" which is %d", result);

    printf("\n\n");

    return 0;
}
```



```
C:\Windows\system32\cmd.exe
Enter the base: 2
Enter the power: 4
2 ^ 4 is the same as...
2 * 2 * 2 * 2 * 2  which is 32
Pressione qualquer tecla para continuar. . .
```

Figure 212: Power Using a Loop Wrong Result.

The program was doing the operation one more time than it was supposed to do. To fix this, I initialized the variable i as one.

```
#include <stdio.h>

int main()
{
    int base;
    printf("Enter the base: ");
    scanf("%d", &base);

    int pow;
    printf("Enter the power: ");
    scanf("%d", &pow);

    printf("\n");

    printf("%d ^ %d is the same as...", base, pow);
    printf("\n\n");

    printf("%d ", base);

    int result = base;

    for (int i = 1; i < pow; i++)
    {
        printf("* %d ", base);
        result = result * base;
    }

    printf(" which is %d", result);
```

```
    printf("\n\n");
    return 0;
}
```

This code outputs:

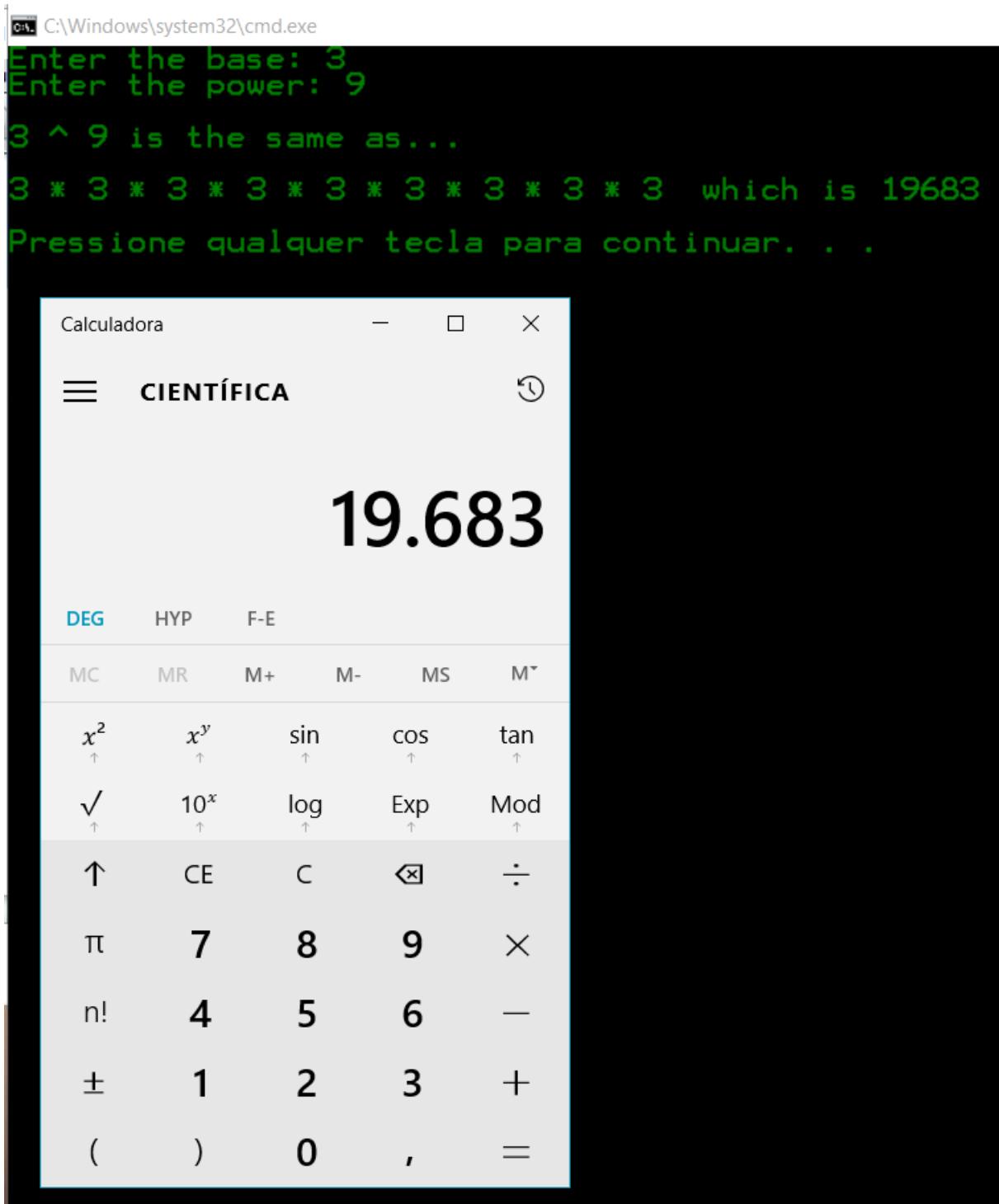


Figure 213: Power Using a Loop Output.

This program does not work for numbers powered to 0.

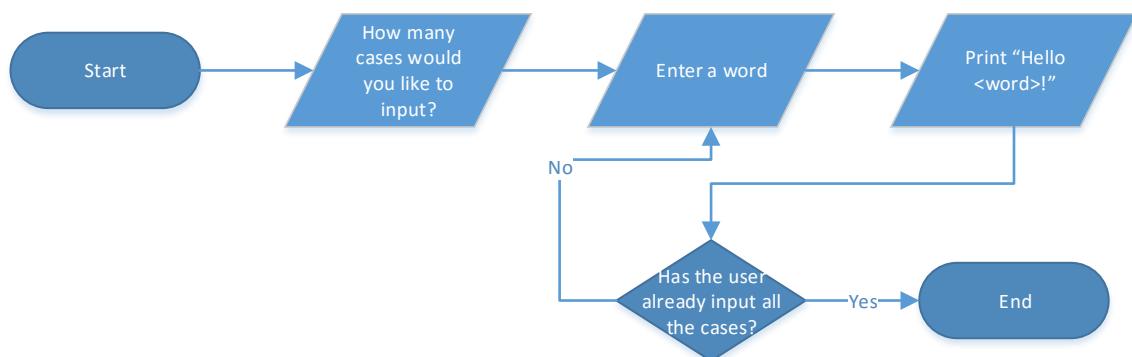
6.3.13 Exercise 13

This program asks the user to input how many test cases will be input into the program (this number must be between 1 and 100 inclusive). Each test case should ask the user to input a single word with no more than 99 characters and then output “Hello <word>!”, as in the example:

```
Number of test cases? 3
Test Case 1: world
Output: Hello world!
Test Case 2: Student
Output: Hello Student!
Test Case 3: TeachingAssistants
Output: Hello TeachingAssistants!
```

Figure 214: Hello Input Loop.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
```

```
int cases;
printf("Number of test cases? ");
scanf("%d", &cases);

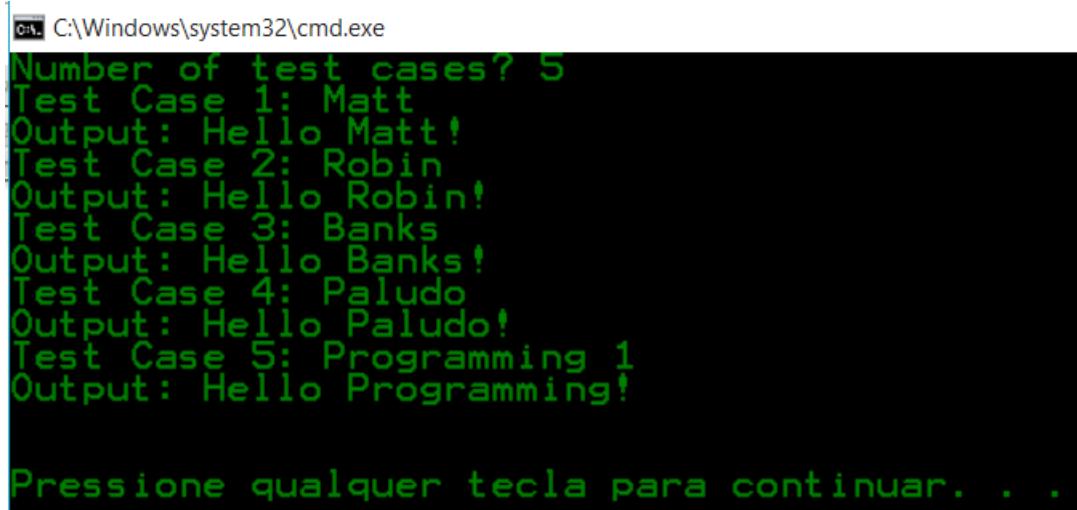
char word[100];

for (int i = 0; i < cases; i++)
{
    printf("Test Case %d: ", i + 1);
    scanf("%99s", word);
    printf("Output: Hello %s!\n", word);
}

printf("\n\n");

return 0;
}
```

This program outputs:



```
C:\Windows\system32\cmd.exe
Number of test cases? 5
Test Case 1: Matt
Output: Hello Matt!
Test Case 2: Robin
Output: Hello Robin!
Test Case 3: Banks
Output: Hello Banks!
Test Case 4: Paludo
Output: Hello Paludo!
Test Case 5: Programming 1
Output: Hello Programming!

Pressione qualquer tecla para continuar. . .
```

Figure 215: Hello Input Loop Output.

6.3.14 Exercise 14

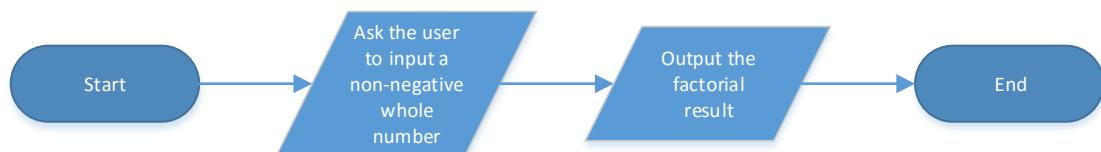
This program takes from the user a non-negative whole number as input and gives the factorial of this number as output.

```
Enter a non-negative whole number: 5
```

```
5! is 120
```

Figure 216: Factorial Calculator with Loop.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    int number;
    printf("Enter a non-negative whole number: ");
    scanf("%d", &number);
    printf("\n");

    printf("%d! is ", number);

    int result = 1;

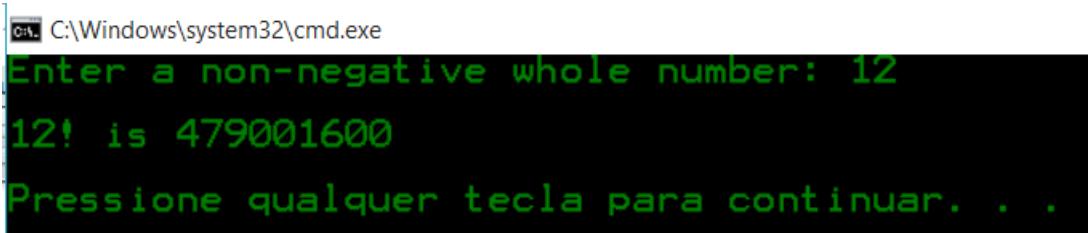
    for (int i = number; i > 0; i--)
    {
        result = result * number;
        number--;
    }
}
```

```
    printf("%d", result);

    printf("\n\n");

    return 0;
}
```

This program outputs:



A screenshot of a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
Enter a non-negative whole number: 12
12! is 479001600
Pressione qualquer tecla para continuar. . .

Figure 217: Factorial Calculator with Loop Output.

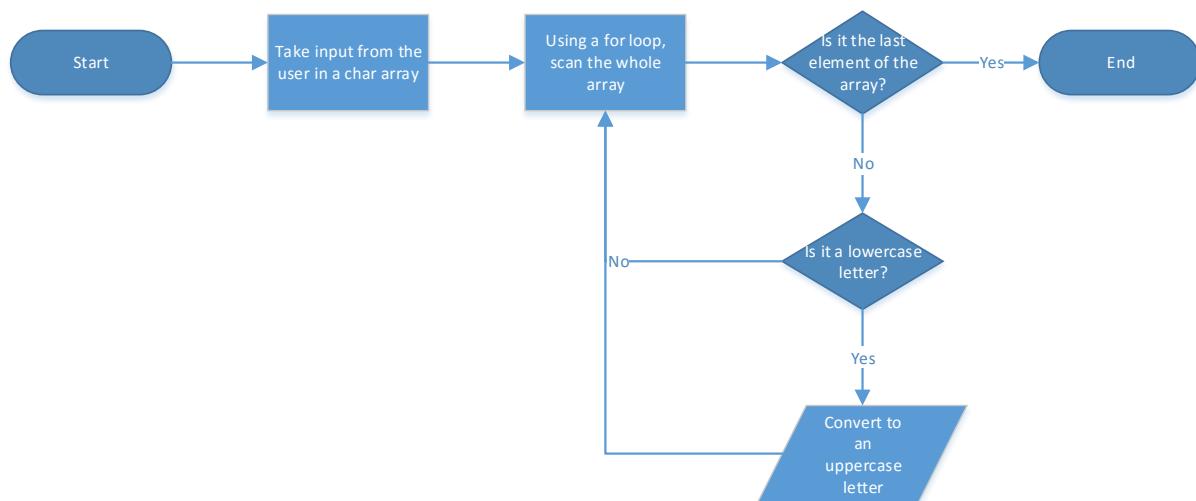
6.3.15 Exercise 15

This programs allows the user to input a text with no space, then output all the text in uppercase letters.

```
> HelloCOMP500andENSE501Students!
HELLOCOMP500ANDENSE501STUDENTS!
```

Figure 218: for Loop with char Array.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    char character[100];
    printf("> ");
    scanf("%99s", character);

    for (int i = 0; i < 100; i++)
    {
```

```
if (character[i] >= 97 && character[i] <= 122)
{
    character[i] = character[i] - 32;
}
printf("%s", character);
printf("\n\n");
return 0;
}
```

This program outputs:



```
C:\Windows\system32\cmd.exe
> MyNameIsBarryAllen, AndIAmTheFastestManAlive
MYNAMEISBARRYALLEN, ANDIAMTHEFASTESTMANALIVE
Pressione qualquer tecla para continuar. . .
```

Figure 219: for Loop with char Array.

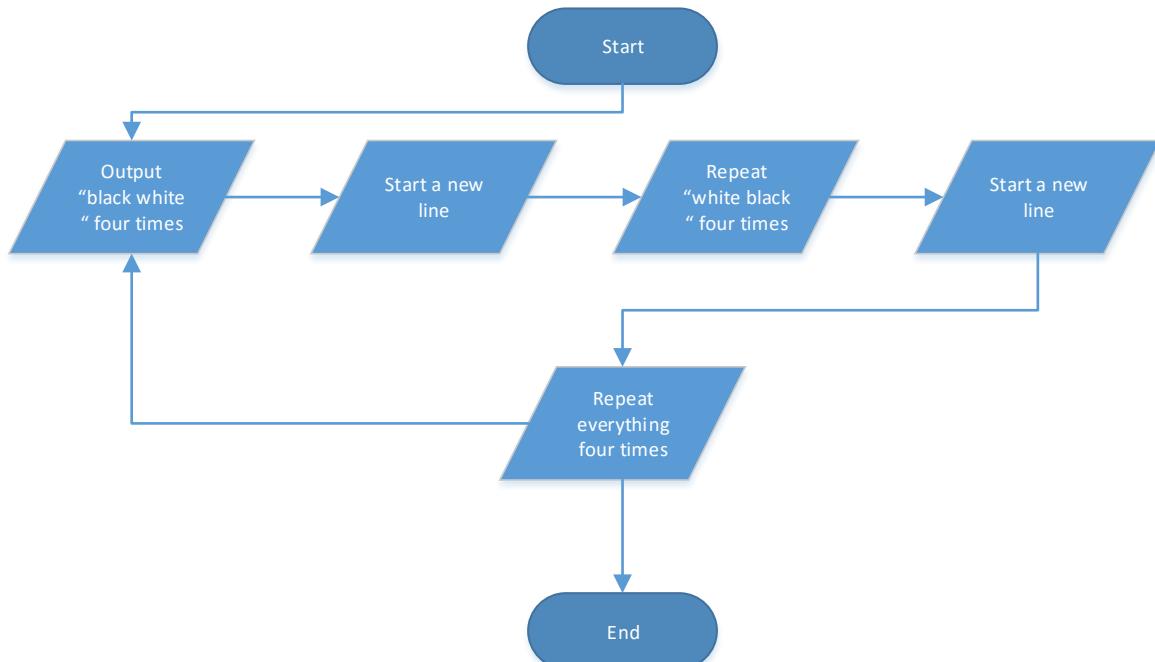
6.3.16 Exercise 16

This program should output this pattern:

```
black white black white black white black white  
white black white black white black white black  
black white black white black white black white  
white black white black white black white black  
black white black white black white black white  
white black white black white black white black  
black white black white black white black white  
white black white black white black white black
```

Figure 220: Checkerboard Pattern.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    char character[100];
    printf("> ");
    scanf("%99s", character);

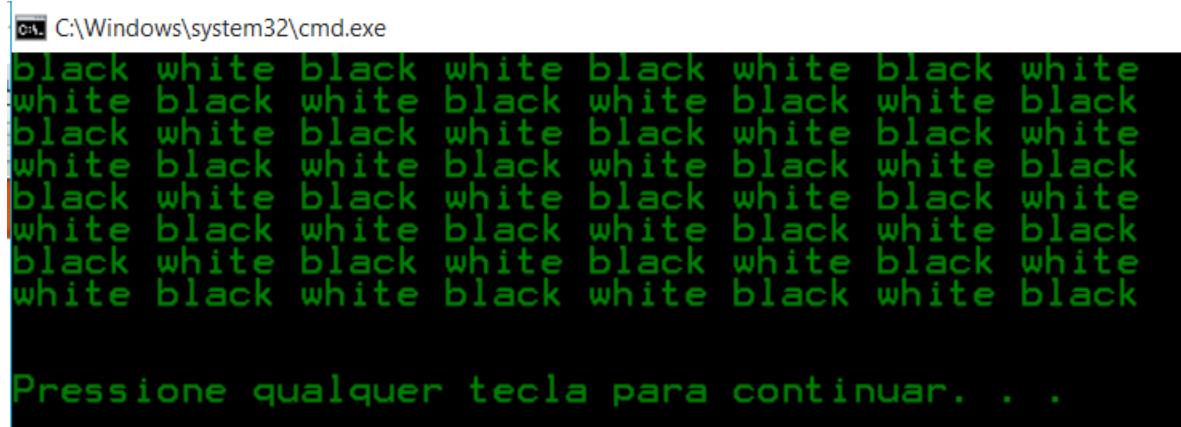
    for (int i = 0; i < 100; i++)
    {
        if (character[i] >= 97 && character[i] <= 122)
        {
            character[i] = character[i] - 32;
        }
    }

    printf("%s", character);

    printf("\n\n");

    return 0;
}
```

This program outputs:



The screenshot shows a Windows Command Prompt window titled 'cmd' with the path 'C:\Windows\system32\cmd.exe'. The window displays a 10x10 grid of characters, alternating between 'black' and 'white'. The text is in a monospaced font. At the bottom of the grid, there is a message in Portuguese: 'Pressione qualquer tecla para continuar. . .' (Press any key to continue. .).

Figure 221: Checkerboard Pattern Output.

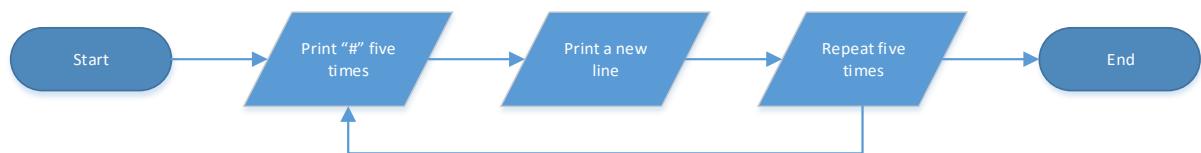
6.3.17 Exercise 17

This program should use just single # and new line calls to print the following pattern:

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

Figure 222: Nested for Loop Square.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    for (int i = 0; i < 5; i++)
    {
        for (int n = 0; n < 5; n++)
        {
            printf("#");
        }
        printf("\n");
    }

    printf("\n\n");

    return 0;
}
```

This program outputs:



```
C:\Windows\system32\cmd.exe
#####
#####
#####
#####
#####
#####
Pressione qualquer tecla para continuar. . .
```

Figure 223: Nested for Loop Square Output.

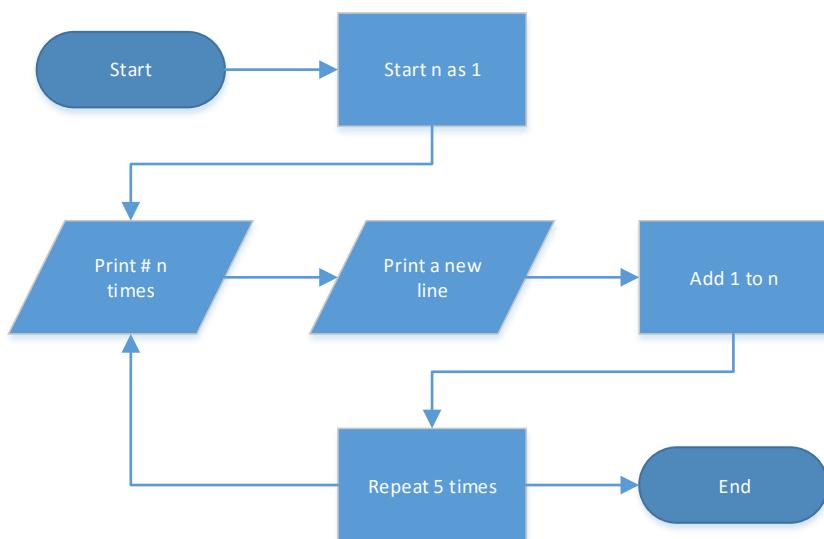
6.3.18 Exercise 18

This program should use just printf calls with # and new lines to print the following pattern:

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

Figure 224: Nested for Loop Triangle.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    int i = 1;
```

```
while (i <= 5)
{
    for (int n = 0; n < i; n++)
    {
        printf("#");
    }

    printf("\n");

    i++;
}

printf("\n\n");

return 0;
}
```

This program outputs:



```
C:\Windows\system32\cmd.exe
#
##
###
####
#####
#####

Pressione qualquer tecla para continuar. . .
```

Figure 225: Nested for Loop Triangle Output.

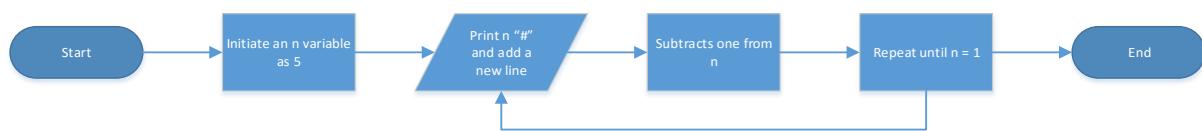
6.3.19 Exercise 19

This program, as the previous program, should use just printf calls with new lines and # to output:

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

Figure 226: Nested for Loop Inverse Triangle.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    int i;

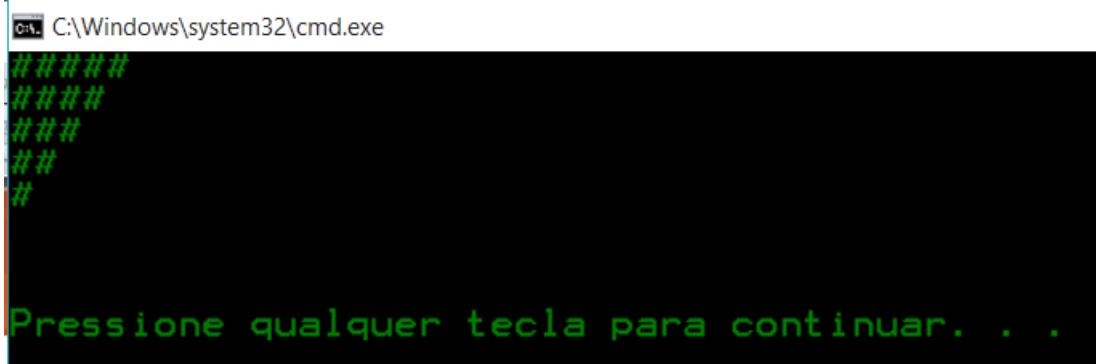
    for (i = 5; i >= 0; i--)
    {
        for (int n = 0; n < i; n++)
        {
            printf("#");
        }

        printf("\n");
    }
}
```

```
    printf("\n\n");

    return 0;
}
```

This program outputs:



The screenshot shows a Windows command prompt window titled 'cmd' with the path 'C:\Windows\system32\cmd.exe'. Inside the window, there is a series of '#' characters forming an inverse triangle pattern. The pattern starts with four '#' on the top row, followed by three '#', two '#', and one '#' on subsequent rows. Below this pattern, the text 'Pressione qualquer tecla para continuar. . .' is displayed, indicating the program is waiting for user input to close.

Figure 227: Nested for Loop Inverse Triangle Output.

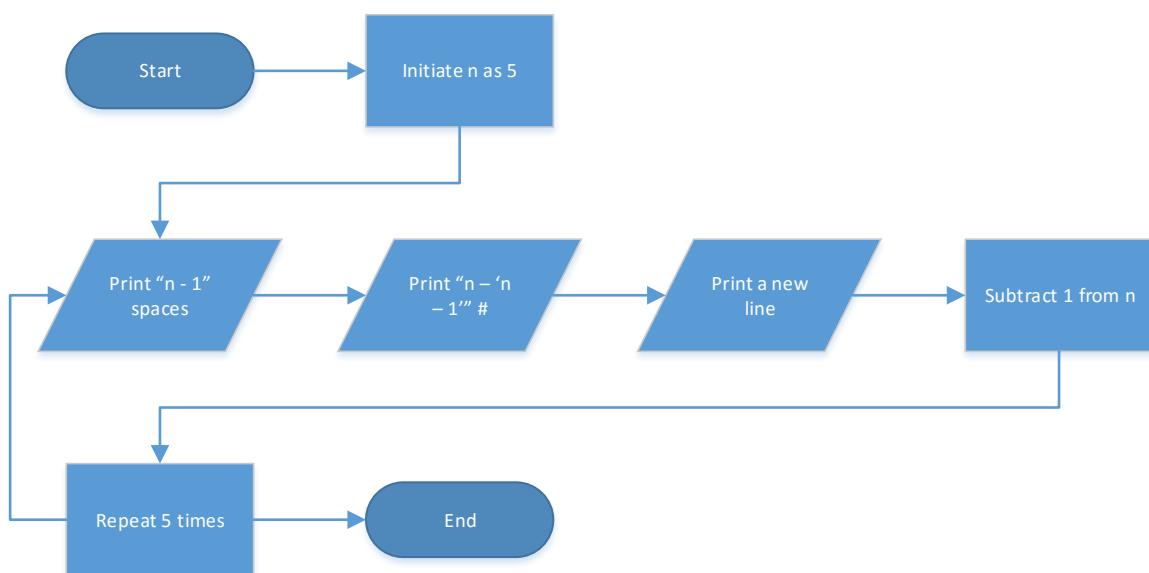
6.3.20 Exercise 20

This program should output a right aligned triangle using just for loops with printf calls of spaces, #, and new lines.

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

Figure 228: Nested for Loop Triangle, Right Aligned.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>  
  
int main()  
Paludo
```

```
{  
    for (int i = 5; i > 0; i--)  
    {  
        for (int a = i - 1; a > 0; a--)  
        {  
            printf(" ");  
        }  
  
        for (int n = 5 - (i - 1); n > 0; n--)  
        {  
            printf("#");  
        }  
  
        printf("\n");  
    }  
  
    printf("\n\n");  
  
    return 0;  
}
```

This program outputs:



The screenshot shows a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The output of the program is a right-aligned triangle made of '#' characters. The triangle has 5 rows, starting with one '#' in the first row and increasing by one each subsequent row. Below the triangle, the text 'Pressione qualquer tecla para continuar...' (Press any key to continue...) is displayed in green.

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

Pressione qualquer tecla para continuar. . .

Figure 229: Nested for Loop Triangle, Right Aligned Output.

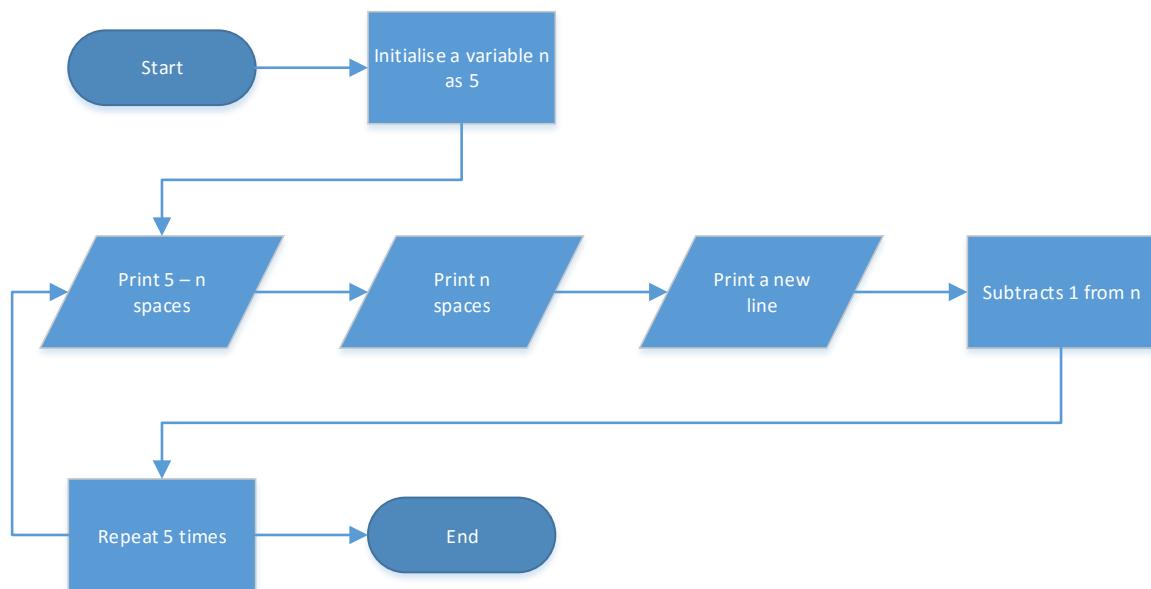
6.3.21 Exercise 21

This program, should print a inverted, right aligned inverse triangle using #, spaces and new lines printf calls.

#

Figure 230: Nested for Loop Inverse Triangle, Right Aligned.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    for (int i = 5; i > 0; i--)
```

```
{  
    for (int n = 5 - i; n > 0; n--)  
    {  
        printf(" ");  
    }  
  
    for (int a = i; a > 0; a--)  
    {  
        printf("#");  
    }  
  
    printf("\n");  
}  
  
printf("\n\n");  
  
return 0;  
}
```

This program outputs:



```
C:\Windows\system32\cmd.exe  
#####  
 #####  
 ####  
 ##  
 #  
  
Pressione qualquer tecla para continuar. . .
```

Figure 231: Nested for Loop Inverse Triangle, Right Aligned.

6.3.22 Exercise 22

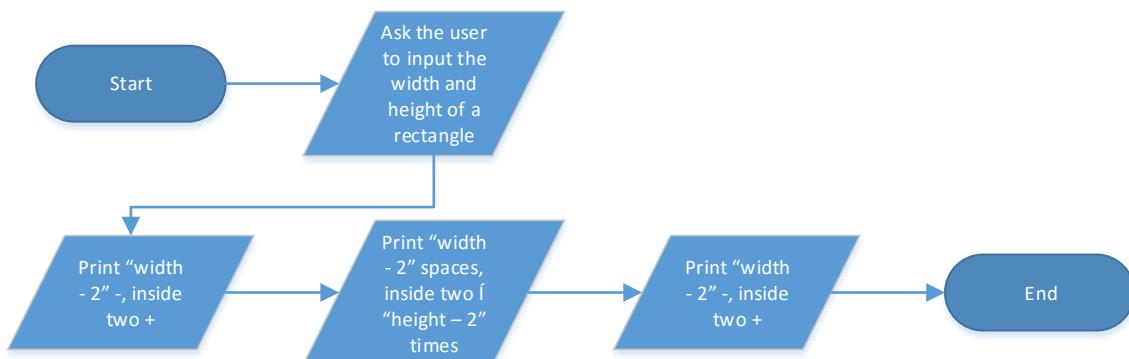
This program asks the user to input the height and width of a rectangle, as shown in the example:

```
Rectangle width? 8
Rectangle height? 4

+-----+
|       |
|       |
+-----+
```

Figure 232: Rectangle Drawing.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    int a, b, c;

    int width;
    printf("Rectangle width? ");
    scanf("%d", &width);
    printf("Rectangle height? ");
    scanf("%d", &height);

    for (c = 1; c <= height; c++) {
        for (a = 1; a <= width; a++) {
            if ((a == 1) || (a == width) || (c == 1) || (c == height)) {
                printf("+");
            } else {
                printf(" ");
            }
        }
        printf("\n");
    }
}
```

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

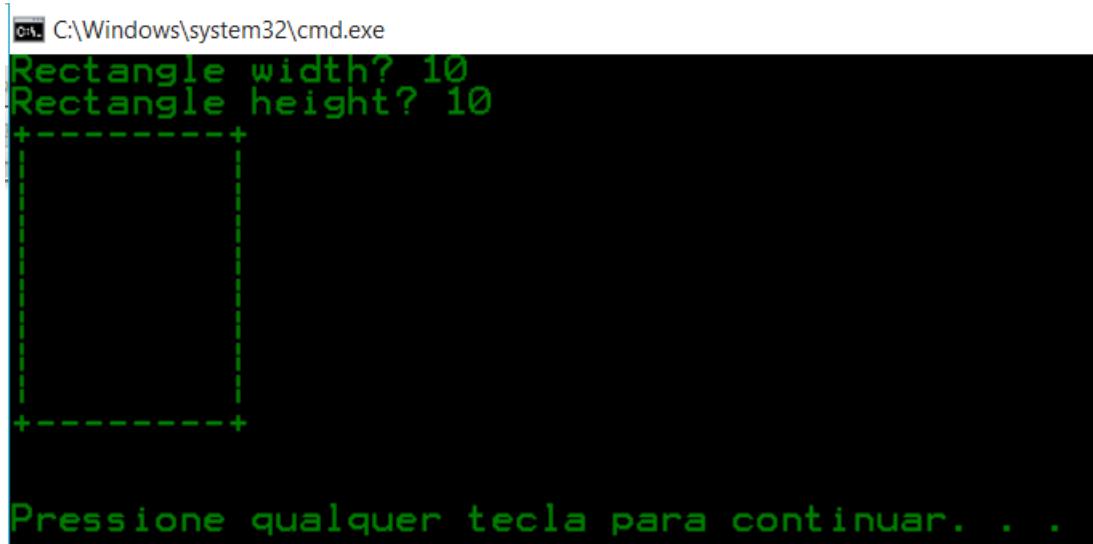
int height;
printf("Rectangle height? ");
scanf("%d", &height);

for (a = height; a > 0; a--)
{
    if (a == height || a == 1)
    {
        for (b = width; b > 0; b--)
        {
            if (b == width || b == 1)
            {
                printf("+");
            }
            else
            {
                printf("-");
            }
        }
        printf("\n");
    }
    else
    {
        for (c = width; c > 0; c--)
        {
            if (c == width || c == 1)
            {
                printf("|");
            }
            else
            {
                printf(" ");
            }
        }
        printf("\n");
    }
}

printf("\n\n");

return 0;
}
```

This program outputs:



A screenshot of a Windows command-line window titled "C:\Windows\system32\cmd.exe". The window displays the following text:

```
Rectangle width? 10
Rectangle height? 10
+-----+
|       |
|       |
|       |
|       |
+-----+
Pressione qualquer tecla para continuar. . .
```

The window shows a dashed rectangle outline with a width of 10 and a height of 10. The prompt "Pressione qualquer tecla para continuar. . ." (Press any key to continue. . .) is at the bottom.

Figure 233: Rectangle Drawing Output.

6.3.23 Exercise 23

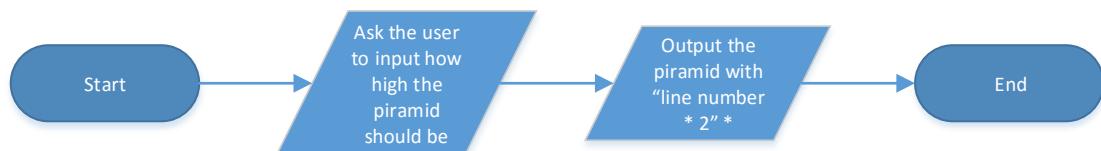
This program asks the user to input the highness of a pyramid and then output a pyramid, as in the example:

```
How high would you like the pyramid? 7

    **
    ****
    *****
    ******
    *****
    *****
    *****
```

Figure 234: Loop Pyramid.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    int high;
    printf("How high would you like the pyramid? ");
    scanf("%d", &high);

    for (int line = 0; line < high; line++)
    {
        for (int spaces = high - line; spaces > 0; spaces--)
        {
```

```

        printf(" ");
    }

    for (int blocks = 0; blocks < line * 2; blocks++)
    {
        printf("*");
    }

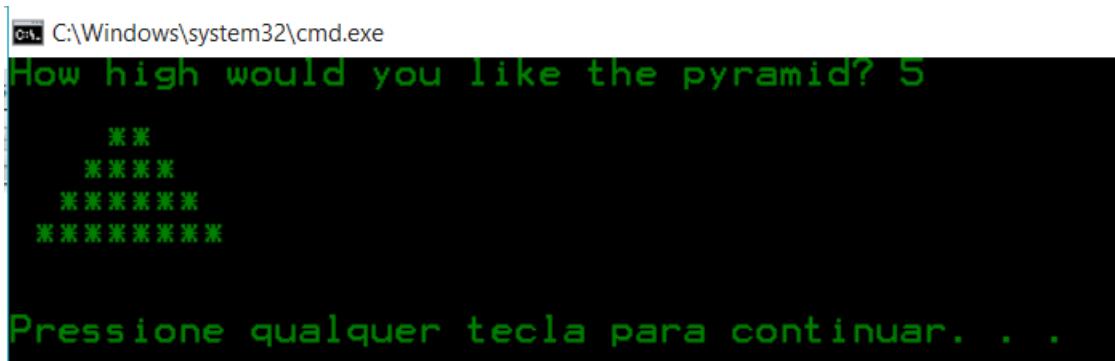
    printf("\n");
}

printf("\n\n");

return 0;
}

```

But this code was outputting one line less than it should and one more space in every line:



```

C:\Windows\system32\cmd.exe
How high would you like the pyramid? 5
      *
      **
      ***
      ****
      *****
      ******
      *****
Pressione qualquer tecla para continuar. . .

```

Figure 235: Loop Pyramid Wrong Output.

To fix these errors, I wrote the following code:

```

#include <stdio.h>

int main()
{
    int high;
    printf("How high would you like the pyramid? ");
    scanf("%d", &high);

    for (int line = 0; line <= high; line++)
    {
        for (int spaces = high - line; spaces > 0; spaces--)

```

```
{           printf(" ");  
}  
  
for (int blocks = 0; blocks < line * 2; blocks++)  
{           printf("*");  
}  
  
printf("\n");  
}  
  
printf("\n\n");  
  
return 0;  
}
```

This program outputs:

Figure 236: Loop Pyramid Output.

6.3.24 Exercise 24

This program should use loop structures to print a holiday tree with heightness determinated by the user.

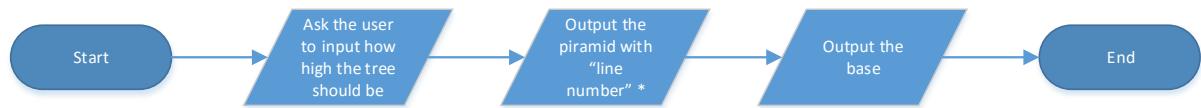
```
How high would you like the tree? 5
```

```

*
 *
 *
 *
 *
 *
   \ /
```

Figure 237: Holiday Tree.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int main()
{
    int rows;
    printf("How high would you like the tree?");
    scanf("%d", &rows);

    for (int line = 0; line <= rows; line++)
    {
        if (rows == 1)
        {
            printf("  ");
        }
        else
        {
            for (int i = 0; i < line; i++)
            {
                printf("*");
            }
            printf("\n");
        }
    }
}
```

```
    }

    else if (rows == 2)
    {
        printf(" ");
    }

    for (int spaces = rows - line; spaces > 0; spaces--)
    {
        printf(" ");
    }

    for (int blocks = 0; blocks < line; blocks++)
    {
        printf("* ");
    }

    printf("\n");
}

for (int base = 0; base < rows - 3; base++)
{
    printf(" ");
}
printf("_|_|_\n");

for (int base = 0; base < rows - 3; base++)
{
    printf(" ");
}
printf("\\\\____/\n");

printf("\n\n");

return 0;
}
```

This program outputs:



A terminal window titled "C:\Windows\system32\cmd.exe" displays a Christmas tree pattern. The tree is composed of asterisks (*) and has a total height of 15 rows. The pattern is symmetrical, with the widest part at the base and tapering towards the top. At the very bottom of the tree, there is a small decorative element consisting of two vertical bars and a horizontal bar connecting them.

```
How high would you like the tree?15
 *
 * *
 * * *
 * * * *
 * * * * *
 * * * * * *
 * * * * * * *
 * * * * * * * *
 * * * * * * * * *
 * * * * * * * * * *
 * * * * * * * * * * *
 * * * * * * * * * * * *
 * * * * * * * * * * * * *
Pressione qualquer tecla para continuar. . .
```

Figure 238: Holiday Tree Output.

7 Week Seven

7.1 Lectures

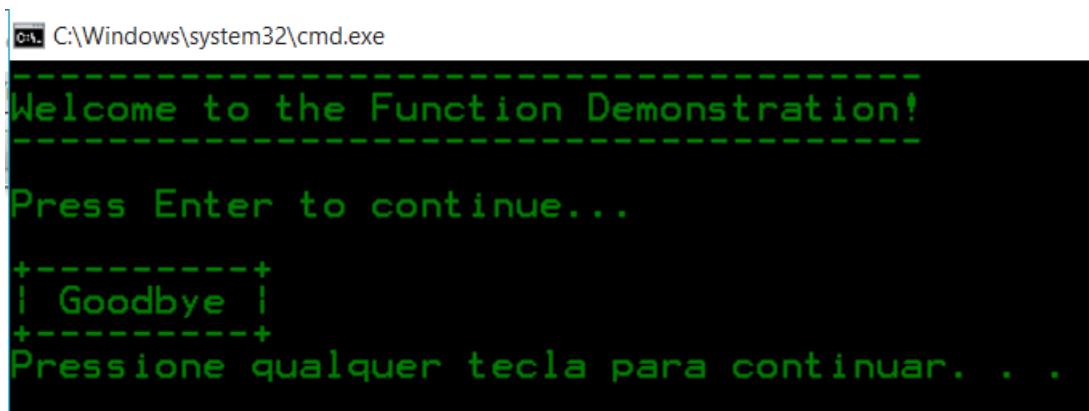
Modularity is the ability to divide a program into smaller pieces, this makes the code easier to manage and understand. The first step to do so, is to create reusable functions with clear purpose with an auto explaining name and does a single job with success.

To avoid compiling errors, these functions may be defined before the main using a prototype.

7.2 Examples

7.2.1 Example 1

This program outputs a welcome and wait for the user input, then print a goodbye message using functions. The program outputs:



The screenshot shows a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:

Welcome to the Function Demonstration!

Press Enter to continue...
+-----+
| Goodbye |
+-----+
Pressione qualquer tecla para continuar. . .

Figure 239: Simple void Functions.

This was the written code:

```
#include <stdio.h>

void wait_for_enter()
{
    char input = 0;
    scanf("%c", &input);
}

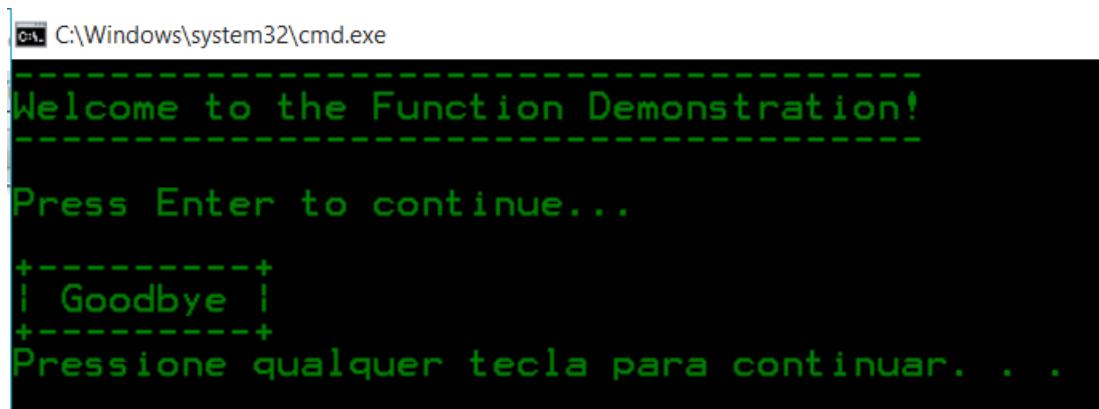
void print_welcome()
{
    printf("-----\n");
    printf("Welcome to the Function Demonstration!\n");
    printf("-----\n");
}

void print_goodbye()
```

```
{  
    printf("-----+\n");  
    printf(" | Goodbye | \n");  
    printf("-----+\n");  
}  
  
void print_prompt()  
{  
    printf("\nPress Enter to continue...\n");  
}  
  
int main()  
{  
    print_welcome();  
  
    print_prompt();  
  
    wait_for_enter();  
  
    print_goodbye();  
  
    return 0;  
}
```

7.2.2 Example 2

This program calls the main before the other functions and uses prototypes to call the other functions and prevent compiling bugs, the program's output is the same as the previous program.



The screenshot shows a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The output of the program is displayed in green text:

```
Welcome to the Function Demonstration!
-----
Press Enter to continue...
+-----+
| Goodbye |
+-----+
Pressione qualquer tecla para continuar. . .
```

Figure 240: Simple Functions with Prototypes.

This was the written code:

```
#include <stdio.h>

void wait_for_enter();
void print_welcome();
void print_goodbye();
void print_prompt();

int main()
{
    print_welcome();

    print_prompt();

    wait_for_enter();

    print_goodbye();
```

```
    return 0;
}

void wait_for_enter()
{
    char input = 0;
    scanf("%c", &input);
}

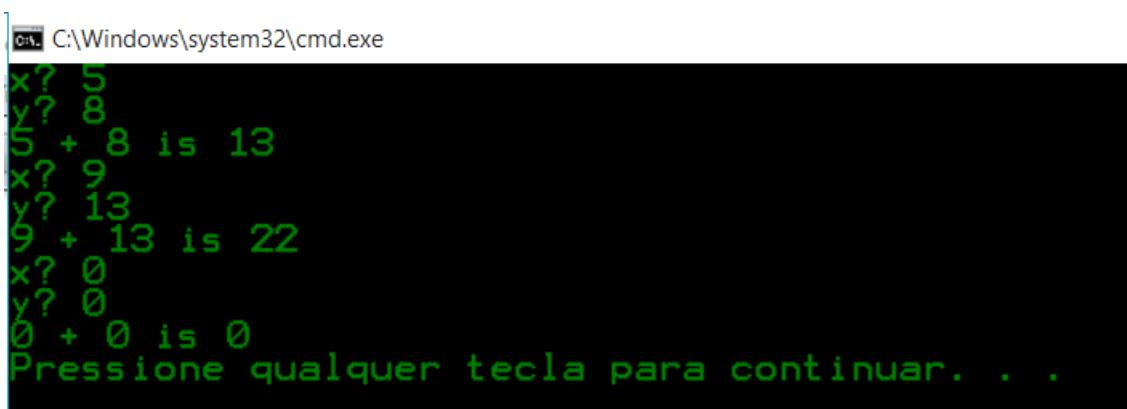
void print_welcome()
{
    printf("-----\n");
    printf("Welcome to the Function Demonstration!\n");
    printf("-----\n");
}

void print_goodbye()
{
    printf("-----+\n");
    printf("| Goodbye |\n");
    printf("-----+\n");
}

void print_prompt()
{
    printf("\nPress Enter to continue...\n");
}
```

7.2.3 Example 3

This program asks the user to input two numbers and sums these numbers using functions, the program outputs:



```
C:\Windows\system32\cmd.exe
x? 5
y? 8
5 + 8 is 13
x? 9
y? 13
9 + 13 is 22
x? 0
y? 0
0 + 0 is 0
Pressione qualquer tecla para continuar. . .
```

Figure 241: Simple Function with return.

This was the written code:

```
#include <stdio.h>

int add(int a, int b)
{
    return a + b;
}

int main()
{
    int x = 0;
    int y = 0;

    do
    {
        printf("x? ");
        scanf("%d", &x);

        printf("y? ");
        scanf("%d", &y);

        printf("x + y is %d\n", add(x, y));
    }
}
```

```
int result = add(x, y);

printf("%d + %d is %d\n", x, y, result);
}
while (!(0 == x || 0 == y));

return 0;
}
```

7.2.4 Example 4

This program uses an infinite loop that calls functions which take a number as input from the user and counts the number of the steps, the program outputs:

```
C:\Windows\system32\cmd.exe
To quit, press Ctrl-C...
> 3
Action 0
Action 1
Action 2
To quit, press Ctrl-C...
> 0
To quit, press Ctrl-C...
> 1
Action 0
To quit, press Ctrl-C...
> 2
Action 0
Action 1
To quit, press Ctrl-C...
>
> 2
Action 0
Action 1
To quit, press Ctrl-C...
> ^CPressione qualquer tecla para continuar. . .
```

Figure 242: Example Function Usage.

This was the written code:

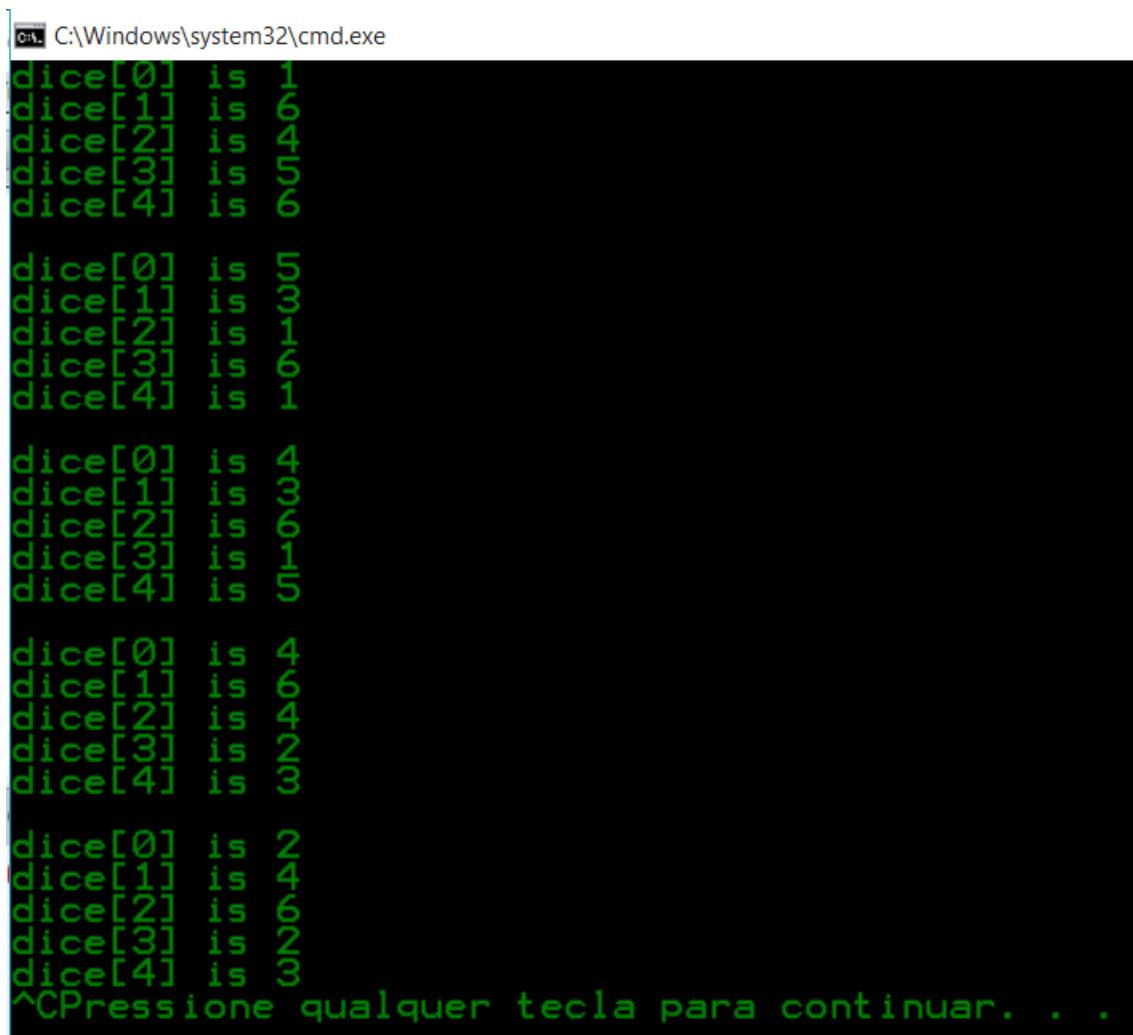
```
#include <stdio.h>

void do_action(int repeat_count)
```

```
{  
    for (int i = 0; i < repeat_count; ++i)  
    {  
        printf("Action %d\n", i);  
    }  
}  
  
int query_for_number()  
{  
    int number = 0;  
    scanf("%d", &number);  
  
    return number;  
}  
  
void print_prompt()  
{  
    printf("> ");  
}  
  
void print_quit_reminder()  
{  
    printf("To quit, press Ctrl-C...\\n\\n");  
}  
  
int main()  
{  
    while (1)  
    {  
        print_quit_reminder();  
        print_prompt();  
  
        int num = query_for_number();  
        do_action(num);  
    }  
  
    return 0;  
}
```

7.2.5 Example 5

This program rolls 5 dices randomly and waits for the user to input enter. The program outputs:



```
C:\Windows\system32\cmd.exe
dice[0]  is 1
dice[1]  is 6
dice[2]  is 4
dice[3]  is 5
dice[4]  is 6

dice[0]  is 5
dice[1]  is 3
dice[2]  is 1
dice[3]  is 6
dice[4]  is 1

dice[0]  is 4
dice[1]  is 3
dice[2]  is 6
dice[3]  is 1
dice[4]  is 5

dice[0]  is 4
dice[1]  is 6
dice[2]  is 4
dice[3]  is 2
dice[4]  is 3

dice[0]  is 2
dice[1]  is 4
dice[2]  is 6
dice[3]  is 2
dice[4]  is 3
^CPressione qualquer tecla para continuar. . .
```

Figure 243: Random Numbers Helper Functions.

This was the written code:

```
#include <stdio.h>
```

Paludo

```
#include <stdlib.h>
#include <time.h>

void seed_random();
int get_random(int low, int high);
void wait_for_enter();

int main()
{
    seed_random();

    do
    {
        int dice[5];

        for (int i = 0; i < 5; ++i)
        {
            dice[i] = get_random(1, 6);
        }

        for (int i = 0; i < 5; ++i)
        {
            printf("dice[%d] is %d\n", i, dice[i]);
        }

        wait_for_enter();
    }
    while (1);

    return 0;
}

void seed_random()
{
    srand(time(0));
}

int get_random(int low, int high)
{
    return ((rand() % (high - low + 1)) + low);
}

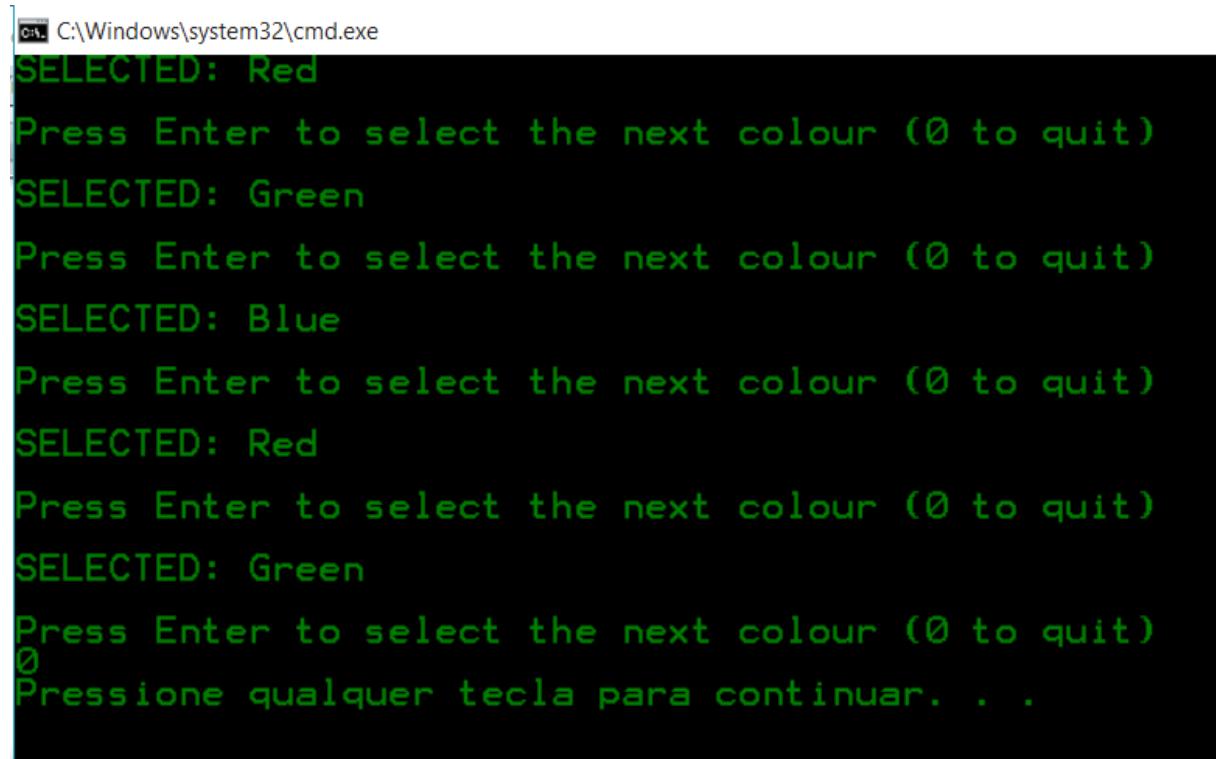
void wait_for_enter()
{
    char input = 0;
    scanf("%c", &input);
}
```

14885857

Reporting Journal #4

7.2.6 Example 6

This program uses an ENUM class to enumerate colours and outputs the selected color, the user must type 0 to exit. The program outputs:



The screenshot shows a command-line interface (cmd.exe) running on a Windows system. The window title is 'C:\Windows\system32\cmd.exe'. The text output is as follows:

```
SELECTED: Red
Press Enter to select the next colour (0 to quit)
SELECTED: Green
Press Enter to select the next colour (0 to quit)
SELECTED: Blue
Press Enter to select the next colour (0 to quit)
SELECTED: Red
Press Enter to select the next colour (0 to quit)
SELECTED: Green
Press Enter to select the next colour (0 to quit)
0
Pressione qualquer tecla para continuar. . .
```

Figure 244: enum with Loop.

This was the written code:

```
#include <stdio.h>

int main()
{
    enum Colour
    {
        RED,
        GREEN,
        BLUE,
```

```
    MAX_COLOUR
};

char keypress = 0;
enum Colour current = RED;

do
{
    if (current >= MAX_COLOUR)
    {
        current = RED;
    }

    switch (current)
    {
        case RED:
        {
            printf("SELECTED: Red");
        }
        break;
        case GREEN:
        {
            printf("SELECTED: Green");
        }
        break;
        case BLUE:
        {
            printf("SELECTED: Blue");
        }
        break;
        default:
        {
            printf("SELECTED: Error!");
        }
        break;
    }

    printf("\n\nPress Enter to select the next colour (0 to
quit)\n");
    int next = current + 1;
    current = next;

    scanf("%c", &keypress);
}
while ('0' != keypress);

return 0;
```

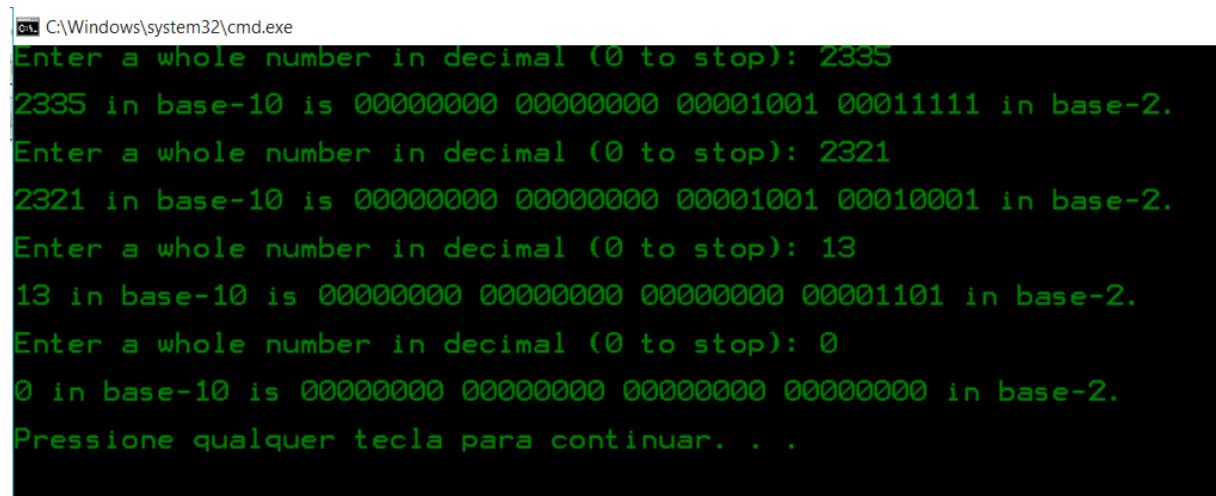
14885857

Reporting Journal #4

}

7.2.7 Example 7

This program takes a whole number as input from the user and converts from decimal to binary. The program outputs:



```
C:\Windows\system32\cmd.exe
Enter a whole number in decimal (0 to stop): 2335
2335 in base-10 is 00000000 00000000 00001001 00011111 in base-2.
Enter a whole number in decimal (0 to stop): 2321
2321 in base-10 is 00000000 00000000 00001001 00010001 in base-2.
Enter a whole number in decimal (0 to stop): 13
13 in base-10 is 00000000 00000000 00000000 00001101 in base-2.
Enter a whole number in decimal (0 to stop): 0
0 in base-10 is 00000000 00000000 00000000 00000000 in base-2.
Pressione qualquer tecla para continuar. . .
```

Figure 245: int Bit Pattern.

This was the written code:

```
#include <stdio.h>

int main()
{
    int input = 0;

    do
    {
        printf("Enter a whole number in decimal (0 to stop): ");
        scanf("%d", &input);

        int num_bytes = sizeof(int);
        int num_bits = num_bytes * 8;

        printf("\n%d in base-10 is", input);

        for (int i = num_bits - 1; i >= 0; --i)
```

```
{  
    int bit_to_check = (1 << i);  
  
    if (0 == (i + 1) % 8)  
    {  
        printf(" ");  
    }  
  
    if (input & bit_to_check)  
    {  
        printf("1");  
    }  
    else  
    {  
        printf("0");  
    }  
}  
  
printf(" in base-2.\n\n");  
}  
while (0 != input);  
  
return 0;  
}
```

7.2.8 Example 8

This program uses a enum class for a switch case structure. The program outputs:



The screenshot shows a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
Option 2
Option 4
Option 5
Pressione qualquer tecla para continuar. . .

Figure 246: Bit Flags.

This was the written code:

```
#include <stdio.h>

enum Flag
{
    OPTION1 = 1,
    OPTION2 = 2,
    OPTION3 = 4,
    OPTION4 = 8,
    OPTION5 = 16,
    OPTION6 = 32
};

void print_flag(int flag)
{
    switch (flag)
    {
        case OPTION1:
        {
            printf("Option 1\n");
        }
        break;
        case OPTION2:
        {
            printf("Option 2\n");
        }
        break;
        case OPTION3:
```

```
{           printf("Option 3\n");
}
break;
case OPTION4:
{
    printf("Option 4\n");
}
break;
case OPTION5:
{
    printf("Option 5\n");
}
break;
case OPTION6:
{
    printf("Option 6\n");
}
break;
default:
{
    printf("Unknown Flag!\n");
}
break;
}
}
void print_status(int status)
{
    for (int k = 0; k < 6; ++k)
    {
        int bit_to_check = 1 << k;
        if (status & bit_to_check)
        {
            print_flag(bit_to_check);
        }
    }
}
int main()
{
    int main_status = OPTION2 | OPTION4 | OPTION5;
    print_status(main_status);
    return 0;
}
```

7.3 Exercises

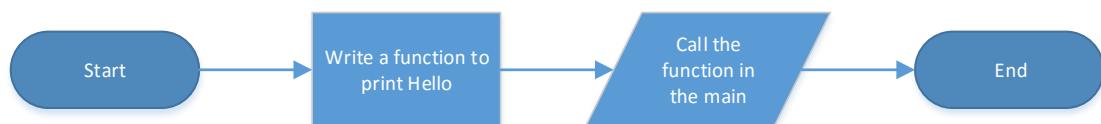
7.3.1 Exercise 1

This program uses a void function to print Hello on the screen.

Hello

Figure 247: Say Hello.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

void say_hello()
{
    printf("Hello \n");

    return 0;
}
int main()
{
    say_hello();

    printf("\n");

    return 0;
}
```

This code outputs:



A screenshot of a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
Hello
Pressione qualquer tecla para continuar. . .

Figure 248: Say Hello Output.

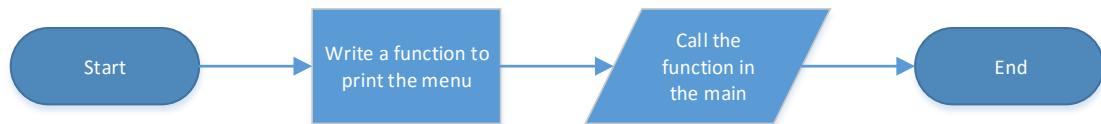
7.3.2 Exercise 2

This program should have a function that prints a menu, as shown in the example:

```
Calculator Menu:  
-----  
1: Add two numbers  
2: Subtract two numbers  
3: Multiply two numbers  
4: Divide two numbers  
5: Quit
```

Figure 249: Print Menu.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

void print_menu()
{
    printf("Calculator Menu:\n");
    printf("-----\n");
    printf("\n");
    printf("1: Add two numbers\n");
    printf("2: Subtract two numbers\n");
    printf("3: Multiply two numbers\n");
    printf("4: Divide two numbers\n");
    printf("5: Quit\n");

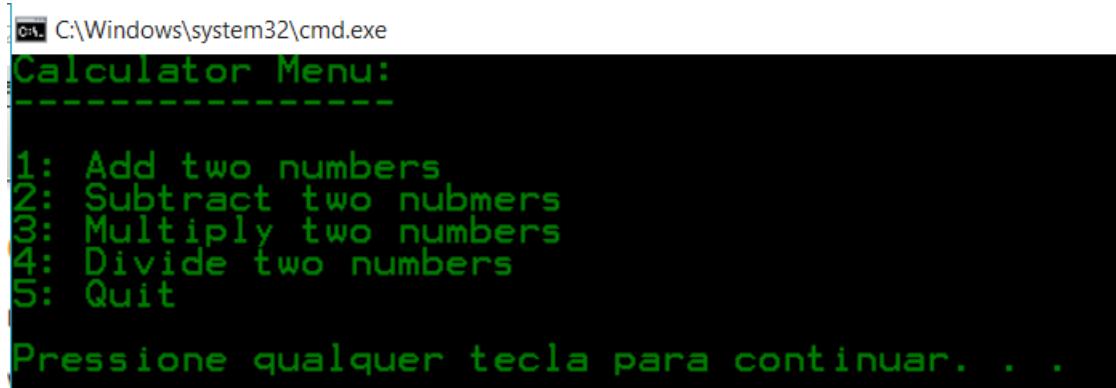
    return 0;
}
```

```
int main()
{
    print_menu();

    printf("\n");

    return 0;
}
```

This program outputs:



A screenshot of a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The window displays a menu titled 'Calculator Menu:' followed by a list of five options numbered 1 through 5. Option 5 is 'Quit'. Below the menu, there is a message in Portuguese: 'Pressione qualquer tecla para continuar. . .'. The background of the window is black, and the text is white.

```
C:\Windows\system32\cmd.exe
Calculator Menu:
-----
1: Add two numbers
2: Subtract two numbers
3: Multiply two numbers
4: Divide two numbers
5: Quit

Pressione qualquer tecla para continuar. . .
```

Figure 250: Print Menu Output.

7.3.3 Exercise 3

This program should have a function that takes two whole numbers as input and returns the minimum number.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

int find_minimum(int a, int b, int c)
{
    if (a > b)
    {
        return b;
    }
    else
    {
        return a;
    }
}
int main()
{
    printf("Please input two whole numbers:\n");

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

    int b;
    scanf("%d", &b);

    printf("\n");

    int min_result = find_minimum(a, b);
    printf("The minimum is: %d", min_result);

    printf("\n\n");

    return 0;
}
  
```

}

This program outputs:

```
C:\Windows\system32\cmd.exe
Please input two whole numbers:
1000
100

The minimum is: 100
Pressione qualquer tecla para continuar. . .
```

Figure 251: Find Minimum Output.

7.3.4 Exercise 4

This program asks the user to input three real numbers, calls a function that takes these numbers as input parameters and returns the biggest number, the program then, output the result.

To develop this program, I draw the following flowchart:



But I had a lot of problems writing the code. I usually write each function in a separate file, during the lab I forgot to call the functions before the main, it was working well for the other programs, but it did not work with this one. I tried several approaches, changing the file names, changing the variable names, using the function as a header and calling it before the main, cleaning the solution, closing the program and trying again, restarting my computer, rebuilding the solution, always debugging and trying it in different programs as well, such as CodeBlocks and CodeBunk. The program works well if I change the variables to integers.

I wrote the following code to try making the program work:

```

#include <stdio.h>

float find_maximum(float a, float b, float c);

int main()
{
    printf("Enter three real numbers\n");

    float a;
    scanf("%f", &a);
  
```

```
float b;
scanf("%f", &b);

float c;
scanf("%f", &c);

printf("\n");

float max_result = find_maximum(a, b, c);
printf("The maximum is: %f", max_result);

printf("\n\n");

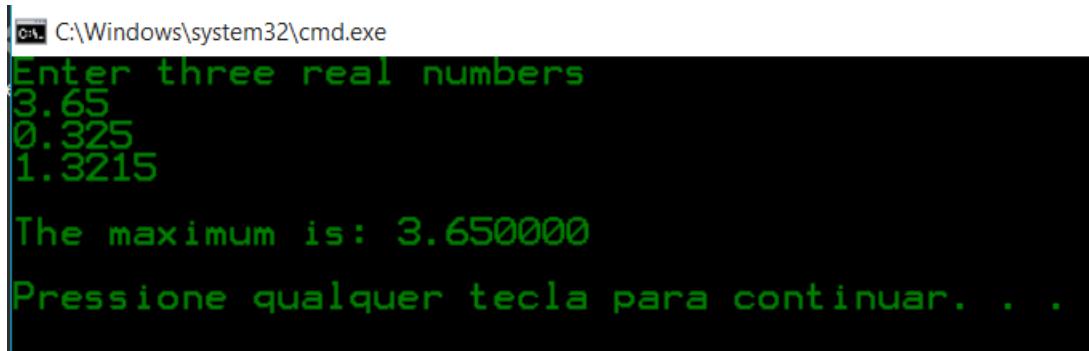
return 0;
}
```

And, in another file:

```
#include <stdio.h>

float find_maximum(float a, float b, float c)
{
    if (a > b && a > c)
    {
        return a;
    }
    else if (b > a && b > c)
    {
        return b;
    }
    else
    {
        return c;
    }
}
```

This program outputs:



A screenshot of a Windows command-line window titled "C:\Windows\system32\cmd.exe". The window contains the following text:
Enter three real numbers
3.65
0.325
1.3215
The maximum is: 3.650000
Pressione qualquer tecla para continuar. . .

Figure 252: Find Maximum Output.

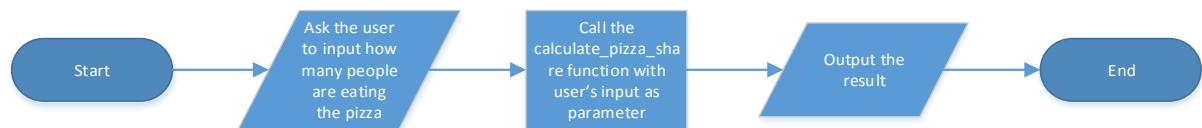
7.3.5 Exercise 5

This program has a function `calculate_pizza_share` which takes an integer as parameter for the number of people that are going to eat the pizza. Each pizza has eight slices, the function returns the number of slices each person can eat. The program asks the user to input a number in the main function and outputs the result.

```
How many people? 2
2 people get 4 slice(s) each.
```

Figure 253: Pizza Calculator.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int calculate_pizza_share(int number_of_people);

int main()
{
    int number_of_people;
    printf("How many people? ");
    scanf("%d", &number_of_people);

    int slices_per_person =
    calculate_pizza_share(number_of_people);
    printf("%d people get %d slice(s) each.", number_of_people,
    slices_per_person);

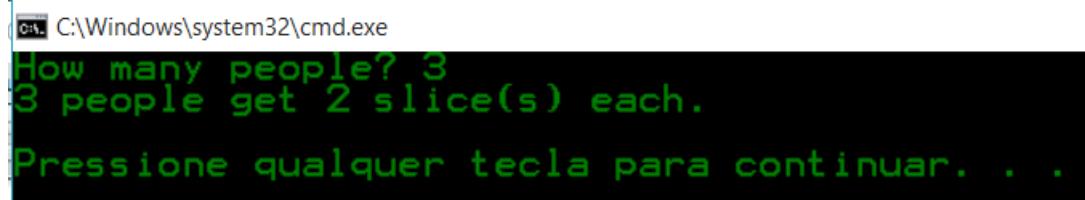
    printf("\n\n");
}
```

```
    return 0;
}

int calculate_pizza_share(int number_of_people)
{
    int slices_per_person = 8 / number_of_people;

    return slices_per_person;
}
```

This program outputs:



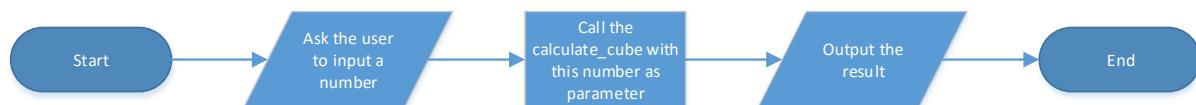
A screenshot of a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
How many people? 3
3 people get 2 slice(s) each.
Pressione qualquer tecla para continuar. . .

Figure 254: Pizza Calculator Output.

7.3.6 Exercise 6

This program takes an input from the user, calls a function using this number as parameter and returns the cube of this number.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

int calculate_cube(int input);

int main()
{
    int input;
    printf("Input a number: ");
    scanf("%d", &input);

    int cube = calculate_cube(input);
    printf("The cube of %d is %d.", input, cube);

    printf("\n\n");

    return 0;
}

int calculate_cube(int input)
{
    int cube = pow(input, 3);

    return cube;
}
  
```

This program outputs:



The screenshot shows a terminal window titled 'cmd' running on 'C:\Windows\system32\cmd.exe'. The user has input '3' and the program outputs 'The cube of 3 is -15152.' followed by a prompt to press any key to continue.

Figure 255: Cube Function Wrong Output.

This program does not work properly, I noticed that I forgot to include the math library.

After solving this problem, this was the written code:

```
#include <stdio.h>

int calculate_cube(int input);

int main()
{
    int input;
    printf("Input a number: ");
    scanf("%d", &input);

    int cube = calculate_cube(input);
    printf("The cube of %d is %d.", input, cube);

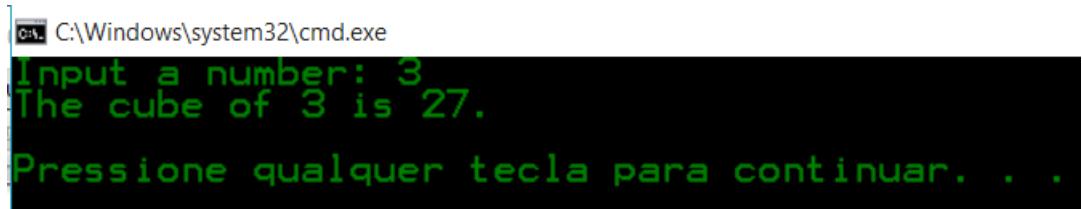
    printf("\n\n");

    return 0;
}
#include <stdio.h>
#include <math.h>

int calculate_cube(int input)
{
    int cube = pow(input, 3);

    return cube;
}
```

This program outputs:



A screenshot of a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
Input a number: 3
The cube of 3 is 27.
Pressione qualquer tecla para continuar. . .

Figure 256: Cube Function.

7.3.7 Exercise 7

This program has a function that takes in no parameters and returns a whole number.

The function asks the user to input a number for a choice, and returns this number. The program calls the function in the main, saving the result in a variable.

To develop this program, I draw the following flowchart:



This was the written code:

```

#include <stdio.h>

int get_user_choice();

int main()
{
    int choice = get_user_choice();
    printf("Your choice was: %d", choice);

    printf("\n\n");

    return 0;
}

int get_user_choice()
{
    int choice;
    printf("Enter your choice: ");
    scanf("%d", &choice);

    return choice;
}
  
```

This program outputs:



A screenshot of a Windows command prompt window titled 'cmd C:\Windows\system32\cmd.exe'. The window contains the following text:
Enter your choice: 3
Your choice was: 3
Pressione qualquer tecla para continuar. . .

Figure 257: Get User Choice Output.

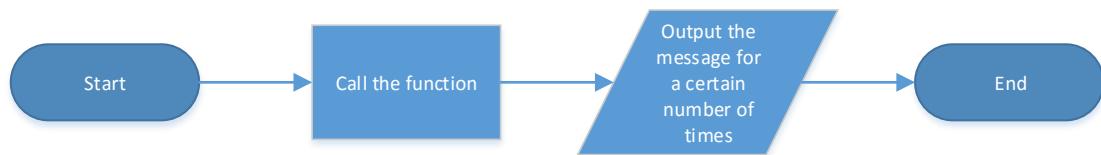
7.3.8 Exercise 8

A function called in the main take as input a whole number and returns this number of welcome messages. The program should output:

```
Welcome to the Week 7 Lab!
-----
Welcome to the Week 7 Lab!
-----
Welcome to the Week 7 Lab!
-----
Welcome to the Week 7 Lab!
```

Figure 258: Repeated Welcome.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

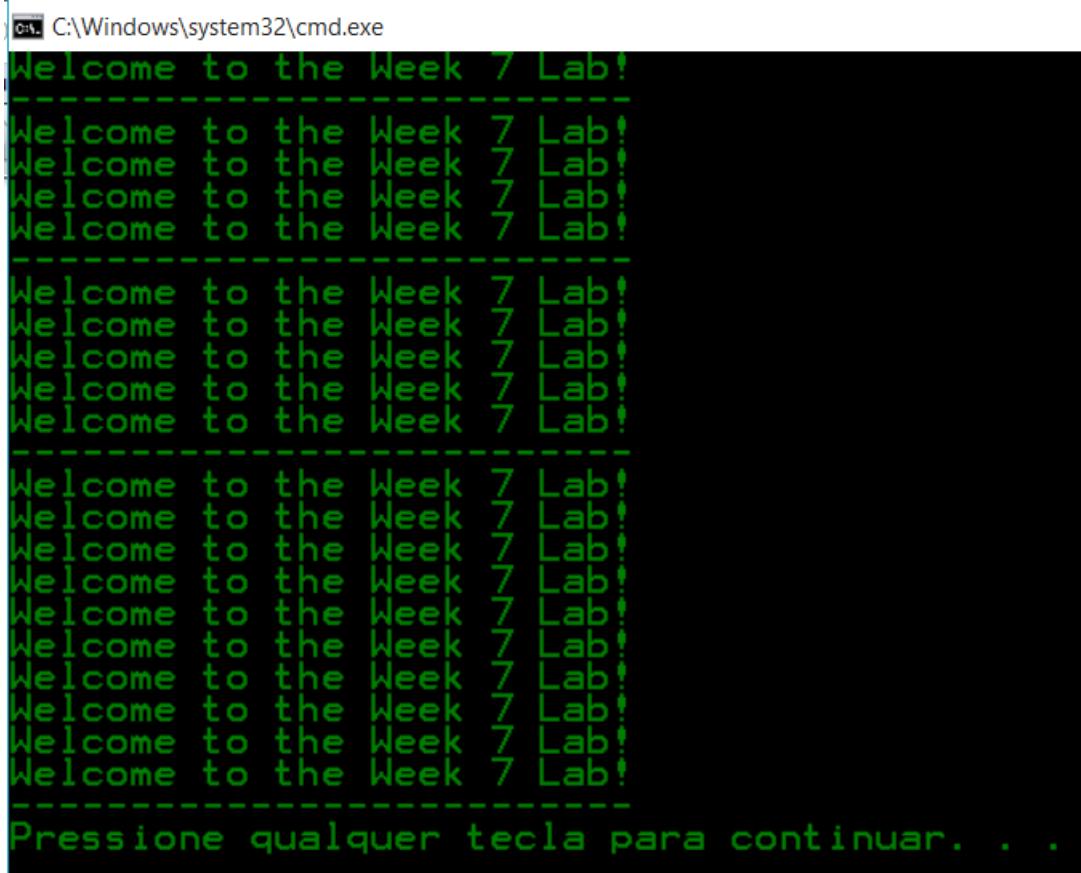
int main()
{
    int count;
    scanf("%d", &count);
    repeated_welcome(count);

}

void repeated_welcome(int count)
{
    for (int i = 0; i < count; i++)
    {
        printf("Welcome to the Week 7 Lab!\n");
    }

    printf("-----\n");
}
```

This program outputs:



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:

```
Welcome to the Week 7 Lab!
-----
Welcome to the Week 7 Lab!
-----
Welcome to the Week 7 Lab!
-----
Welcome to the Week 7 Lab!
-----
Pressione qualquer tecla para continuar. . .
```

Figure 259: Repeated Welcome Output.

7.3.9 Exercise 9

This program has a function `print_colour` that takes an integer as argument and returns either the color of light or if the light is invisible to the human eye, according to this table:

Colour:		Wavelength Interval:
Red		~ 700 to 635 nm
Orange		~ 635 to 590 nm
Yellow		~ 590 to 560 nm
Green		~ 560 to 520 nm
Cyan		~ 520 to 490 nm
Blue		~ 490 to 450 nm
Violet		~ 450 to 400 nm

Figure 260: Rainbow Spectrum Table.

The program should output:

```
Enter a wavelength in nanometers: 701
701nm is invisible
```

Figure 261: Rainbow Spectrum Example.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

void print_colour(int nanometers);

int main()
```

Paludo

```
{  
    int nanometers;  
    printf("Enter a wavelength in nanometers: ");  
    scanf("%d", &nanometers);  
  
    print_colour(nanometers);  
  
    printf("\n\n");  
  
    return 0;  
}  
#include <stdio.h>  
  
void print_colour(int nanometers)  
{  
    if (nanometers < 400 || nanometers > 700)  
    {  
        printf("%dnm is invisible", nanometers);  
    }  
    else if (nanometers <= 700 && nanometers >= 635)  
    {  
        printf("%dnm is red", nanometers);  
    }  
    else if (nanometers < 635 && nanometers >= 590)  
    {  
        printf("%dnm is orange", nanometers);  
    }  
    else if (nanometers < 590 && nanometers >= 560)  
    {  
        printf("%dnm is yellow", nanometers);  
    }  
    else if (nanometers < 560 && nanometers >= 520)  
    {  
        printf("%dnm is green", nanometers);  
    }  
    else if (nanometers < 520 && nanometers >= 490)  
    {  
        printf("%dnm is cyan", nanometers);  
    }  
    else if (nanometers < 490 && nanometers >= 450)  
    {  
        printf("%dnm is blue", nanometers);  
    }  
    else if (nanometers < 450 && nanometers >= 400)  
    {  
        printf("%dnm is violet", nanometers);  
    }  
}
```

}

This program outputs:

```
C:\Windows\system32\cmd.exe
Enter a wavelength in nanometers: 598
598nm is orange
Pressione qualquer tecla para continuar. . .
```

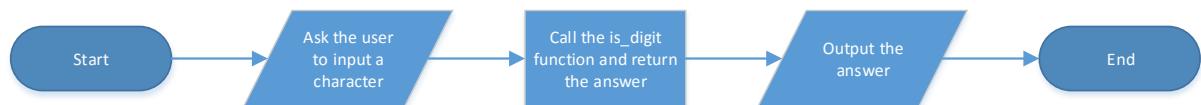
Figure 262: Rainbow Spectrum.

7.3.10 Exercise 10

The `is_digit` function takes a char as input and returns an integer value of 1 if the input, according to the ASCII table, is between 0 and 9 inclusive, otherwise, returns 0.

This function is called in the main using the user input.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

int is_digit(char character);

int main()
{
    char character;
    printf("Enter a character: ");
    scanf("%c", &character);

    int result = is_digit(character);
    printf("The result is: %d", result);

    printf("\n\n");

    return 0;
}
int is_digit(char character)
{
    if (character >= 48 && character <= 57)
    {
        return 1;
    }
    else
  
```

```
{  
    return 0;  
}  
}
```

This program outputs:



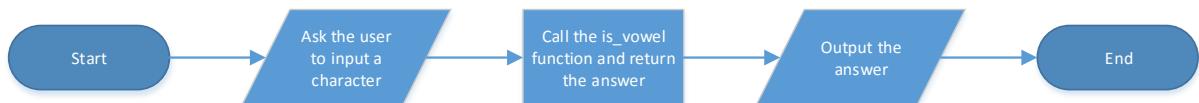
```
C:\Windows\system32\cmd.exe  
Enter a character: 9  
The result is: 1  
Pressione qualquer tecla para continuar. . .
```

Figure 263: Is Digit Output.

7.3.11 Exercise 11

This program has a `is_vowel` function that takes a `char` as input and returns 1 if the input is a vowel and 0 if not. This function is called in the main using the user's input.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int is_vowel(char character);

int main()
{
    char character;
    printf("Enter a character: ");
    scanf("%c", &character);

    int result = is_vowel(character);
    printf("The result is: %d", result);

    printf("\n\n");

    return 0;
}

#include <stdio.h>

int is_vowel(char character)
{
    if (character == 'a' || character == 'e' || character == 'i' ||
        character == 'o' || character == 'u' || character == 'A' ||
        character == 'E' || character == 'I' || character == 'O' ||
        character == 'U')
    {
        return 1;
    }
}
```

```
    }
    else
    {
        return 0;
    }
}
```

This code outputs:

```
C:\Windows\system32\cmd.exe
Enter a character: a
The result is: 1
Pressione qualquer tecla para continuar. . .
```

Figure 264: Is Vowel Output.

7.3.12 Exercise 12

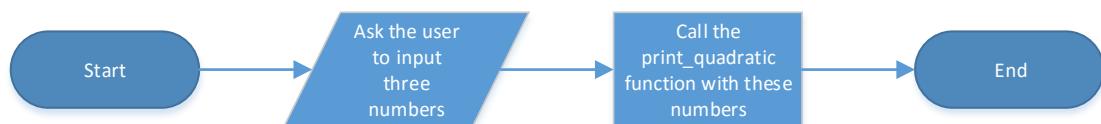
In the main function, the user enters 3 coefficients of a quadratic, then the program calls the print_quadratic function. This function takes the user's input as parameters and returns nothing. The program should output:

```
Enter a: 5
Enter b: 7
Enter c: -2

5x^2 + 7x - 2
```

Figure 264: Print Quadratic.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

void print_quadratic(int a, int b, int c);

int main()
{
    int a;
    printf("Enter a: ");
    scanf("%d", &a);

    int b;
    printf("Enter b: ");
    scanf("%d", &b);

    int c;
```

```
printf("Enter c: ");
scanf("%d", &c);

printf("\n\n");

print_quadratic(a, b, c);

printf("\n\n");

return 0;
}
#include <stdio.h>

void print_quadratic(int a, int b, int c)
{
    printf("%dx^2 + %dx - %d", a, b, c);
}
```

This program outputs:



```
C:\Windows\system32\cmd.exe
Enter a: 3
Enter b: 5
Enter c: 10

3x^2 + 5x - 10

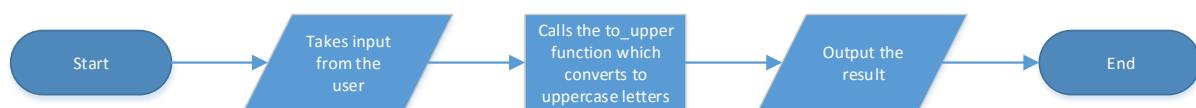
Pressione qualquer tecla para continuar. . .
```

Figure 265: Print Quadratic Output.

7.3.13 Exercise 13

This program converts a letter from lowercase to uppercase using the user's input as parameter for a function.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int to_upper(int character);

int main()
{
    char character;
    printf("Enter a lowercase letter: ");
    scanf("%c", &character);

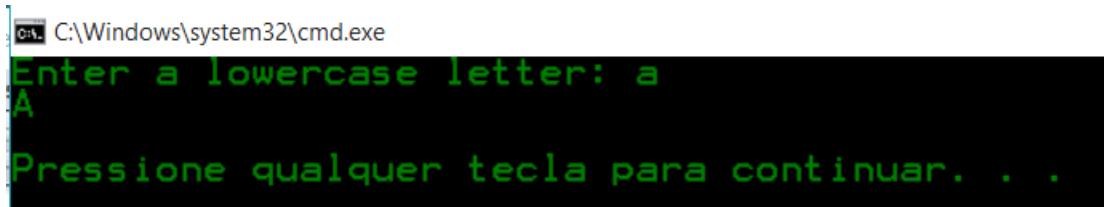
    character = to_upper(character);
    printf("%c", character);

    printf("\n\n");

    return 0;
}
#include <stdio.h>

int to_upper(int character)
{
    return character -= 32;
}
```

This program outputs:



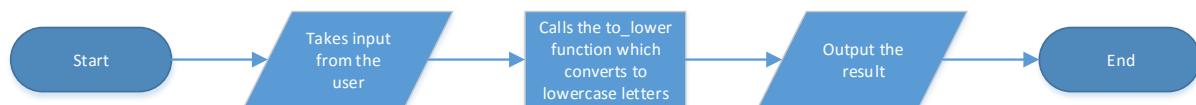
A screenshot of a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
Enter a lowercase letter: a
A
Pressione qualquer tecla para continuar. . .

Figure 266: To Uppercase Output.

7.3.14 Exercise 14

This program converts a letter from uppercase to lowercase using the user's input as parameter for a function.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

int to_lower(int character);

int main()
{
    char character;
    printf("Enter a uppercase letter: ");
    scanf("%c", &character);

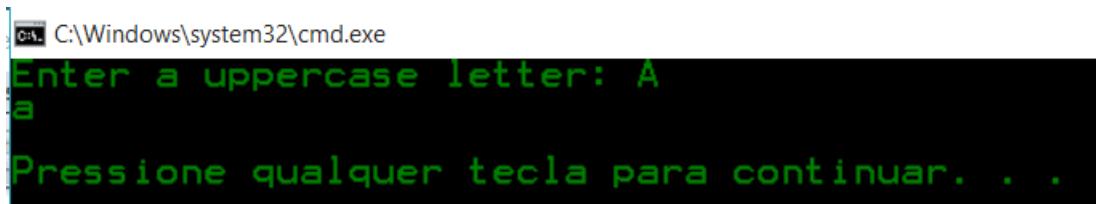
    character = to_lower(character);
    printf("%c", character);

    printf("\n\n");

    return 0;
}
#include <stdio.h>

int to_lower(int character)
{
    return character += 32;
}
  
```

This program outputs:



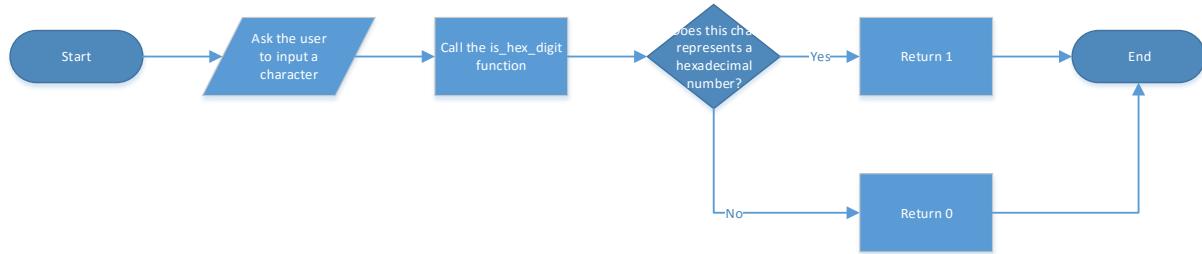
A screenshot of a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
Enter a uppercase letter: A
a
Pressione qualquer tecla para continuar. . .

Figure 267: To Lowercase Output.

7.3.15 Exercise 15

The `is_hex_digit` function takes a `char` as input and returns an integer value of 1 if this symbol represents a hexadecimal digit, otherwise, 0 is returned. This function is called in the `main` with the user's input as parameter.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

int is_hex_digit(int character);

int main()
{
    char character;
    printf("Enter a hexadecimal number: ");
    scanf("%c", &character);

    character = is_hex_digit(character);
    printf("%d", character);

    printf("\n\n");

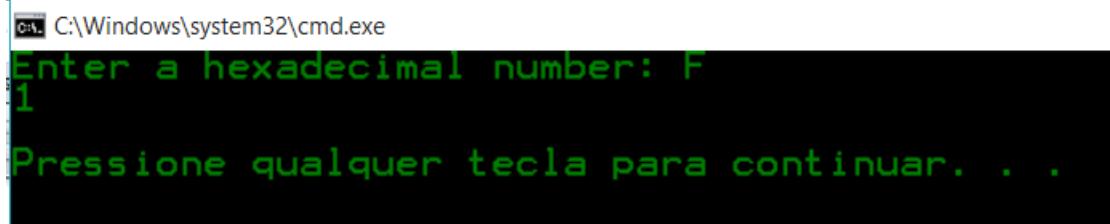
    return 0;
}

#include <stdio.h>

int is_hex_digit(int character)
  
```

```
{  
    if ((character >= 48 && character <= 57) || (character >= 65 &&  
character <= 70))  
    {  
        return 1;  
    }  
    else  
    {  
        return 0;  
    }  
}
```

This program outputs:



The screenshot shows a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
Enter a hexadecimal number: F
1
Pressione qualquer tecla para continuar. . .

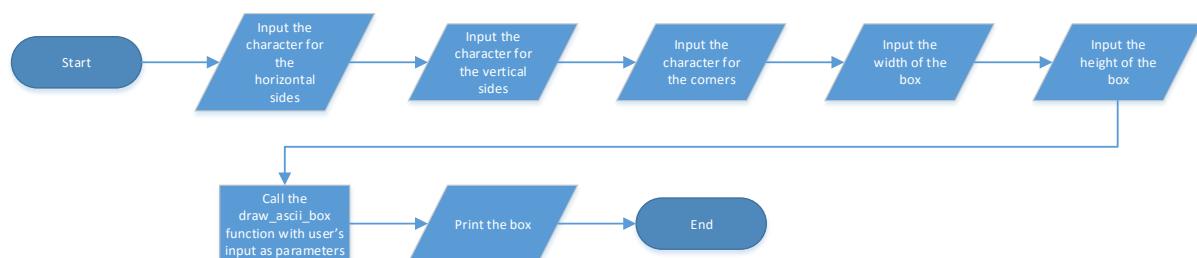
Figure 268: Is Hex Character Output.

7.3.16 Exercise 16

The draw_ascii_box outputs a ACII box with 5 parameters, which are: the character to use for the horizontal sides; the character to user for the vertical sides; the character to user for the corners; the width of the box in characters; and the height. As in the example, the program should output:

```
#=====#
H       H
H       H
H       H
#=====#
```

Figure 269: Draw ASCII Box.



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

void draw_ascii_box(char horizontal, char vertical, char corner, int width, int height);

int main()
{
    char horizontal;
    printf("Horizontal character: ");
    scanf(" %c", &horizontal);

    char vertical;
  
```

```
printf("Vertical character: ");
scanf(" %c", &vertical);

char corner;
printf("Corner character: ");
scanf(" %c", &corner);

int width;
printf("Width: ");
scanf("%d", &width);

int height;
printf("Height: ");
scanf("%d", &height);

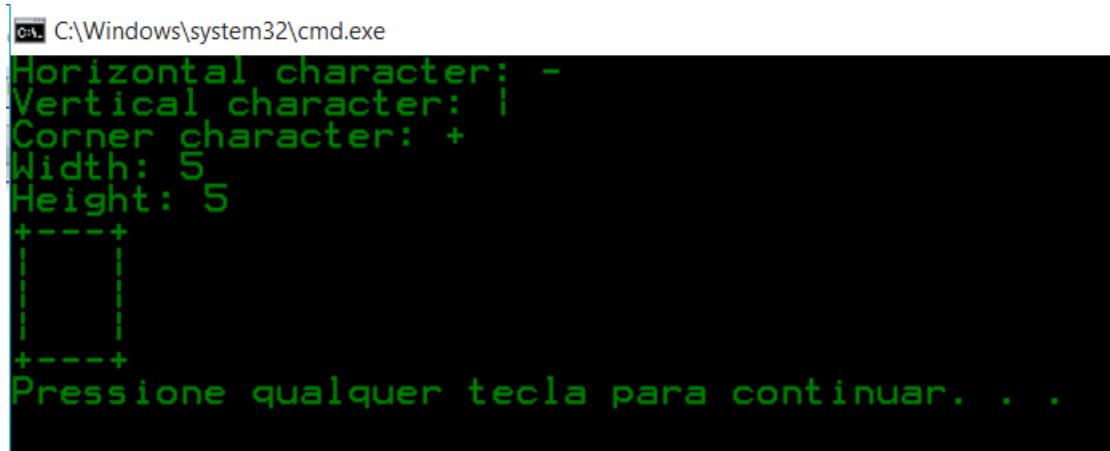
draw_ascii_box(horizontal, vertical, corner, width, height);
}

#include <stdio.h>

void draw_ascii_box(char horizontal, char vertical, char corner, int
width, int height)
{
    for (int a = height; a > 0; a--)
    {
        if (a == height || a == 1)
        {
            for (int b = width; b > 0; b--)
            {
                if (b == width || b == 1)
                {
                    printf("%c", corner);
                }
                else
                {
                    printf("%c", horizontal);
                }
            }
            printf("\n");
        }
        else
        {
            for (int c = width; c > 0; c--)
            {
                if (c == width || c == 1)
                {
                    printf("%c", vertical);
                }
            }
        }
    }
}
```

```
        else
        {
            printf(" ");
        }
    printf("\n");
}
}
```

This program outputs:



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The window displays the following text:
Horizontal character: -
Vertical character: |
Corner character: +
Width: 5
Height: 5
+---+
| |
+---+
Pressione qualquer tecla para continuar. . .

Figure 270: Draw ASCII Box Output.

7.3.17 Exercise 17

The `draw_triangle` function uses integer variables as parameters to output an ASCII triangle with defined height, as in the example:

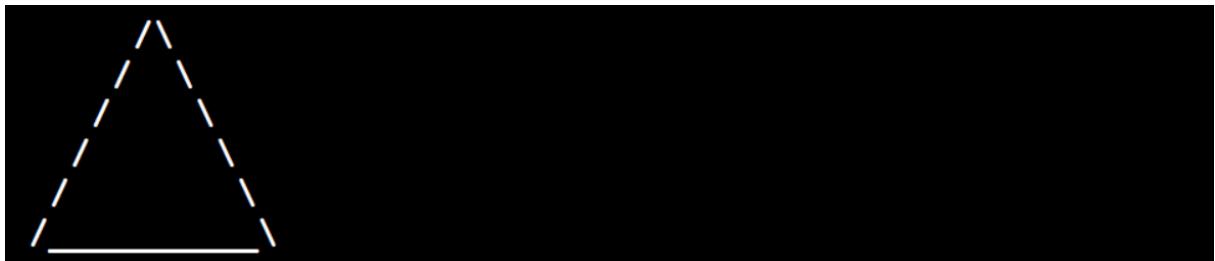
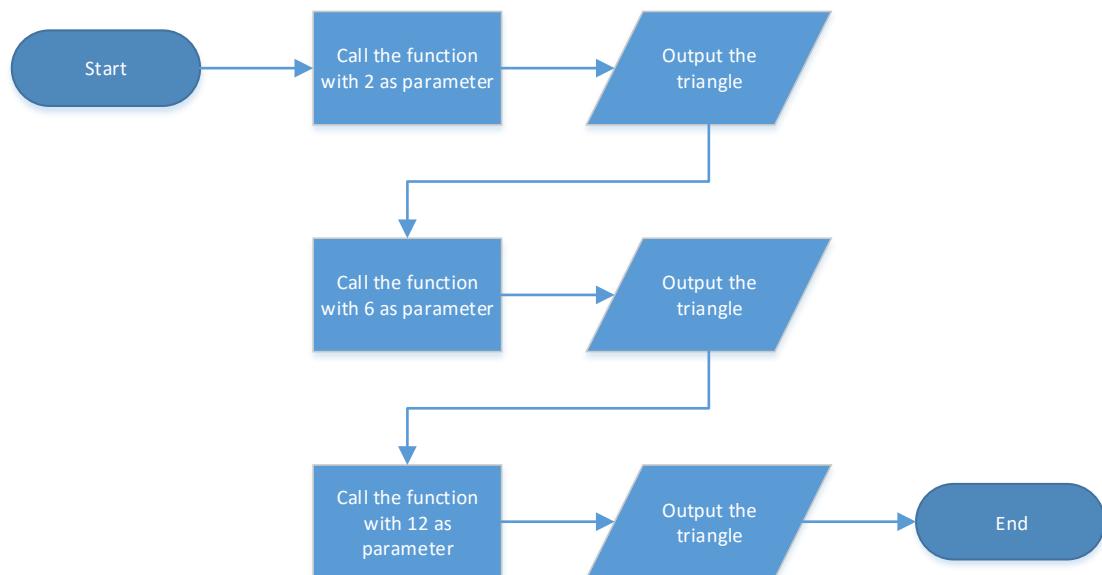


Figure 271: Draw Triangle.

To develop this program, I draw the following flowchart:



Using this flowchart, I started by using last week's triangle code:

```
#include <stdio.h>
```

```
void draw_triangle(int height);

int main()
{
    draw_triangle(2);
    draw_triangle(6);
    draw_triangle(12);

    printf("\n\n");

    return 0;
}
#include <stdio.h>

void draw_triangle(int height)
{
    for (int line = 0; line < height; line++)
    {
        for (int spaces = height - line; spaces > 0; spaces--)
        {
            printf(" ");
        }

        for (int blocks = 0; blocks <= line * 2; blocks++)
        {
            printf("*");
        }

        printf("\n");
    }

    printf("\n\n");
}
```

Then I changed the code a little bit:

```
#include <stdio.h>

void draw_triangle(int height);

int main()
{
    draw_triangle(2);
    draw_triangle(6);
    draw_triangle(12);

    printf("\n\n");
```

```
    return 0;
}

#include <stdio.h>

void draw_triangle(int height)
{
    for (int line = 1; line <= height; line++)
    {
        for (int spaces = height - line; spaces > 0; spaces--)
        {
            printf(" ");
        }

        for (int blocks = 0; blocks <= line * 2; blocks++)
        {
            if (line != height)
            {
                if (blocks == 0)
                {
                    printf("/");
                }
                else if (blocks == line * 2)
                {
                    printf("\\\\");
                }
                else
                {
                    printf(" ");
                }
            }
            if (line != height)
            {
                printf("\n");
            }
        }

        printf("/");
        for (int base = 0; base < height * 2 - 1; base++)
        {
            printf("_");
        }
        printf("\\\\");

        printf("\n\n");
    }
}
```

This program outputs:

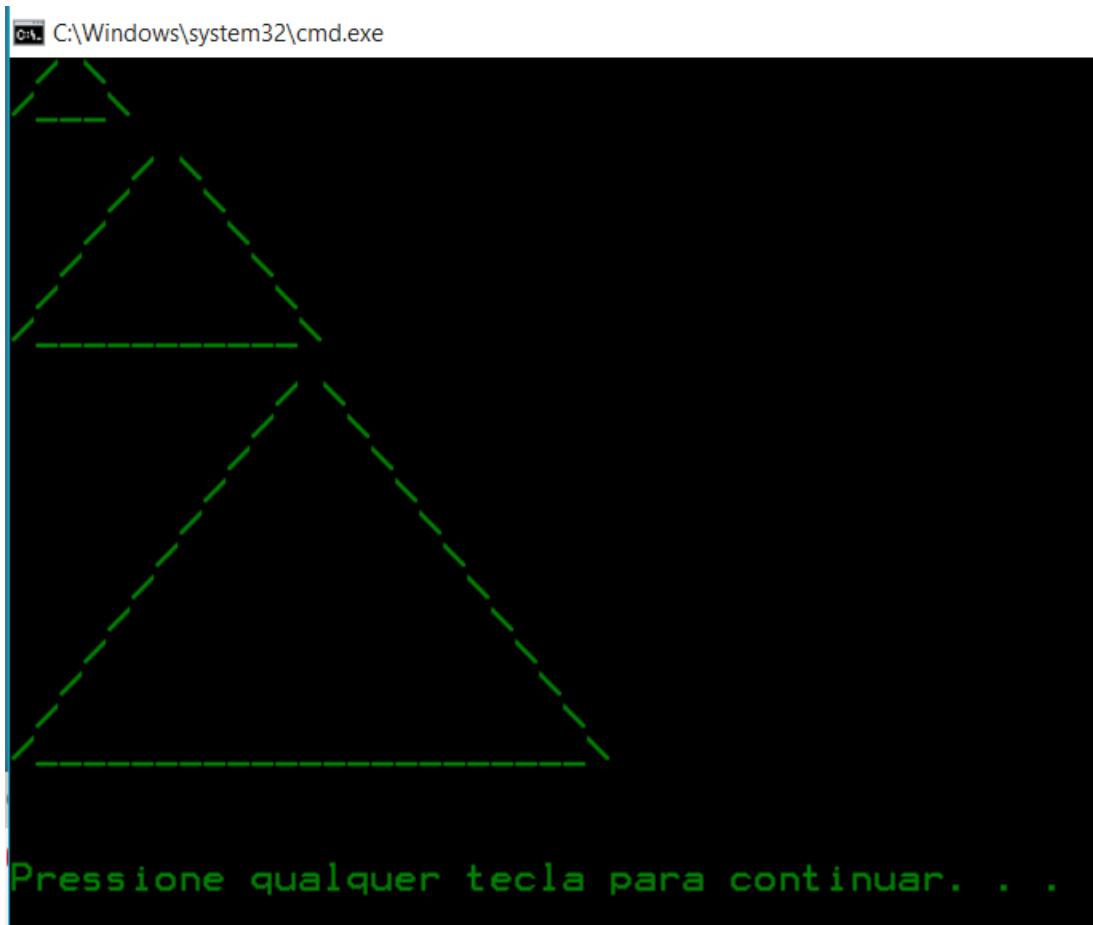


Figure 272: Draw Triangle Output.

7.3.18 Exercise 18

The `draw_inverted_triangle` function uses integer variables as parameters to output an ASCII inverted triangle with defined height, as in the example:

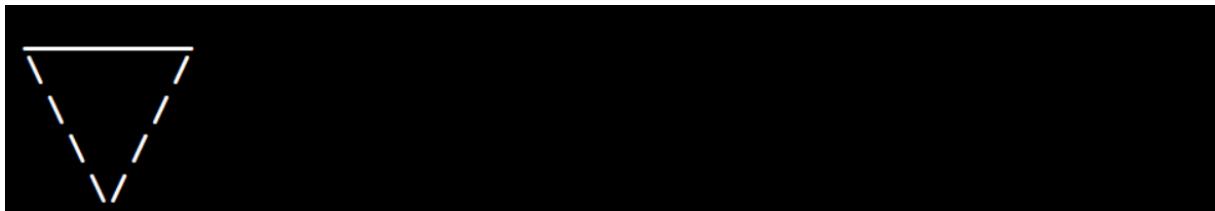
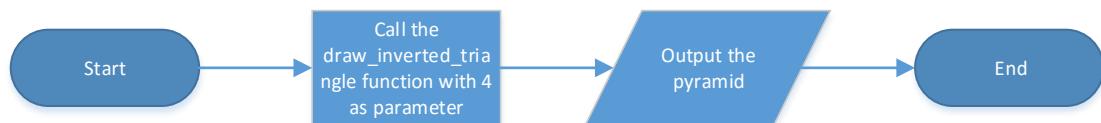


Figure 273: Draw Inverted Triangle.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

void draw_triangle(int height);

int main()
{
    draw_triangle(4);

    printf("\n\n");

    return 0;
}
#include <stdio.h>

void draw_triangle(int height)
{
    for (int base = 0; base < height * 2; base++)
  
```

```
{  
    printf("_");  
}  
  
for (int line = height; line >= 0; line--)  
{  
    for (int spaces = height - line - 1; spaces > 0; spaces--)  
    {  
        printf(" ");  
    }  
  
    for (int blocks = 0; blocks <= line * 2 + 1; blocks++)  
    {  
        if (line != height)  
        {  
            if (blocks == 0)  
            {  
                printf("\\");  
            }  
            else if (blocks == line * 2 + 1)  
            {  
                printf("/");  
            }  
            else  
            {  
                printf(" ");  
            }  
        }  
    }  
    printf("\n");  
}  
  
printf("\n\n");  
}
```

This code outputs:



Figure 274: Draw Inverted Triangle Output.

7.3.19 Exercise 19

This program should refactor a code written previously as example:

```
#include <stdio.h>

int main()
{
    int input = 0;

    do
    {
        printf("Enter a whole number in decimal (0 to stop): ");
        scanf("%d", &input);

        int num_bytes = sizeof(int);
        int num_bits = num_bytes * 8;

        printf("\n%d in base-10 is", input);

        for (int i = num_bits - 1; i >= 0; --i)
        {
            int bit_to_check = (1 << i);

            if (0 == (i + 1) % 8)
            {
                printf(" ");
            }

            if (input & bit_to_check)
            {
                printf("1");
            }
            else
            {
                printf("0");
            }
        }

        printf(" in base-2.\n\n");
    }
    while (0 != input);
```

```
    return 0;
}
```

This code should be restructured to be a program with multiple functions and modular.

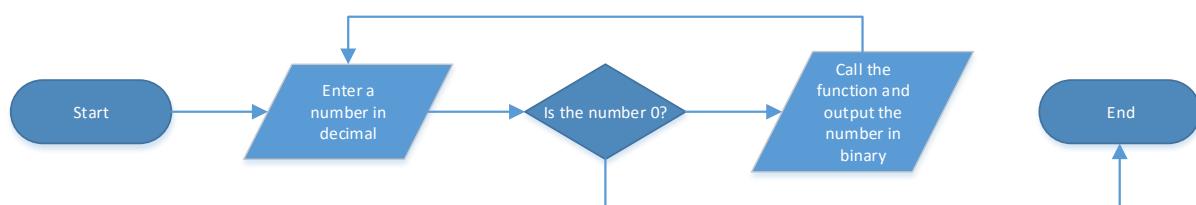
The output can not be changed. The original output is:

```
C:\Windows\system32\cmd.exe
Enter a whole number in decimal (0 to stop): 2335
2335 in base-10 is 00000000 00000000 0001001 00011111 in base-2.
Enter a whole number in decimal (0 to stop): 2321
2321 in base-10 is 00000000 00000000 00001001 00010001 in base-2.
Enter a whole number in decimal (0 to stop): 13
13 in base-10 is 00000000 00000000 00000000 00001101 in base-2.
Enter a whole number in decimal (0 to stop): 0
0 in base-10 is 00000000 00000000 00000000 00000000 in base-2.
Pressione qualquer tecla para continuar. . .
```

Figure 275: Refactor int Bit Pattern.

This program takes a whole number as input from the user and converts from decimal to binary.

To refactor the code, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

int convert_to_binary(int decimal);
```

```
int main()
{
    int decimal;

    do
    {
        printf("Enter a whole number in decimal (0 to stop): ");
        scanf("%d", &decimal);

        convert_to_binary(decimal);
    } while (0 != decimal);

    printf("\n\n");

    return 0;
}
#include <stdio.h>

int convert_to_binary(int decimal)
{
    int num_bytes = sizeof(int);
    int num_bits = num_bytes * 8;

    printf("\n%d in base-10 is", decimal);

    for (int i = num_bits - 1; i >= 0; --i)
    {
        int bit_to_check = (1 << i);

        if (0 == (i + 1) % 8)
        {
            printf(" ");
        }

        if (decimal & bit_to_check)
        {
            printf("1");
        }
        else
        {
            printf("0");
        }
    }

    printf(" in base-2.\n\n");
}
```

This program outputs:

```
C:\Windows\system32\cmd.exe
Enter a whole number in decimal (0 to stop): 658
658 in base-10 is 00000000 00000000 00000010 10010010 in base-2.
Enter a whole number in decimal (0 to stop): 13215
13215 in base-10 is 00000000 00000000 00110011 10011111 in base-2.
Enter a whole number in decimal (0 to stop): 232
232 in base-10 is 00000000 00000000 00000000 11101000 in base-2.
Enter a whole number in decimal (0 to stop): 2
2 in base-10 is 00000000 00000000 00000000 00000010 in base-2.
Enter a whole number in decimal (0 to stop): 0
0 in base-10 is 00000000 00000000 00000000 00000000 in base-2.

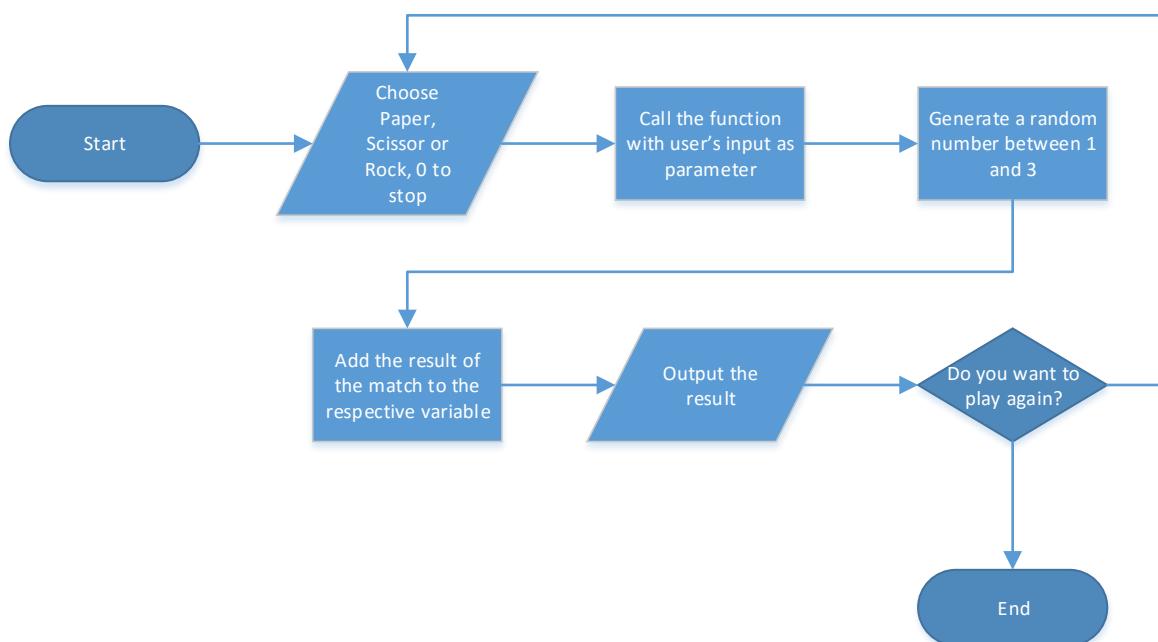
Pressione qualquer tecla para continuar. . .
```

Figure 276: Refactor int Bit Pattern Output.

7.3.20 Exercise 20

This program plays Paper, Scissors, Rock against the user. After each round, the player is able to play again or stop. The details of draws, wins and matches lost are displayed in the end.

To develop this program, I draw the following flowchart:



To develop the program, I used the following previously written code:

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
    srand(time(0));
  
```

```

int dice[5];

// Roll five Yahtzee (Milton Bradley, 1973) dice:
dice[0] = (rand() % 6) + 1;
dice[1] = (rand() % 6) + 1;
dice[2] = (rand() % 6) + 1;
dice[3] = (rand() % 6) + 1;
dice[4] = (rand() % 6) + 1;

// Simple output:
printf("First dice : %d\n", dice[0]);
printf("Second dice : %d\n", dice[1]);
printf("Third dice : %d\n", dice[2]);
printf("Fourth dice : %d\n", dice[3]);
printf("Fifth dice : %d\n", dice[4]);
printf("\n");

// Fancy output:
printf("----+----+----+----+----+\n");
printf("| %d | | %d | ", dice[0], dice[1]);
printf("| %d | | %d | ", dice[2], dice[3]);
printf("| %d |\n", dice[4]);
printf("----+----+----+----+----+\n");

return 0;
}

```

This example rolls 5 Yahtzee random dices and outputs the results in simple and in a fancy outputs:

```

C:\Windows\system32\cmd.exe
First dice : 5
Second dice : 4
Third dice : 2
Fourth dice : 6
Fifth dice : 2

----+----+----+----+----+
| 5 | | 4 | | 2 | | 6 | | 2 |
----+----+----+----+----+
Pressione qualquer tecla para continuar. . .

```

Figure 277: Yahtzee Dice Roller with Array.

This was the written code for the game:

```
#include <stdio.h>

int paper_scissor_rock(int input);

int main()
{
    int input, win, lost, draw;

    printf("Welcome to the Paper, Scissors, Rock game :D");
    printf("\n\n");

    do
    {
        printf("[1] for paper\n");
        printf("[2] for scissor\n");
        printf("[3] for rock\n");
        printf("[0] to stop\n");
        scanf("%d", &input);

        input = paper_scissor_rock(input);

        if (input == 1)
        {
            win++;
        }
        else if (input == 2)
        {
            lost++;
        }
        else
        {
            draw++;
        }
    }
    while (input != 0);

    printf("\n\n");

    if (win > lost)
    {
        printf("Congratulations, you won!");
    }
    else if (lost < win)
    {
```

```

        printf("GAME OVER");
    }
else
{
    printf("Draw");
}

printf("\n\n");

printf("The game had:\n");
printf("WIN: %d\n", win);
printf("LOST: %d\n", lost);
printf("DRAW: %d\n", draw);

printf("\n\n");

return 0;
}
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int paper_scissor_rock(int input)
{
    srand(time(0));

    int machine;

    // Roll five Yahtzee (Milton Bradley, 1973) dice:
    machine = (rand() % 3) + 1;

    // Simple output:
    printf("You choose: %d\n", input);
    printf("The computer choose: %d\n", machine);
    printf("\n");

    // Fancy output:
    printf("----+    ----+\n");
    printf("| %d | VS. | %d |\n", machine, input);
    printf("----+    ----+\n");
    printf("Machine   Human\n");

    if ((input == 1 && machine == 3) || (input == 2 && machine ==
1) || (input == 3 && machine == 2))
    {
        printf("YOU WON!\n");
        return 1;
    }
}

```

```

    }
    else if ((machine == 1 && input == 3) || (machine == 2 && input
== 1) || (machine == 3 && input == 2))
    {
        printf("YOU LOSE!\n");
        return 2;
    }
    else
    {
        printf("DRAW!\n");
        return 3;
    }

    return 0;
}

```

The program was returning an error, debugging it, I found out that the variables for win, lose and draw counting were being used without being initialized.

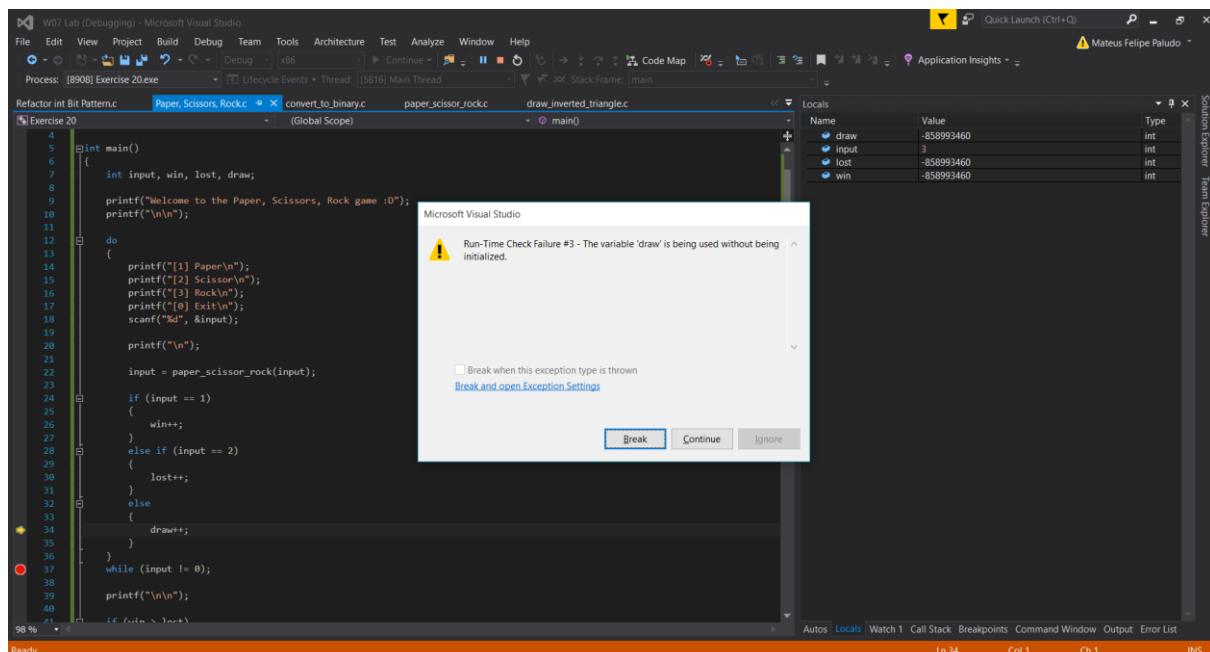


Figure 278: Paper, Scissors, Rock Wrong Output.

This is the fixed code:

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

```
#include <time.h>

int paper_scissor_rock(int input)
{
    srand(time(0));

    int machine;

    // Roll five Yahtzee (Milton Bradley, 1973) dice:
    machine = (rand() % 3) + 1;

    // Simple output:
    printf("You choose: %d\n", input);
    printf("The computer choose: %d\n", machine);
    printf("\n");

    // Fancy output:
    printf(" +---+      +---+\n");
    printf(" | %d | vs. | %d |\n", machine, input);
    printf(" +---+      +---+\n");
    printf("Machine      Human");

    printf("\n\n");

    if ((input == 1 && machine == 3) || (input == 2 && machine ==
1) || (input == 3 && machine == 2))
    {
        printf("YOU WON!");
        printf("\n\n");
        return 1;
    }
    else if ((machine == 1 && input == 3) || (machine == 2 && input
== 1) || (machine == 3 && input == 2))
    {
        printf("YOU LOSE!");
        printf("\n\n");
        return 2;
    }
    else
    {
        printf("DRAW!");
        printf("\n\n");
        return 3;
    }

    return 0;
}
```

```
#include <stdio.h>

int paper_scissor_rock(int input);

int main()
{
    int input;

    int win = 0;
    int lost = 0;
    int draw = 0;

    printf("Welcome to the Paper, Scissors, Rock game :D");
    printf("\n\n");

    do
    {
        printf("[1] Paper\n");
        printf("[2] Scissor\n");
        printf("[3] Rock\n");
        printf("[0] Exit\n");
        scanf("%d", &input);

        printf("\n");

        if (input != 0)
        {
            input = paper_scissor_rock(input);

            if (input == 1)
            {
                win++;
            }
            else if (input == 2)
            {
                lost++;
            }
            else
            {
                draw++;
            }

            printf("-----");
        }
        printf("\n\n");
    }
}
```

```
}

while (input != 0);

printf("-----");
printf("\n\n");

if (win > lost)
{
    printf(" +-----+\n");
    printf(" |YOU WIN!|\n");
    printf(" +-----+\n");
}
else if (lost < win)
{
    printf(" +-----+\n");
    printf(" |GAME OVER!|\n");
    printf(" +-----+\n");
}
else
{
    printf(" +----+\n");
    printf(" |DRAW!|\n");
    printf(" +----+\n");
}

printf("\n\n");

printf("The game had:\n");
printf("WIN: %d\n", win);
printf("LOST: %d\n", lost);
printf("DRAW: %d\n", draw);

printf("\n\n");

return 0;
}
```

The program outputs:

```
C:\Windows\system32\cmd.exe
Welcome to the Paper, Scissors, Rock game :D
[1] Paper
[2] Scissor
[3] Rock
[0] Exit
2

You choose: 2
The computer choose: 1

+---+
| 1 | VS. | 2 |
+---+
Machine      Human

YOU WON!
-----
[1] Paper
[2] Scissor
[3] Rock
[0] Exit
0
-----
+-----+
|YOU WIN!|
+-----+

The game had:
WIN: 1
LOST: 0
DRAW: 0

Pressione qualquer tecla para continuar. . .
```

Figure 279: Paper, Scissors, Rock Output.

But while testing the program, I found a bug in it. The output of the lost screen was wrong:

The screenshot shows a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The text output is as follows:

```
+---+ | 2 | VS. | 1 |
+---+ +---+
Machine      Human

YOU LOSE!

-----
[1] Paper
[2] Scissor
[3] Rock
[0] Exit
0

-----
+----+
|DRAW!|
+----+

The game had:
WIN: 0
LOST: 1
DRAW: 0

Pressione qualquer tecla para continuar. . .
```

Figure 280: Paper, Scissors, Rock Lost Screen Wrong Output.

To fix this problem, I wrote the following code:

```
#include <stdio.h>

int paper_scissor_rock(int input);

int main()
```

```
{  
    int input;  
  
    int win = 0;  
    int lost = 0;  
    int draw = 0;  
  
    printf("Welcome to the Paper, Scissors, Rock game :D");  
    printf("\n\n");  
  
    do  
    {  
        printf("[1] Paper\n");  
        printf("[2] Scissor\n");  
        printf("[3] Rock\n");  
        printf("[0] Exit\n");  
        scanf("%d", &input);  
  
        printf("\n");  
  
        if (input != 0)  
        {  
            input = paper_scissor_rock(input);  
  
            if (input == 1)  
            {  
                win++;  
            }  
            else if (input == 2)  
            {  
                lost++;  
            }  
            else  
            {  
                draw++;  
            }  
  
            printf("-----");  
        }  
    }  
    while (input != 0);  
  
    printf("-----");  
    printf("\n\n");  
}
```

```

if (win > lost)
{
    printf(" +-----+\n");
    printf(" |YOU WIN!|\n");
    printf(" +-----+\n");
}
else if (lost > win)
{
    printf(" +-----+\n");
    printf(" |GAME OVER!|\n");
    printf(" +-----+\n");
}
else
{
    printf(" +----+\n");
    printf(" |DRAW!|\n");
    printf(" +----+\n");
}

printf("\n\n");

printf("The game had:\n");
printf("WIN: %d\n", win);
printf("LOST: %d\n", lost);
printf("DRAW: %d\n", draw);

printf("\n\n");

return 0;
}

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int paper_scissor_rock(int input)
{
    srand(time(0));

    int machine;

    // Roll five Yahtzee (Milton Bradley, 1973) dice:
    machine = (rand() % 3) + 1;

    // Simple output:
    printf("You choose: %d\n", input);
    printf("The computer choose: %d\n", machine);
    printf("\n");
}

```

```
// Fancy output:  
printf(" +---+     +---+\n");  
printf(" | %d | VS. | %d | \n", machine, input);  
printf(" +---+     +---+\n");  
printf("Machine      Human");  
  
printf("\n\n");  
  
if ((input == 1 && machine == 3) || (input == 2 && machine ==  
1) || (input == 3 && machine == 2))  
{  
    printf("YOU WON!");  
    printf("\n\n");  
    return 1;  
}  
else if ((machine == 1 && input == 3) || (machine == 2 && input  
== 1) || (machine == 3 && input == 2))  
{  
    printf("YOU LOSE!");  
    printf("\n\n");  
    return 2;  
}  
else  
{  
    printf("DRAW!");  
    printf("\n\n");  
    return 3;  
}  
  
return 0;  
}
```

This program outputs:

```
C:\Windows\system32\cmd.exe
Welcome to the Paper, Scissors, Rock game :D
[1] Paper
[2] Scissor
[3] Rock
[0] Exit
1

You choose: 1
The computer choose: 2

+---+ +---+
| 2 | VS. | 1 |
+---+ +---+
Machine      Human

YOU LOSE!

-----
[1] Paper
[2] Scissor
[3] Rock
[0] Exit
0

-----
+-----+
|GAME OVER!|
+-----+

The game had:
WIN: 0
LOST: 1
DRAW: 0

Pressione qualquer tecla para continuar. . .
```

Figure 281: Paper, Scissors, Rock Lost Screen Output.

7.3.21 Exercise 21

This program allows two people to play Tic-Tac-Toe and announces the winner. The program cannot make a move in a position that is already taken. The program should output:

```

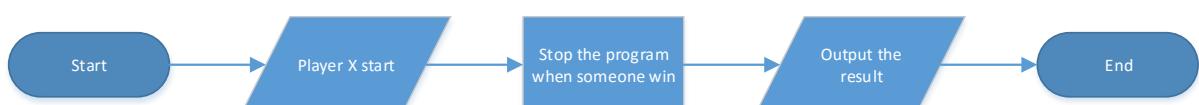
      |   |
  X |   O |   X
      |   |
-----+-----+
      |   |
  X |   O |   X
      |   |
-----+-----+
      |   |
  O |   O |   9
      |   |

Game Over! Player 'O' wins!

```

Figure 282: Tic-Tac-Toe.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

int main()
{
    printf("Welcome to Tic-Tac-Toe!\n");
    printf("-----\n");

    printf("\n\n");

```

```

printf("Player 'X' goes first!");

printf("\n\n");

int input;
int counter = 1;
char position[10] = { '1', '2', '3', '4', '5', '6', '7', '8',
'9' };

do
{
    printf("     |     |      \n");
    printf(" %c | %c | %c  \n", position[0],
position[1], position[2]);
    printf("     |     |      \n");
    printf("-----+-----+-----\n");
    printf("     |     |      \n");
    printf(" %c | %c | %c  \n", position[3],
position[4], position[5]);
    printf("     |     |      \n");
    printf("-----+-----+-----\n");
    printf("     |     |      \n");
    printf(" %c | %c | %c  \n", position[6],
position[7], position[8]);
    printf("     |     |      ");

    printf("\n\n");

    if (counter % 2)
    {
        printf("Where would player 'X' like to go? ");
        scanf("%d", &input);

        while (position[input - 1] == 'O' || position[input
- 1] == 'X')
        {
            printf("Position already taken!\n");
            printf("Where would player 'X' like to go? ");
            scanf("%d", &input);
        }

        position[input - 1] = 'X';
        counter++;
    }
    else

```

```

{
    printf("Where would player 'O' like to go? ");
    scanf("%d", &input);

    while (position[input - 1] == 'O' || position[input
- 1] == 'X')
    {
        printf("Position already taken!\n");
        printf("Where would player 'O' like to go? ");
        scanf("%d", &input);
    }

    position[input - 1] = 'O';

    counter++;
}

printf("\n");
}

while ((position[0] != position[1] || position[1] !=
position[2]) && (position[3] != position[4] || position[4] !=
position[5]) && (position[6] != position[7] || position[7] !=
position[8]) && (position[0] != position[3] || position[3] !=
position[6]) && (position[1] != position[4] || position[4] !=
position[7]) && (position[2] != position[5] || position[5] !=
position[8]) && (position[0] != position[4] || position[4] !=
position[8]) && (position[2] != position[4] || position[4] !=
position[6]) && (counter <= 9));

printf("\n");

printf("     |     |     \n");
printf(" %c | %c | %c \n", position[0], position[1],
position[2]);
printf("     |     |     \n");
printf("-----+-----+-----\n");
printf("     |     |     \n");
printf(" %c | %c | %c \n", position[3], position[4],
position[5]);
printf("     |     |     \n");
printf("-----+-----+-----\n");
printf("     |     |     \n");
printf(" %c | %c | %c \n", position[6], position[7],
position[8]);
printf("     |     |     ");

printf("\n\n");

```

```

        if ((position[0] == position[1] && position[1] == position[2]
&& position[0] == 'X') || (position[3] == position[4] && position[4]
== position[5] && position[3] == 'X') || (position[6] == position[7]
&& position[7] == position[8] && position[6] == 'X') || (position[0]
== position[3] && position[3] == position[6] && position[0] == 'X')
|| (position[1] == position[4] && position[4] == position[7] &&
position[1] == 'X') || (position[2] == position[5] && position[5] ==
position[8] && position[2] == 'X') || (position[0] == position[4] &&
position[4] == position[8] && position[0] == 'X') || (position[2] ==
position[4] && position[4] == position[6] && position[2] == 'X'))
{
    printf("Game Over! Player 'X' wins!");
}
else if ((position[0] == position[1] && position[1] ==
position[2] && position[0] == 'O') || (position[3] == position[4] &&
position[4] == position[5] && position[3] == 'O') || (position[6] ==
position[7] && position[7] == position[8] && position[6] == 'O') ||
(position[0] == position[3] && position[3] == position[6] &&
position[0] == 'O') || (position[1] == position[4] && position[4] ==
position[7] && position[1] == 'O') || (position[2] == position[5] &&
position[5] == position[8] && position[2] == 'O') || (position[0] ==
position[4] && position[4] == position[8] && position[0] == 'O') ||

(position[2] == position[4] && position[4] == position[6] &&
position[2] == 'O'))
{
    printf("Game Over! Player 'O' wins!");
}
else
{
    printf("Game Over! Draw!");
}

printf("\n\n");

return 0;
}

```

This program outputs:

```
C:\Windows\system32\cmd.exe
|   |
Where would player 'X' like to go? 2
+---+
| X | X | 0
+---+
| 0 | X | X
+---+
| 7 | 8 | 0
+---+
Where would player '0' like to go? 8
+---+
| X | X | 0
+---+
| 0 | X | X
+---+
| 7 | 0 | 0
+---+
Where would player 'X' like to go? 7
+---+
| X | X | 0
+---+
| 0 | X | X
+---+
| X | 0 | 0
+---+
Game Over! Draw!
Pressione qualquer tecla para continuar. . .
```

Figure 283: Tic-Tac-Toe Output.

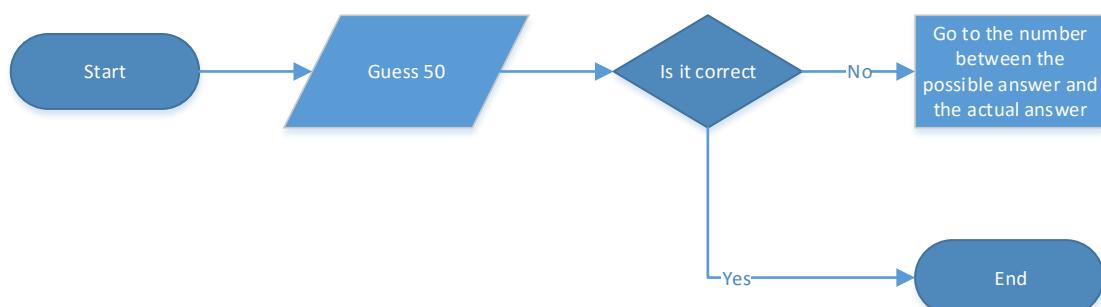
7.3.22 Exercise 22

A user has to think a number between 0 and 100 and the other player has to guess it. The computer will be the player guessing. The program outputs a cheating screen if there are no possible answers left. The program should be able to guess the number or detect cheating in less than 10 guesses. The program should be structured in at least three non-trivial functions and do not use global variables. The program should include comments explaining the guessing algorithm and statin why I believe it will be successful every time. The program should output.

```
Think of a number between 0 and 100, I will guess it!
I guess 50
Am I right (h/l/y)? l
I guess 24
Am I right (h/l/y)? l
I guess 11
Am I right (h/l/y)? l
I guess 5
Am I right (h/l/y)? l
I guess 2
Am I right (h/l/y)? l
I guess 0
Am I right (h/l/y)? l
You are cheating!
```

Figure 284: AI Mind Reader.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```
#include <stdio.h>

void check_cheat(char answer, int guess, int minimum, int maximum);
char scan_char();
int math(int guess, int minimum, int maximum);

int main()
{
    printf("Think of a number between 0 and 100, I will guess
it!\n");

    char answer = 'a';
    int guess = 50;
    int previous = 50;
    int maximum = 100;
    int minimum = 0;

    do
    {
        previous = guess;
        printf("I guess %d", guess);
        printf("\n");
        printf("Am I right (h/l/y)? ");

        answer = scan_char();

        check_cheat(answer, guess, minimum, maximum);

        if (answer == 'l')
        {
            maximum = previous;
            math(guess, minimum, maximum);
        }
        else if (answer == 'h')
        {
            minimum = previous;
            guess = math(guess, minimum, maximum);
        }
    }
    while (answer != 'y');

    printf("\n\n");

    return 0;
}
#include <stdio.h>
```

```
#include <stdlib.h>

void check_cheat(char answer, int guess, int minimum, int maximum)
{
    if ((answer == 'l' && guess == minimum + 1) || (answer == 'h'
&& guess == maximum - 1))
    {
        printf("You are cheating!");
        printf("\n\n");
        exit(1);
    }
}
#include <stdio.h>

char scan_char()
{
    char answer;

    scanf(" %c", &answer);

    return answer;
}
#include <stdio.h>

int math(int guess, int minimum, int maximum)
{
    guess = (maximum - minimum) / 2 + minimum;

    return guess;
}
```

This program outputs:

```
C:\> C:\Windows\system32\cmd.exe
Think of a number between 0 and 100, I will guess it!
I guess 50
Am I right (h/l/y)? h
I guess 75
Am I right (h/l/y)? h
I guess 87
Am I right (h/l/y)? h
I guess 93
Am I right (h/l/y)? h
I guess 96
Am I right (h/l/y)? h
I guess 98
Am I right (h/l/y)? h
I guess 99
Am I right (h/l/y)? h
You are cheating!

Pressione qualquer tecla para continuar. . .
```

Figure 285: AI Mind Reader Output.

8 Eight

8.1 Lectures

The first slide of the classes this week demonstrated how to convert a number from base-16 to base-10 in a very simple way.

This week we had an introduction to pointers. To understand how pointers work, we need to know that an address is the location in the memory where the variable's data is stored. This address will be 32-bits (in a 32-bit program) in size, this can be represented either in decimal or in hexadecimal, and each location addresses a single byte. We also saw examples of how to print the address of a variable in our program, during these examples, we understood that the address will be different every time the program runs.

A pointer is also a variable, but pointers stores addresses of other variables. During the classes we saw examples of how to declare, initialize, print, and change pointers.

Wild pointers were also mentioned during classes. These pointers are declared pointers that have not been initialized or do not have a valid value. Dangling pointers are pointers with valid addresses but the address has become invalid.

Using pointers the programmer is able to change the value of a variable in another function, this is pass by reference behaviour. This also allows the programmer to return multiple things from functions. Using this program behaviour the programmer can alter the caller's data using the callee function. To use arrays and pointers together correctly, we learned about pointer arithmetic.

Multi-dimensional arrays were also mentioned during classes. 2D arrays are often useful to store tabular data, such as the Tic-Tac-Toe program board. We also learned how to declare and initialize these arrays.

It is possible to run a program using the command prompt, this was also demonstrated during classes.

8.2 Examples

8.2.1 Example 1

This program introduces a quick example of how addresses work in C programming.

The program displays variables values and its respective addresses. The program outputs:

```
C:\Windows\system32\cmd.exe
whole_number has a value of: 25
whole_number is stored at: 003DF8BC

real_number has a value of: 7.500000
real_number is stored at: 003DF8B0

one_letter has a value of: k
one_letter is stored at: 003DF8A7

phrase has a value of: Week 8 Example
phrase is a C-string: Week 8 Example
phrase is stored at: 003DF88C
&phrase and phrase are the same: 003DF88C

Pressione qualquer tecla para continuar. . .
```

Figure 286: Addresses.

This was the written code:

```
#include <stdio.h>

int main()
{
    int whole_number = 25;
    float real_number = 7.5f;
    char one_letter = 'k';
    char phrase[] = "Week 8 Example";

    printf("whole_number has a value of: %d\n", whole_number);
```

```
printf("whole_number is stored at: %p\n", &whole_number);
printf("\n");

printf("real_number has a value of: %f\n", real_number);
printf("real_number is stored at: %p\n", &real_number);
printf("\n");

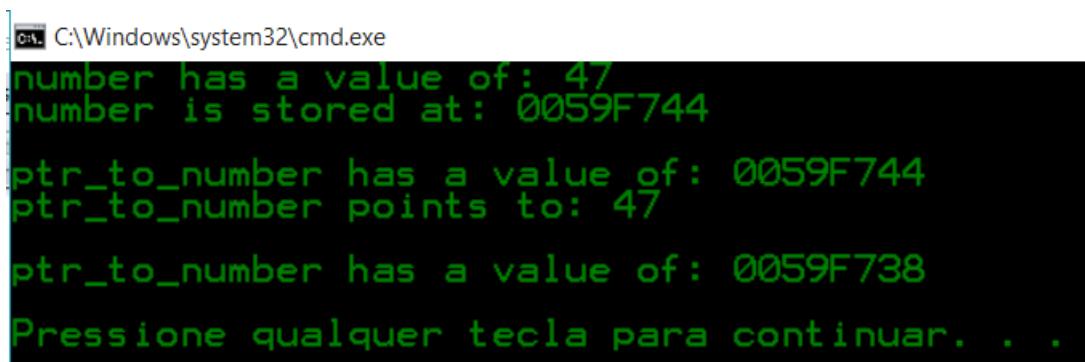
printf("one_letter has a value of: %c\n", one_letter);
printf("one_letter is stored at: %p\n", &one_letter);
printf("\n");

printf("phrase has a value of: %c\n", *phrase);
printf("phrase is a C-string: %s\n", phrase);
printf("phrase is stored at: %p\n", &phrase);
printf("&phrase and phrase are the same: %p\n", phrase);
printf("\n");

return 0;
}
```

8.2.2 Example 2

This program declares and initializes a variable, prints the address and the number that the variable is holding. A pointer to the variable is declared, to declare a point, the type of variable has to be followed by a “*”, and to store a variable’s address in a pointer, the variable’s name has to be preceded by a “&”. To access a variable’s address the “%p” accessor must be used. To access data in memory via a pointer the accessor “%d” must be used. To print an address of a pointer the “%p” must be used as well, to print the value, “%d” must be used. The example outputs:



The screenshot shows a command-line interface window titled "cmd C:\Windows\system32\cmd.exe". The window contains the following text:
number has a value of: 47
number is stored at: 0059F744

ptr_to_number has a value of: 0059F744
ptr_to_number points to: 47

ptr_to_number has a value of: 0059F738

Pressione qualquer tecla para continuar. . .

Figure 287: Simple pointer.

This was the written code:

```
#include <stdio.h>

int main()
{
    // Create a local main variable named 'number'...
    // and store 47 in it:
    int number = 47;

    // Print out the variable's contents,
    // and the address of the variable:
    printf("number has a value of: %d\n", number);
```

```
printf("number is stored at: %p\n", &number);
printf("\n");

// Create a local main variable named ptr_to_number...
// of type int pointer...
// and store the address of the variable 'number' in it:
int* ptr_to_number = &number;

// Print out the ptr_to_number's contents,
// and then print out the data that ptr_to_number points to:
printf("ptr_to_number has a value of: %p\n", ptr_to_number);
printf("ptr_to_number points to: %d\n", *ptr_to_number);
printf("\n");

// Notice that ptr_to_number is stored somewhere too,
// print out the address of ptr_to_number:
printf("ptr_to_number has a value of: %p\n", &ptr_to_number);
printf("\n");

return 0;
}
```

8.2.3 Example 3

This programs outputs a series of examples of numbers holden by variables and its respective addresses, then prints the value of a pointer to the same variable, and the number the pointer points to.

```
C:\Windows\system32\cmd.exe
whole_number has a value of: 152
whole_number is stored at: 001DF82C

ptr_to_whole has a value of: 001DF82C
ptr_to_whole points to: 152

one_letter has a value of: P
one_letter is stored at: 001DF823

ptr_to_letter has a value of: 001DF823
ptr_to_letter points to: P

phrase has a value of: A
phrase is a C-String: Another Week 8 Example
phrase is stored at: 001DF800

ptr_to_phrase has a value of: 001DF800
ptr_to_phrase points to: A
ptr_to_phrase points to: Another Week 8 Example

Pressione qualquer tecla para continuar. . .
```

Figure 288: Pointers and Dereferencing.

This was the written code:

```
#include <stdio.h>

int main()
{
    int whole_number = 152;
    char one_letter = 'P';
    char phrase[] = "Another Week 8 Example";
```

```
int* ptr_to_whole = &whole_number;
char* ptr_to_letter = &one_letter;
char* ptr_to_phrase = phrase;

printf("whole_number has a value of: %d\n", whole_number);
printf("whole_number is stored at: %p\n", &whole_number);
printf("\n");

printf("ptr_to_whole has a value of: %p\n", ptr_to_whole);
printf("ptr_to_whole points to: %d\n", *ptr_to_whole);
printf("\n");

printf("one_letter has a value of: %c\n", one_letter);
printf("one_letter is stored at: %p\n", &one_letter);
printf("\n");

printf("ptr_to_letter has a value of: %p\n", ptr_to_letter);
printf("ptr_to_letter points to: %c\n", *ptr_to_letter);
printf("\n");

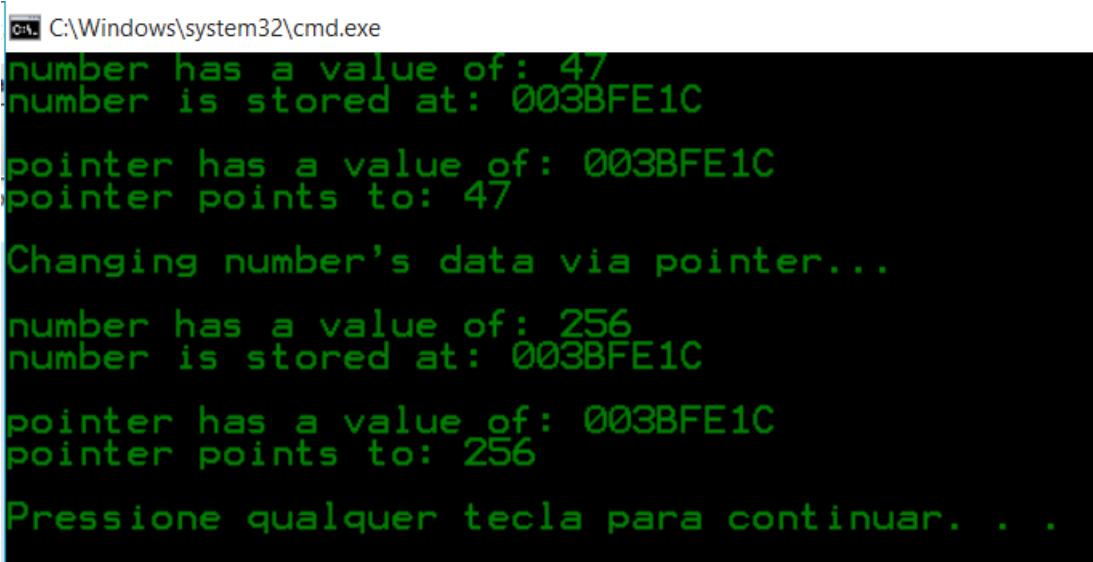
printf("phrase has a value of: %c\n", *phrase);
printf("phrase is a C-String: %s\n", phrase);
printf("phrase is stored at: %p\n", phrase);
printf("\n");

printf("ptr_to_phrase has a value of: %p\n", ptr_to_phrase);
printf("ptr_to_phrase points to: %c\n", *ptr_to_phrase);
printf("ptr_to_phrase points to: %s\n", ptr_to_phrase);
printf("\n");

return 0;
}
```

8.2.4 Example 4

This program declares, initializes, prints the value, address, sets a pointer to a variable, outputs the value of the pointer and the number it points to. To change a number via pointer, the program uses “*” before the pointer variable name. The address is not changed, just the value. The program outputs:



```
C:\Windows\system32\cmd.exe
number has a value of: 47
number is stored at: 003BFE1C

pointer has a value of: 003BFE1C
pointer points to: 47

Changing number's data via pointer...

number has a value of: 256
number is stored at: 003BFE1C

pointer has a value of: 003BFE1C
pointer points to: 256

Pressione qualquer tecla para continuar. . .
```

Figure 289: Changing Data via Dereferencing.

This was the written code:

```
#include <stdio.h>

int main()
{
    int number = 47;

    int* pointer = &number;

    printf("number has a value of: %d\n", number);
    printf("number is stored at: %p\n", &number);
    printf("\n");
```

```
printf("pointer has a value of: %p\n", pointer);
printf("pointer points to: %d\n", *pointer);
printf("\n");

printf("Changing number's data via pointer...\\n\\n");
*pointer = 256;

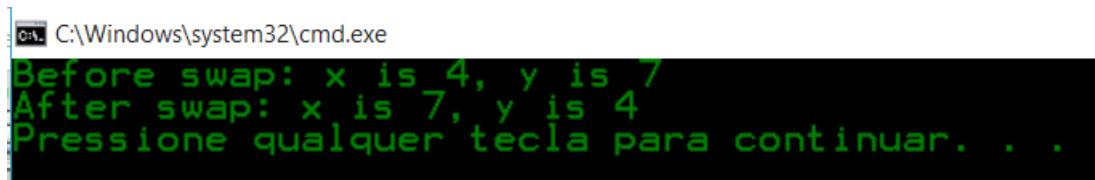
printf("number has a value of: %d\n", number);
printf("number is stored at: %p\n", &number);
printf("\n");

printf("pointer has a value of: %p\n", pointer);
printf("pointer points to: %d\n", *pointer);
printf("\n");

return 0;
}
```

8.2.5 Example 5

This program declares and initiates two variables, calls a function using the variables as input, returns nothing and uses pointers to change the variables values.



```
C:\Windows\system32\cmd.exe
Before swap: x is 4, y is 7
After swap: x is 7, y is 4
Pressione qualquer tecla para continuar. . .
```

Figure 290: Good Swap.

This was the written code:

```
#include <stdio.h>

void swap(int* p_a, int* p_b)
{
    int temp = *p_a;
    *p_a = *p_b;
    *p_b = temp;
}

int main()
{
    int x = 4;
    int y = 7;

    printf("Before swap: x is %d, y is %d\n", x, y);

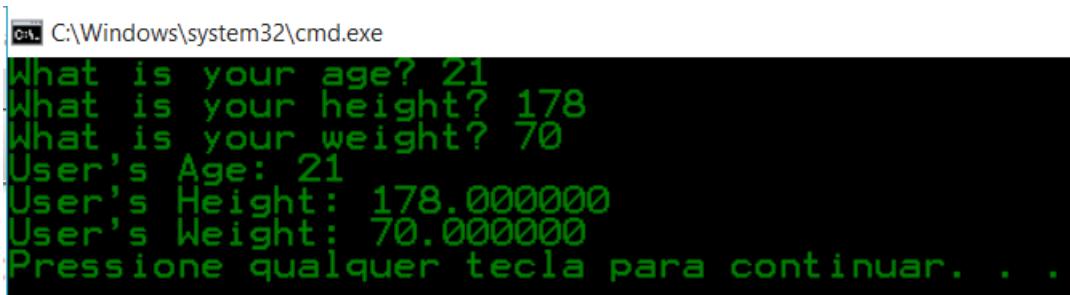
    swap(&x, &y);

    printf("After swap: x is %d, y is %d\n", x, y);

    return 0;
}
```

8.2.6 Example 6

The program takes from the user input from the user for the age, height and weight in a function that returns nothing and outputs the statistics in the main function. The program outputs.



```
C:\Windows\system32\cmd.exe
What is your age? 21
What is your height? 178
What is your weight? 70
User's Age: 21
User's Height: 178.000000
User's Weight: 70.000000
Pressione qualquer tecla para continuar. . .
```

Figure 291: Returning Multiple Things.

This was the written code:

```
#include <stdio.h>

void ask_questions(int* p_age, float* p_height, float* p_weight)
{
    int age = 0;
    float height = 0.0f;
    float weight = 0.0f;

    printf("What is your age? ");
    scanf("%d", &age);

    printf("What is your height? ");
    scanf("%f", &height);

    printf("What is your weight? ");
    scanf("%f", &weight);

    *p_age = age;
    *p_height = height;
    *p_weight = weight;
}
```

```
int main()
{
    int users_age = 0;
    float users_height = 0.0f;
    float users_weight = 0.0f;

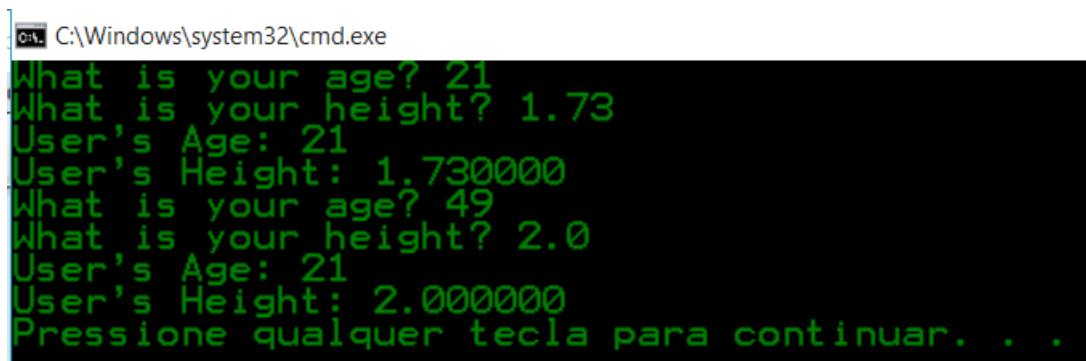
    ask_questions(&users_age, &users_height, &users_weight);

    printf("User's Age: %d\n", users_age);
    printf("User's Height: %f\n", users_height);
    printf("User's Weight: %f\n", users_weight);

    return 0;
}
```

8.2.7 Example 7

This program asks the user to input the age and height two times, but does not returns the age in the second time. The program outputs:



```
C:\Windows\system32\cmd.exe
What is your age? 21
What is your height? 1.73
User's Age: 21
User's Height: 1.730000
What is your age? 49
What is your height? 2.0
User's Age: 21
User's Height: 2.000000
Pressione qualquer tecla para continuar. . .
```

Figure 292: Testing for a Null Pointer.

This was the written code:

```
#include <stdio.h>

void ask_questions(int* p_age, float* p_height)
{
    int age = 0;
    float height = 0.0f;

    printf("What is your age? ");
    scanf("%d", &age);

    printf("What is your height? ");
    scanf("%f", &height);

    if (p_age)
    {
        *p_age = age;
    }

    if (p_height)
    {
        *p_height = height;
    }
}
```

```
        }
    }

int main()
{
    int users_age = 0;
    float users_height = 0.0f;

    ask_questions(&users_age, &users_height);

    printf("User's Age: %d\n", users_age);
    printf("User's Height: %f\n", users_height);

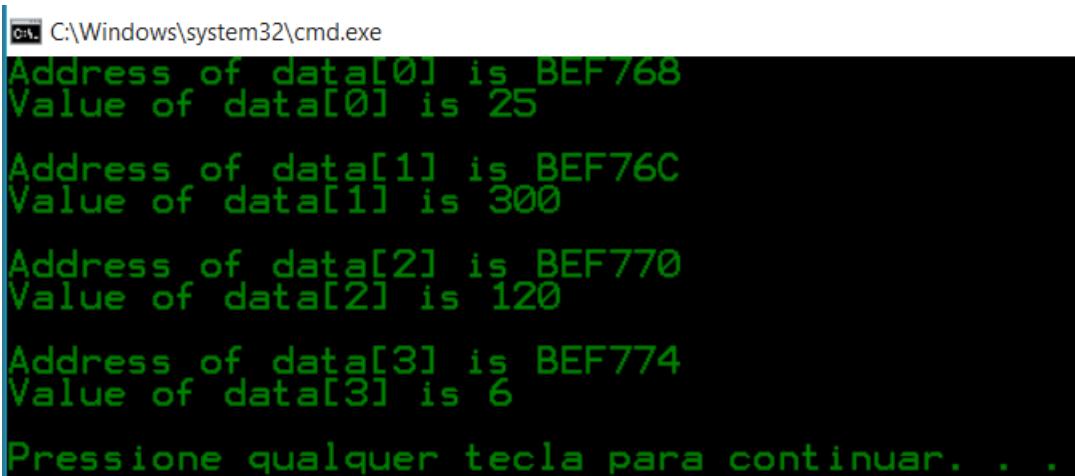
    // This time, do not send back the age...
    ask_questions(0, &users_height);

    printf("User's Age: %d\n", users_age);
    printf("User's Height: %f\n", users_height);

    return 0;
}
```

8.2.8 Example 8

The program declares and initializes an array of integers and uses a for loop to print the address and value of each element. The program outputs:



```
C:\Windows\system32\cmd.exe
Address of data[0] is BEF768
Value of data[0] is 25

Address of data[1] is BEF76C
Value of data[1] is 300

Address of data[2] is BEF770
Value of data[2] is 120

Address of data[3] is BEF774
Value of data[3] is 6

Pressione qualquer tecla para continuar. . .
```

Figure 293: Simple Pointer Arithmetic.

This was the written code:

```
#include <stdio.h>

int main()
{
    int data[] = { 25, 300, 120, 6 };
    int* pointer = 0;

    pointer = data;

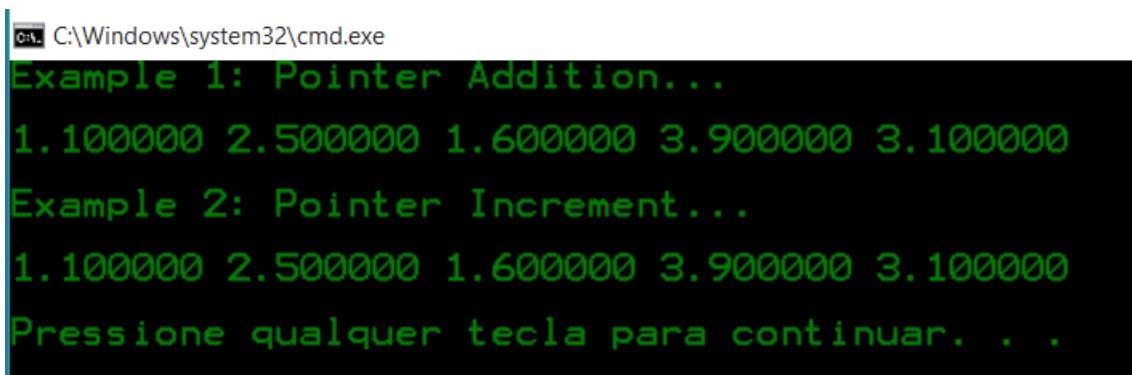
    for (int k = 0; k < 4; ++k)
    {
        printf("Address of data[%d] is %X\n", k, pointer);
        printf("Value of data[%d] is %d\n", k, *pointer);
        printf("\n");

        pointer++;
    }
}
```

```
    return 0;  
}
```

8.2.9 Example 9

The program declares and initializes an array of float elements and outputs the numbers using pointers. The program outputs:



```
C:\Windows\system32\cmd.exe
Example 1: Pointer Addition...
1.100000 2.500000 1.600000 3.900000 3.100000
Example 2: Pointer Increment...
1.100000 2.500000 1.600000 3.900000 3.100000
Pressione qualquer tecla para continuar. . .
```

Figure 294: Pointer Arithmetic.

This was the written code:

```
#include <stdio.h>

int main()
{
    float real_array[] = { 1.1f, 2.5f, 1.6f, 3.9f, 3.1f };

    printf("Example 1: Pointer Addition...\n\n");

    float* ptr_first = real_array;

    for (int k = 0; k < 5; ++k)
    {
        printf("%f ", *(ptr_first + k));
    }

    printf("\n\n");

    printf("Example 2: Pointer Increment...\n\n");

    float* ptr_current = real_array;
```

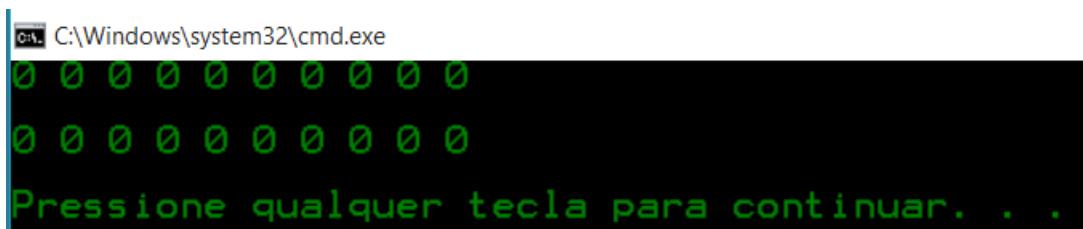
```
for (int k = 0; k < 5; ++k)
{
    printf("%f ", *ptr_current);
    ++ptr_current;
}

printf("\n\n");

return 0;
}
```

8.2.10 Example 10

This program creates an array and copies it to another array using pointers and two methods. The program outputs:



```
C:\Windows\system32\cmd.exe
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
Pressione qualquer tecla para continuar. . .
```

Figure 295: Passing an Array.

The program outputs:

```
#include <stdio.h>

void fill_array(int input[10])
{
    for (int k = 0; k < 10; ++k)
    {
        input[k] = 0;
    }
}

void print_array(int input[10])
{
    for (int k = 0; k < 10; ++k)
    {
        printf("%d ", input[k]);
    }

    printf("\n\n");
}

int main()
{
    int data[10];
    int more_data[10];
```

```
fill_array(data);
fill_array(more_data);

print_array(data);
print_array(more_data);

return 0;
}
```

8.2.11 Example 11

The program creates three int arrays fills the arrays using pointers and outputs the arrays. The program's output is:



```
C:\Windows\system32\cmd.exe
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0
Pressione qualquer tecla para continuar. . .
```

Figure 296: Passing an Array via Pointer.

```
#include <stdio.h>

void fill_array(int* p_first, int num_elements)
{
    for (int k = 0; k < num_elements; ++k)
    {
        p_first[k] = 0;
    }
}

void print_array(int* p_first, int num_elements)
{
    for (int k = 0; k < num_elements; ++k)
    {
        printf("%d ", p_first[k]);
    }

    printf("\n\n");
}

int main()
{
    int data[10];
    int more_data[15];
    int even_more_data[7];
```

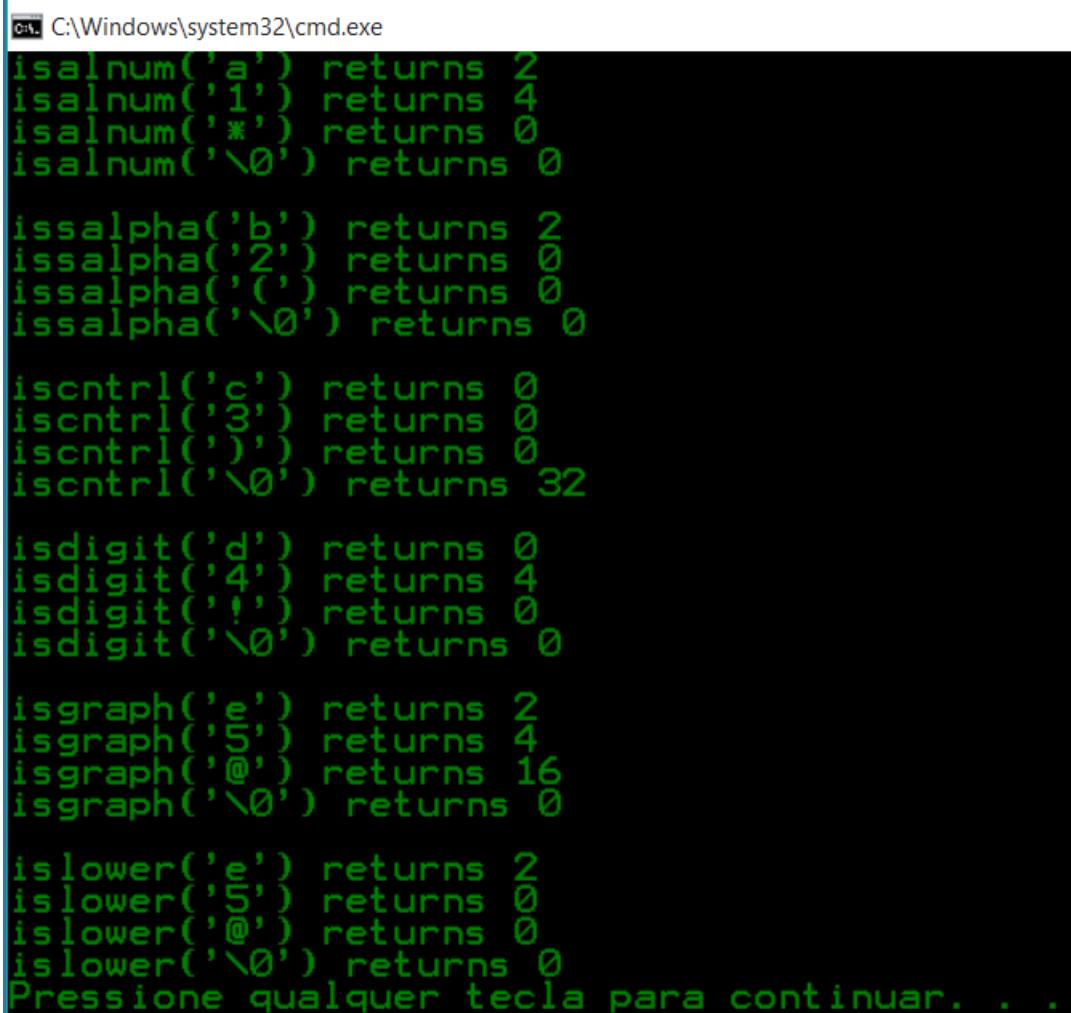
```
fill_array(data, 10);
fill_array(more_data, 15);
fill_array(even_more_data, 7);

print_array(data, 10);
print_array(more_data, 15);
print_array(even_more_data, 7);

return 0;
}
```

8.2.12 Example 12

The program outputs some examples of using the ctype library. The program outputs:



```
C:\Windows\system32\cmd.exe
isalnum('a') returns 2
isalnum('1') returns 4
isalnum('*') returns 0
isalnum('\0') returns 0

issalpha('b') returns 2
issalpha('2') returns 0
issalpha('(') returns 0
issalpha('\0') returns 0

iscntrl('c') returns 0
iscntrl('3') returns 0
iscntrl(')') returns 0
iscntrl('\0') returns 32

isdigit('d') returns 0
isdigit('4') returns 4
isdigit('!') returns 0
isdigit('\0') returns 0

isgraph('e') returns 2
isgraph('5') returns 4
isgraph('@') returns 16
isgraph('\0') returns 0

islower('e') returns 2
islower('5') returns 0
islower('@') returns 0
islower('\0') returns 0
Pressione qualquer tecla para continuar. . .
```

Figure 297: C Type Functions.

```
#include <stdio.h>
#include <ctype.h>

int main()
{
    printf("isalnum('a') returns %d\n", isalnum('a'));
```

```
printf("isalnum('1') returns %d\n", isalnum('1'));
printf("isalnum('*') returns %d\n", isalnum('*'));
printf("isalnum('\\0') returns %d\n", isalnum('\0'));
printf("\n");

printf("issalpha('b') returns %d\n", isalpha('b'));
printf("issalpha('2') returns %d\n", isalpha('2'));
printf("issalpha('(') returns %d\n", isalpha('('));
printf("issalpha('\\0') returns %d\n", isalpha('\0'));
printf("\n");

printf("iscntrl('c') returns %d\n", iscntrl('c'));
printf("iscntrl('3') returns %d\n", iscntrl('3'));
printf("iscntrl(')') returns %d\n", iscntrl(')''));
printf("iscntrl('\\0') returns %d\n", iscntrl('\0'));
printf("\n");

printf("isdigit('d') returns %d\n", isdigit('d'));
printf("isdigit('4') returns %d\n", isdigit('4'));
printf("isdigit('!') returns %d\n", isdigit('!'));
printf("isdigit('\\0') returns %d\n", isdigit('\0'));
printf("\n");

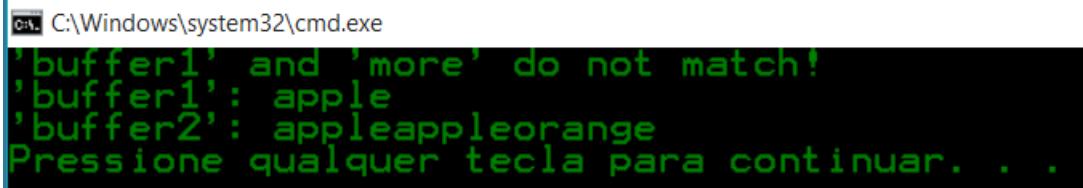
printf("isgraph('e') returns %d\n", isgraph('e'));
printf("isgraph('5') returns %d\n", isgraph('5'));
printf("isgraph('@') returns %d\n", isgraph('@'));
printf("isgraph('\\0') returns %d\n", isgraph('\0'));
printf("\n");

printf("islower('e') returns %d\n", islower('e'));
printf("islower('5') returns %d\n", islower('5'));
printf("islower('@') returns %d\n", islower('@'));
printf("islower('\\0') returns %d\n", islower('\0'));

return 0;
}
```

8.2.13 Example 13

The program does some operations using char arrays. The program outputs:



```
C:\Windows\system32\cmd.exe
'buffer1' and 'more' do not match!
'buffer1': apple
'buffer2': appleappleorange
Pressione qualquer tecla para continuar. . .
```

Figure 298: C-String Functions.

This was the written code:

```
#include <stdio.h>
#include <string.h>

int main()
{
    char text[] = "apple";
    char more[] = "orange";

    char buffer1[20];
    char buffer2[20];

    int length_of_text = strlen(text);
    int length_of_more = strlen(more);

    strcpy(buffer1, text);

    if (strcmp(buffer1, more) == 0)
    {
        printf("'buffer1' and 'more' match!\n");
    }
    else
    {
        printf("'buffer1' and 'more' do not match!\n");
    }

    strcpy(buffer2, buffer1);
    strcat(buffer2, text);
    strcat(buffer2, more);

    printf("'buffer1': %s\n", buffer1);
```

```
printf("'buffer2': %s\n", buffer2);

return 0;
}
```

8.3 Exercises

8.3.1 Exercise 1

1. What is an address?

An address is a numerical (usually decimal or hexadecimal) of a address in the computer's memory in which the variable is stored on.

2. How do you get the address of a variable?

To get the address of a variable the & command must be used before the name of a variable.

3. What is a pointer?

A pointer is a variable that stores the representation of an address.

4. How do you declare a pointer?

To declare a pointer the type of variable must be followed by a *.

5. How do you initialize a pointer?

To declare a pointer the name of the variable the pointer is going to refer to must be preceded by an &.

6. What is dereferencing?

Dereferencing is to use a pointer to take the value that the address the pointer points to is holding.

7. What is a null pointer?

A null pointer is a pointer that holds the value of zero.

8. How do you get access to the data that a pointer points to?

To get access to the data that a pointer points to the proper caller must be used, and a * must be used before the name of the variable.

9. How do you find the address of the first element in an array?

To find the address of the first element in an array the array must be in parentheses adding the position of the element (zero, in this case), preceded by a * outside of the parentheses .

10. How do you find the address of the last element in an array?

To find the address of the last element in an array the quantity of elements in the array must be added to the parentheses .

11. Write a loop to iterate through an array from the first element to the last element, using pointers.

```
#include <stdio.h>

void iterate_first_to_last();
void iterate_last_to_first();

int main()
{
    iterate_first_to_last();
    iterate_last_to_first();
    printf("\n\n");
    return 0;
}

#include <stdio.h>

void iterate_first_to_last()
{
    int data[] = { 25, 300, 120, 6 };
```

```

int* pointer = 0;

pointer = data;

for (int k = 0; k < 4; ++k)
{
    printf("Address of data[%d] is %X\n", k, pointer);
    printf("Value of data[%d] is %d\n", k, *pointer);
    printf("\n");

    pointer++;
}
}

```

12. Write a loop to iterate thought an array from the last element to the first element, using pointers.

```

#include <stdio.h>

void iterate_first_to_last();

void iterate_last_to_first();

int main()
{
    iterate_first_to_last();

    iterate_last_to_first();

    printf("\n\n");

    return 0;
}

#include <stdio.h>

void iterate_last_to_first()
{
    int data[] = { 25, 300, 120, 6 };
    int* pointer = 0;

    pointer = data;

    for (int k = 4; k > 0; ++k)
    {
        printf("Address of data[%d] is %X\n", k, pointer);
        printf("Value of data[%d] is %d\n", k, *pointer);
    }
}

```

```
    printf("\n");
    pointer++;
}
}
```

13. What are the advantages to using pointers?

The advantages of using pointers is that you can return more than one variable from a method.

14. What are the disadvantages to using pointers?

One of the disadvantages of using pointers is that it can cause some bugs if not used properly.

8.3.2 Exercise 2

This program should follow these orders:

- Declare a `main` function which returns zero.
- Call `printf` with the output message: "`Week8: Introduction to Pointers\n`"
- Declare an integer variable called `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`:
 - Call `printf` once to display the contents of `my_age`.
 - Call `printf` once to display the contents of `my_pointer`.
 - 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`.
 - Dereference the pointer, and then assign to the value-pointed-to the value of zero.
- Using `printf`:
 - Call `printf` once to display the contents of `my_age`.
 - Call `printf` once to display the contents of `my_pointer`.
 - Call `printf` once to display the value pointed to by `my_pointer`

The output should be:

```
Week8: Introduction to Pointers
my_age holds the value _____
my_pointer holds the value _____
my_pointer points to the value _____
Indirection test!
my_age holds the value _____
my_pointer holds the value _____
my_pointer points to the value _____
```

Figure 299: Introduction to pointers.

This was the written code:

```
#include <stdio.h>

int main()
{
    printf("Week8: Introduction to Pointers\n");

    int my_age;
    my_age = 21;

    int* my_pointer;
    my_pointer = &my_age;

    printf("my_age holds the value %d\n", my_age);
    printf("my_pointer holds the value %p\n", my_pointer);
    printf("my_pointer points to the value %d\n", *my_pointer);

    printf("Indirection test!\n");

    *my_pointer = 0;

    printf("my_age holds the value %d\n", my_age);
    printf("my_pointer holds the value %p\n", my_pointer);
    printf("my_pointer points to the value %d\n", *my_pointer);

    printf("\n\n");

    return 0;
}
```

The program outputs:

```
C:\Windows\system32\cmd.exe
Week8: Introduction to Pointers
my_age holds the value 21
my_pointer holds the value 0044F76C
my_pointer points to the value 21
Indirection test!
my_age holds the value 0
my_pointer holds the value 0044F76C
my_pointer points to the value 0

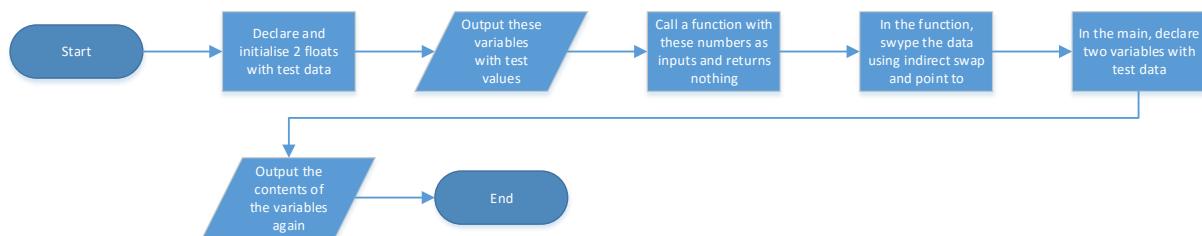
Pressione qualquer tecla para continuar. . .
```

Figure 300: Introduction to Pointers.

8.3.3 Exercise 3

This program is a prototype for a function named exchange, which takes two pointers to real numbers. The function must use indirection to swap the data and point to. In the main, declare two real variables, and set them to appropriate test values, then print out the contents of the variables again.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

void exchange(float* px, float* py);

int main()
{
    float x = 1;
    float y = 2;

    printf("x = %f\n", x);
    printf("y = %f\n", y);

    printf("\n");
    printf("Swybe numbers:");

    exchange(&x, &y);

    printf("\n");
}

```

```
printf("x = %f\n", x);
printf("y = %f\n", y);

printf("\n");

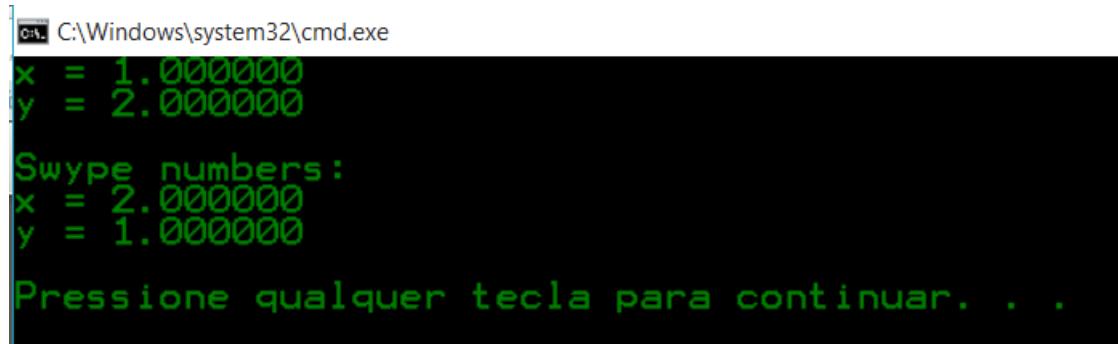
return 0;
}
#include <stdio.h>

void exchange(float* px, float* py)
{
    int temporary = *px;

    *px = *py;
    *py = temporary;
}
```

To understand properly, I ran the program using the debug mode.

The program displays:



A screenshot of a Windows command prompt window titled 'cmd'. The window shows the following text output:

```
C:\Windows\system32\cmd.exe
x = 1.000000
y = 2.000000
Swype numbers:
x = 2.000000
y = 1.000000
Pressione qualquer tecla para continuar. . .
```

Figure 301: float Exchange Output.

8.3.4 Exercise 4

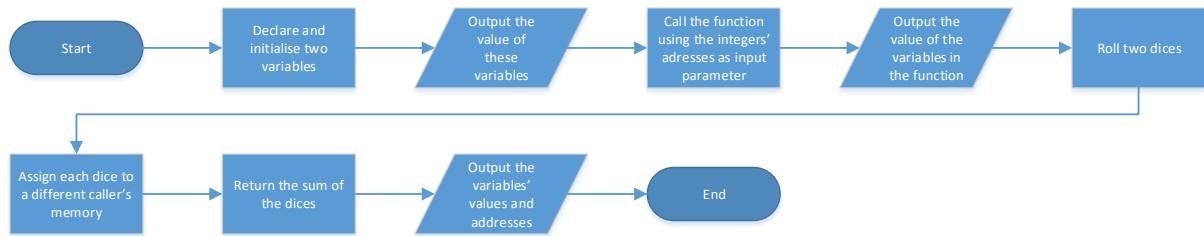
This program has a function which rolls two dices, and returns the value of the sum of the face values of the two dices. Using pointers, and pass-by-reference behaviour, the function must take two integer pointers as parameters, each parameter representing a dice. In the main function, call the function with the address of two declared and properly initialized integer variables as parameters. A third variable in main outputs the variable returned by the function.

The program should output:

```
main: Week8: Pass by Reference, with Pointers
main: Starting main function:
main: variable dice1 holds the value: 0
main: variable dice1 stored at: ????????
main: variable dice1 holds the value: 0
main: variable dice2 stored at: ????????
main: calling: roll_dice(???????, ????????
roll_dice: Starting roll_dice function!
roll_dice: variable address1 holds the value: ????????
roll_dice: variable address2 holds the value: ????????
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: ?
main: variable dice1 stored at: ????????
main: variable dice1 holds the value: ?
main: variable dice2 stored at: ????????
main: variable total_roll holds the value: ?
main: Returning zero to the operating system...
```

Figure 302: Pass by Reference, with Pointers.

To develop this program, I draw the following flowchart:



This was the written code:

```

#include <stdio.h>

int roll_dice(int* address1, int* address2);

int main()
{
    int dice1 = 1;
    int dice2 = 2;

    int total_roll = roll_dice(&dice1, &dice2);

    printf("\n\n");

    return 0;
}

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int roll_dice(int* address1, int* address2)
{
    srand(time(0));

    *address1 = (rand() % 6) + 1;
    *address2 = (rand() % 6) + 1;

    int sum = *address1 + *address2;

    return sum;
}
    
```

To have the same output, the code was changed to:

```

#include <stdio.h>

int roll_dice(int* address1, int* address2);
    
```

```
int main()
{
    printf("main: Week8: Pass by Reference, with pointers\n");
    printf("main: Starting main function:\n");

    int dice1 = 0;
    printf("main: variable dice1 holds the value: %d\n", dice1);
    printf("main: variable dice1 stored at: %d\n", &dice1);

    int dice2 = 0;
    printf("main: variable dice2 holds the value: %d\n", dice2);
    printf("main: variable dice2 stored at: %d\n", &dice2);

    printf("main: calling: roll_dice(%d, %d)\n", &dice1, &dice2);
    int total_roll = roll_dice(&dice1, &dice2);

    printf("main: variable dice1 holds the value: %d\n", dice1);
    printf("main: variable dice1 stored at: %d\n", &dice1);

    printf("main: variable dice2 holds the value: %d\n", dice2);
    printf("main: variable dice2 stored at: %d\n", &dice2);

    printf("main: variable total_roll holds the value: %d\n",
total_roll);
    printf("main: variable total_roll stored at: %d\n",
&total_roll);

    printf("main: Returning zero to the operating system...\n");

    printf("\n\n");

    return 0;
}

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int roll_dice(int* address1, int* address2)
{
    printf("roll_dice: Starting roll_dice function!\n");

    printf("roll_dice: variable address1 holds the value: %d\n",
address1);
    printf("roll_dice: variable address2 holds the value: %d\n",
address2);
```

```

printf("roll_dice: ROLLING TWO DICE!\n");
srand(time(0));

printf("roll_dice: Assigning first dice to caller's
memory...\n");
*address1 = (rand() % 6) + 1;

printf("roll_dice: Assigning second dice to caller's
memory...\n");
*address2 = (rand() % 6) + 1;

int sum = *address1 + *address2;

printf("roll_dice: Returning sum of two dice...\\n");
return sum;
}

```

This program outputs:

```

C:\Windows\system32\cmd.exe
main: Week8: Pass by Reference, with pointers
main: Starting main function:
main: variable dice1 holds the value: 0
main: variable dice1 stored at: 8780732
main: variable dice2 holds the value: 0
main: variable dice2 stored at: 8780720
main: calling: roll_dice(8780732, 8780720)
roll_dice: Starting roll_dice function!
roll_dice: variable address1 holds the value: 8780732
roll_dice: variable address2 holds the value: 8780720
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: 4
main: variable dice1 stored at: 8780732
main: variable dice2 holds the value: 1
main: variable dice2 stored at: 8780720
main: variable total_roll holds the value: 5
main: variable total_roll stored at: 8780708
main: Returning zero to the operating system...

Pressione qualquer tecla para continuar. . .

```

Figure 303: Pass by Reference, with Pointers Output.

8.3.5 Exercise 5

Given the following code:

```
#include <stdio.h>

// Insert your code here!

int main()
{
    int data_array_1 = { 1, 3, 5, 7, 9, 11 };
    int data_array_2 = { 2, -4, 6, -8, 10, -12, 14, -16 };
    int data_array_3 = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0 };

    int result_1 = count_even(data_array_1, 6);

    printf("data_array_1 has %d even numbers.\n", result_1);

    int result_2 = count_even(data_array_2, 8);

    printf("data_array_2 has %d even numbers.\n", result_2);

    int result_3 = count_even(data_array_3, 11);

    printf("data_array_3 has %d even numbers.\n", result_3);

    return 0;
}
```

A function which takes an integer array via pointer, and the array's size in an integer should be written. The function return the number of even numbers in the array. Other two array declarations and initializations should be added in the main function, the other function must be called with these arrays, and then print the result.

This was the written code:

```
#include <stdio.h>

int count_even(int* data_array, int size);
```

```
int main()
{
    int data_array_1 = { 1, 3, 5, 7, 9, 11 };
    int data_array_2 = { 2, -4, 6, -8, 10, -12, 14, -16 };
    int data_array_3 = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0 };

    int result_1 = count_even(data_array_1, 6);

    printf("data_array_1 has %d even numbers.\n", result_1);

    int result_2 = count_even(data_array_2, 8);

    printf("data_array_2 has %d even numbers.\n", result_2);

    int result_3 = count_even(data_array_3, 11);

    printf("data_array_3 has %d even numbers.\n", result_3);

    return 0;
}
#include <stdio.h>

int count_even(int* data_array, int size)
{
    int count_even = 0;

    for (int i = 0; i < size; i++)
    {
        if (data_array[i] % 2 == 1)
        {
            count_even++;
        }
    }

    return count_even;
}
```

But this code will not compile because it has some bugs in its code:

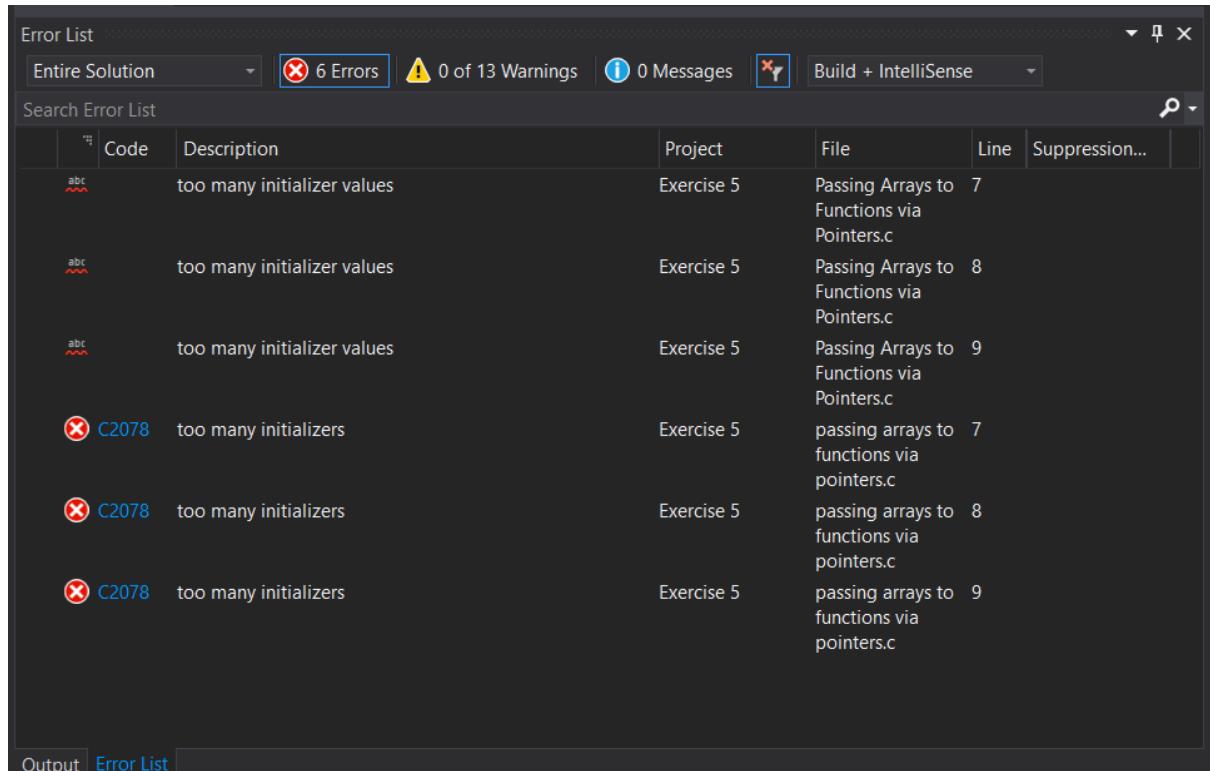


Figure 304: Errors reported.

To fix these errors, I wrote the following code:

```
#include <stdio.h>

int count_even(int* data_array, int size);

int main()
{
    int data_array_1[] = { 1, 3, 5, 7, 9, 11 };
    int data_array_2[] = { 2, -4, 6, -8, 10, -12, 14, -16 };
    int data_array_3[] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0 };

    int result_1 = count_even(data_array_1, 6);

    printf("data_array_1 has %d even numbers.\n", result_1);

    int result_2 = count_even(data_array_2, 8);

    printf("data_array_2 has %d even numbers.\n", result_2);
```

```
int result_3 = count_even(data_array_3, 11);

printf("data_array_3 has %d even numbers.\n", result_3);

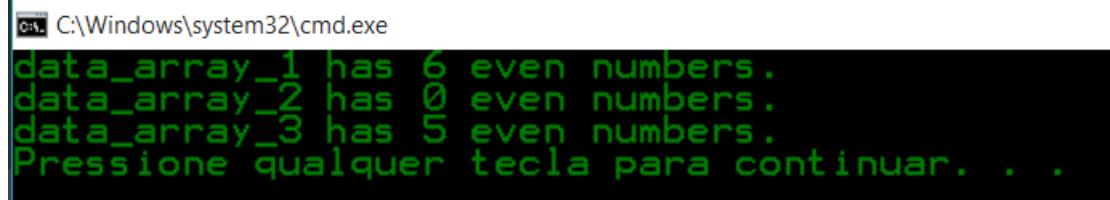
return 0;
}
#include <stdio.h>

int count_even(int* data_array, int size)
{
    int count_even = 0;

    for (int i = 0; i < size; i++)
    {
        if (data_array[i] % 2 == 1)
        {
            count_even++;
        }
    }

    return count_even;
}
```

This program outputs:



C:\Windows\system32\cmd.exe

```
data_array_1 has 6 even numbers.
data_array_2 has 0 even numbers.
data_array_3 has 5 even numbers.
Pressione qualquer tecla para continuar. . .
```

Figure 305: Passing Arrays to Functions via Pointer Output.

8.3.6 Exercise 6

Given the following code:

```
void zero_out_array(int* p_array, int num_elements)
{
    printf("zero_out_array called:\n");
    // Insert code here...
}

void print_array(int* p_array, int num_elements)
{
    printf("print_array called:\n");
    // Insert code here...
}

int main()
{
    int main_array[] = { 15, 24, 33, 42, 51 };
    // Insert code here...
    return 0;
}
```

In the main function, the print_array is called passing in the main_array pointer, an appropriate value for num_elements is added. Then, the zero_out_array function is called with the main_array and appropriate parameters. Then, print_array is called with main_array again.

The functions print_array and zero_out_array requires array iteration. The [] bracket array index access notation is used in one of these functions, and the dereference pointer arithmetic notation in the other.

In print_array, the code will iterate though the p_array, printing each element.

In zero_out_array, p_array will be iterated, setting each element to zero.

This was the written code:

```
#include<stdio.h>

void zero_out_array(int* p_array, int num_elements);

void print_array(int* p_array, int num_elements);

int main()
{
    int main_array[] = { 15, 24, 33, 42, 51 };

    print_array(*main_array, 5);

    zero_out_array(main_array, 5);

    print_array(*main_array, 5);

    return 0;
}
#include <stdio.h>

void print_array(int* p_array, int num_elements)
{
    printf("print_array called:\n");

    for (int i = 0; i < num_elements; i++)
    {
        printf("%d ", p_array[i]);
    }
}
#include <stdio.h>

void zero_out_array(int* p_array, int num_elements)
{
    printf("zero_out_array called:\n");

    for (int i = 0; i < num_elements; i++)
    {
        p_array[i] = 0;
    }
}
```

But this code returns an error, to fix the problem, I wrote the following code:

```
#include<stdio.h>
```

```
void zero_out_array(int* p_array, int num_elements);

void print_array(int* p_array, int num_elements);

int main()
{
    int main_array[] = { 15, 24, 33, 42, 51 };

    print_array(main_array, 5);

    zero_out_array(main_array, 5);

    print_array(main_array, 5);

    printf("\n\n");

    return 0;
}
#include <stdio.h>

void print_array(int* p_array, int num_elements)
{
    printf("print_array called:\n");

    for (int i = 0; i < num_elements; i++)
    {
        printf("%d ", p_array[i]);
    }

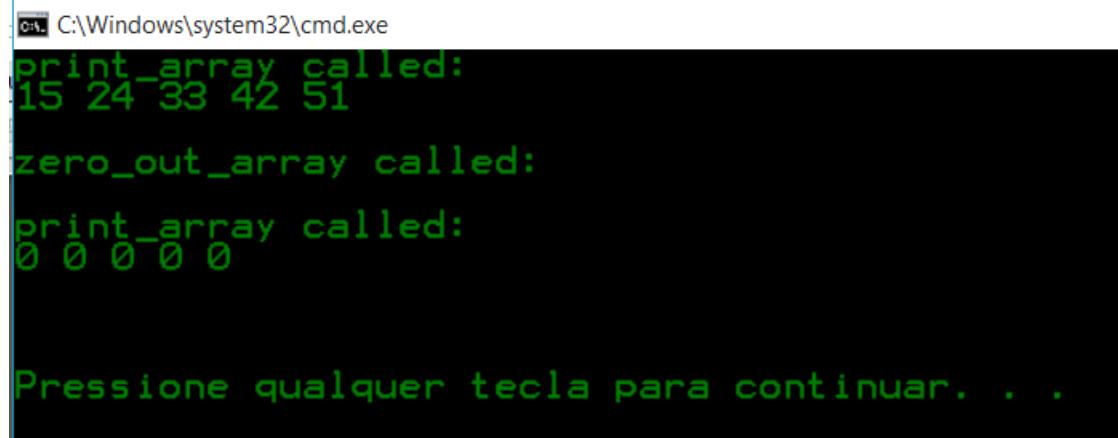
    printf("\n\n");
}
#include <stdio.h>

void zero_out_array(int* p_array, int num_elements)
{
    printf("zero_out_array called:\n");

    for (int i = 0; i < num_elements; i++)
    {
        p_array[i] = 0;
    }

    printf("\n");
}
```

This program outputs:



A screenshot of a Windows command prompt window titled 'cmd C:\Windows\system32\cmd.exe'. The window displays the following text:

```
print_array called:  
15 24 33 42 51  
  
zero_out_array called:  
  
print_array called:  
0 0 0 0 0  
  
Pressione qualquer tecla para continuar. . .
```

Figure 306: Memory Sharing with Pointers Output.

8.3.7 Exercise 7

Given the following code:

```
#include <stdio.h>

int main()
{
    char buffer1[] = "Hello Programming 1 Students";
    char buffer2[] = "Learn to program using arrays and pointers!";

    int upper_count = 0;
    int lower_count = 0;
    int digit_count = 0;

    count_categories(buffer1, &upper_count, &lower_count,
&digit_count);

    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer1,
           upper_count, lower_count, digit_count);

    count_categories(buffer2, &upper_count, &lower_count,
&digit_count);

    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer2,
           upper_count, lower_count, digit_count);

    return 0;
}
#include <stdio.h>

void count_categories(char* p_cstring, int* p_upper_count,
                      int* p_lower_count, int* p_digit_count)
{
    // Insert your code here...
}
```

The program should implement the `count_categories` function to return three pieces of information based upon the `p_cstring` passed into the function. Count the number of upper case, lower case and digits in the `p_cstring`, and return this information via pointer. Add another two array declarations and initialisations in the `main` function, and call `count_categories` with the newly added arrays, then print the results.

I also added another function to set the counters to zero.

This was the written code:

```
#include <stdio.h>

void set_zero(int* p_upper_count, int* p_lower_count, int*
p_digit_count);
void count_categories(char* p_cstring, int* p_upper_count, int*
p_lower_count, int* p_digit_count);

int main()
{
    char buffer1[] = "Hello Programming 1 Students";
    char buffer2[] = "Learn to program using arrays and pointers!";
    char buffer3[] = "If debugging is the process of removing
software bugs, then programming must be the process of putting them
in.";
    char buffer4[] = "\"I am error.\"—random fat dude in \"The Legend
of Zelda 2\"";

    int upper_count, lower_count, digit_count;

    set_zero(upper_count, lower_count, digit_count);
    count_categories(buffer1, &upper_count, &lower_count,
&digit_count);
    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer1,
upper_count, lower_count, digit_count);

    set_zero(upper_count, lower_count, digit_count);
    count_categories(buffer2, &upper_count, &lower_count,
&digit_count);
    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer2,
upper_count, lower_count, digit_count);

    set_zero(upper_count, lower_count, digit_count);
    count_categories(buffer3, &upper_count, &lower_count,
&digit_count);
    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer3,
upper_count, lower_count, digit_count);

    set_zero(upper_count, lower_count, digit_count);
    count_categories(buffer4, &upper_count, &lower_count,
&digit_count);
    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer4,
upper_count, lower_count, digit_count);

    return 0;
}
```

```
#include <stdio.h>

void set_zero(int* p_upper_count, int* p_lower_count, int*
p_digit_count)
{
    p_upper_count = 0;
    p_lower_count = 0;
    p_digit_count = 0;
}
#include <stdio.h>
#include <string.h>

void count_categories(char* p_cstring, int* p_upper_count, int*
p_lower_count, int* p_digit_count)
{
    for (int i = 0; i < strlen(p_cstring); i++)
    {
        if (p_cstring[i] >= 65 && p_cstring[i] <= 90)
        {
            p_upper_count++;
        }
        else if (p_cstring[i] >= 97 && p_cstring[i] <= 122)
        {
            p_lower_count++;
        }
        else
        {
            p_digit_count++;
        }
    }
}
```

But this code returns an error. Debugging the code, I figured out that the variables were being used without being initialized. To fix it, I wrote the following code:

```
#include <stdio.h>

void count_categories(char* p_cstring, int* p_upper_count, int*
p_lower_count, int* p_digit_count);

int main()
{
    char buffer1[] = "Hello Programming 1 Students";
    char buffer2[] = "Learn to program using arrays and pointers!";
```

```

        char buffer3[] = "If debugging is the process of removing
software bugs, then programming must be the process of putting them
in.";
        char buffer4[] = "\"I am error.\"—random fat dude in \"The Legend
of Zelda 2\"";

        int upper_count = 0;
        int lower_count = 0;
        int digit_count = 0;

        count_categories(buffer1, &upper_count, &lower_count,
&digit_count);
        printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer1,
upper_count, lower_count, digit_count);

        count_categories(buffer2, &upper_count, &lower_count,
&digit_count);
        printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer2,
upper_count, lower_count, digit_count);

        count_categories(buffer3, &upper_count, &lower_count,
&digit_count);
        printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer3,
upper_count, lower_count, digit_count);

        count_categories(buffer4, &upper_count, &lower_count,
&digit_count);
        printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer4,
upper_count, lower_count, digit_count);

        return 0;
}
#include <stdio.h>
#include <string.h>

void count_categories(char* p_cstring, int* p_upper_count, int*
p_lower_count, int* p_digit_count)
{
    for (int i = 0; i < strlen(p_cstring); i++)
    {
        if (p_cstring[i] >= 65 && p_cstring[i] <= 90)
        {
            p_upper_count++;
        }
        else if (p_cstring[i] >= 97 && p_cstring[i] <= 122)
        {
            p_lower_count++;
        }
    }
}
```

```

        }
    else
    {
        p_digit_count++;
    }
}
}

```

This program runs, but the output is not as expected:

```
C:\Windows\system32\cmd.exe
[Hello Programming 1 Students] upper: 0, lower: 0, digit: 0
[Learn to program using arrays and pointers!] upper: 0, lower: 0, digit: 0
[If debugging is the process of removing software bugs, then programming must be the process of putting them in.] upper: 0, lower: 0, digit: 0
[I am error.-random fat dude in "The Legend of Zelda 2"] upper: 0, lower: 0, digit: 0
Pressione qualquer tecla para continuar: .
```

Figure 307: Pass by Reference, Multiple Return Values Wrong Output.

Debugging the code I figured out that I was forgetting to put the * before the variable's names. To fix this, I wrote the following code:

```
#include <stdio.h>

void count_categories(char* p_cstring, int* p_upper_count, int*
p_lower_count, int* p_digit_count);

int main()
{
    char buffer1[] = "Hello Programming 1 Students";
    char buffer2[] = "Learn to program using arrays and pointers!";
    char buffer3[] = "If debugging is the process of removing
software bugs, then programming must be the process of putting them
in.";
    char buffer4[] = "\"I am error.\"-random fat dude in \"The Legend
of Zelda 2\"";

    int upper_count = 0;
    int lower_count = 0;
    int digit_count = 0;

    count_categories(buffer1, &upper_count, &lower_count,
&digit_count);
    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer1,
upper_count, lower_count, digit_count);

    count_categories(buffer2, &upper_count, &lower_count,
&digit_count);
    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer2,
upper_count, lower_count, digit_count);
```

```

        count_categories(buffer3, &upper_count, &lower_count,
&digit_count);
        printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer3,
upper_count, lower_count, digit_count);

        count_categories(buffer4, &upper_count, &lower_count,
&digit_count);
        printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer4,
upper_count, lower_count, digit_count);

        printf("\n");

        return 0;
}
#include <stdio.h>
#include <string.h>

void count_categories(char* p_cstring, int* p_upper_count, int*
p_lower_count, int* p_digit_count)
{
    for (int i = 0; i < strlen(p_cstring); i++)
    {
        if (p_cstring[i] >= 65 && p_cstring[i] <= 90)
        {
            *p_upper_count++;
        }
        else if (p_cstring[i] >= 97 && p_cstring[i] <= 122)
        {
            *p_lower_count++;
        }
        else
        {
            *p_digit_count++;
        }
    }
}

```

But the program still outputting the same thing. I realized that the `++` can not be used. To fix this problem, I wrote the following code:

```

#include <stdio.h>

void count_categories(char* p_cstring, int* p_upper_count, int*
p_lower_count, int* p_digit_count);

int main()
{

```

```

char buffer1[] = "Hello Programming 1 Students";
char buffer2[] = "Learn to program using arrays and pointers!";
char buffer3[] = "If debugging is the process of removing
software bugs, then programming must be the process of putting them
in.";
char buffer4[] = "\"I am error.\"—random fat dude in \"The Legend
of Zelda 2\"";

int upper_count = 0;
int lower_count = 0;
int digit_count = 0;

count_categories(buffer1, &upper_count, &lower_count,
&digit_count);
printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer1,
upper_count, lower_count, digit_count);

count_categories(buffer2, &upper_count, &lower_count,
&digit_count);
printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer2,
upper_count, lower_count, digit_count);

count_categories(buffer3, &upper_count, &lower_count,
&digit_count);
printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer3,
upper_count, lower_count, digit_count);

count_categories(buffer4, &upper_count, &lower_count,
&digit_count);
printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer4,
upper_count, lower_count, digit_count);

printf("\n");

return 0;
}

#include <stdio.h>
#include <string.h>

void count_categories(char* p_cstring, int* p_upper_count, int*
p_lower_count, int* p_digit_count)
{
    for (int i = 0; i < strlen(p_cstring); i++)
    {
        if (p_cstring[i] >= 65 && p_cstring[i] <= 90)
        {
            *p_upper_count++;
        }
        else
            *p_lower_count++;
    }
}

```

```

    }
    else if (p_cstring[i] >= 97 && p_cstring <= 122)
    {
        *p_lower_count++;
    }
    else
    {
        *p_digit_count += 1;
    }
}
}
}

```

But the output still not being the expected, all the variables are being counted as digits.

Debugging the program, I figured out that I was missing to put the [i] in some variables inside my “for loop” in the “if statements” in the “count_categories” function. To fix this problem, I wrote the following code:

```

#include <stdio.h>

void count_categories(char* p_cstring, int* p_upper_count, int*
p_lower_count, int* p_digit_count);

int main()
{
    char buffer1[] = "Hello Programming 1 Students";
    char buffer2[] = "Learn to program using arrays and pointers!";
    char buffer3[] = "If debugging is the process of removing
software bugs, then programming must be the process of putting them
in.";
    char buffer4[] = "\"I am error.\"—random fat dude in \"The Legend
of Zelda 2\"";

    int upper_count = 0;
    int lower_count = 0;
    int digit_count = 0;

    count_categories(buffer1, &upper_count, &lower_count,
&digit_count);
    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer1,
upper_count, lower_count, digit_count);

    count_categories(buffer2, &upper_count, &lower_count,
&digit_count);

```

```

    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer2,
upper_count, lower_count, digit_count);

    count_categories(buffer3, &upper_count, &lower_count,
&digit_count);
    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer3,
upper_count, lower_count, digit_count);

    count_categories(buffer4, &upper_count, &lower_count,
&digit_count);
    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer4,
upper_count, lower_count, digit_count);

    printf("\n");

    return 0;
}
#include <stdio.h>
#include <string.h>

void count_categories(char* p_cstring, int* p_upper_count, int*
p_lower_count, int* p_digit_count)
{
    for (int i = 0; i < strlen(p_cstring); i++)
    {
        if (p_cstring[i] >= 65 && p_cstring[i] <= 90)
        {
            *p_upper_count++;
        }
        else if (p_cstring[i] >= 97 && p_cstring[i] <= 122)
        {
            *p_lower_count++;
        }
        else
        {
            *p_digit_count += 1;
        }
    }
}

```

The program still having the same output because I did not change the other variables to be increased with “+= 1”. To fix it, the following code was written:

```
#include <stdio.h>
```

```
void count_categories(char* p_cstring, int* p_upper_count, int*
p_lower_count, int* p_digit_count);

int main()
{
    char buffer1[] = "Hello Programming 1 Students";
    char buffer2[] = "Learn to program using arrays and pointers!";
    char buffer3[] = "If debugging is the process of removing
software bugs, then programming must be the process of putting them
in.";
    char buffer4[] = "\"I am error.\"—random fat dude in \"The Legend
of Zelda 2\";

    int upper_count = 0;
    int lower_count = 0;
    int digit_count = 0;

    count_categories(buffer1, &upper_count, &lower_count,
&digit_count);
    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer1,
upper_count, lower_count, digit_count);

    count_categories(buffer2, &upper_count, &lower_count,
&digit_count);
    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer2,
upper_count, lower_count, digit_count);

    count_categories(buffer3, &upper_count, &lower_count,
&digit_count);
    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer3,
upper_count, lower_count, digit_count);

    count_categories(buffer4, &upper_count, &lower_count,
&digit_count);
    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer4,
upper_count, lower_count, digit_count);

    printf("\n");

    return 0;
}

#include <stdio.h>
#include <string.h>

void count_categories(char* p_cstring, int* p_upper_count, int*
p_lower_count, int* p_digit_count)
```

```

{
    for (int i = 0; i < strlen(p_cstring); i++)
    {
        if (p_cstring[i] >= 65 && p_cstring[i] <= 90)
        {
            *p_upper_count += 1;
        }
        else if (p_cstring[i] >= 97 && p_cstring[i] <= 122)
        {
            *p_lower_count += 1;
        }
        else
        {
            *p_digit_count += 1;
        }
    }
}

```

But the program is always adding to the variables, but it does not set them to zero for each string array:

```

C:\Windows\system32\cmd.exe
Hello Programming 1 Students! upper: 3, lower: 21, digit: 4
Learn to program using arrays and pointers! upper: 4, lower: 56, digit: 11
If debugging is the process of removing software bugs, then programming must be the process of putting them in. upper: 5, lower: 146, digit: 31
I am error.-random fat dude in "The Legend of Zelda 2" upper: 9, lower: 181, digit: 48
Pressione qualquer tecla para continuar...

```

Figure 308: Pass by Reference, Multiple Return Values Wrong Output.

To fix this problem, I wrote the following code:

```

#include <stdio.h>

void count_categories(char* p_cstring, int* p_upper_count, int*
p_lower_count, int* p_digit_count);

int main()
{
    char buffer1[] = "Hello Programming 1 Students";
    char buffer2[] = "Learn to program using arrays and pointers!";
    char buffer3[] = "If debugging is the process of removing
software bugs, then programming must be the process of putting them
in.";
    char buffer4[] = "\"I am error.\"-random fat dude in \"The Legend
of Zelda 2\"";

    int upper_count = 0;

```

```
int lower_count = 0;
int digit_count = 0;

count_categories(buffer1, &upper_count, &lower_count,
&digit_count);
printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer1,
upper_count, lower_count, digit_count);

count_categories(buffer2, &upper_count, &lower_count,
&digit_count);
printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer2,
upper_count, lower_count, digit_count);

count_categories(buffer3, &upper_count, &lower_count,
&digit_count);
printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer3,
upper_count, lower_count, digit_count);

count_categories(buffer4, &upper_count, &lower_count,
&digit_count);
printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer4,
upper_count, lower_count, digit_count);

printf("\n");

return 0;
}

#include <stdio.h>
#include <string.h>

void count_categories(char* p_cstring, int* p_upper_count, int*
p_lower_count, int* p_digit_count)
{
    *p_upper_count = 0;
    *p_lower_count = 0;
    *p_digit_count = 0;

    for (int i = 0; i < strlen(p_cstring); i++)
    {
        if (p_cstring[i] >= 65 && p_cstring[i] <= 90)
        {
            *p_upper_count += 1;
        }
        else if (p_cstring[i] >= 97 && p_cstring[i] <= 122)
        {
            *p_lower_count += 1;
        }
    }
}
```

```

        else
        {
            *p_digit_count += 1;
        }
    }
}

```

This program outputs:

```
C:\Windows\system32\cmd.exe
Hello Programming 1 Students! upper: 3, lower: 21, digit: 4
Learn to program using arrays and pointers! upper: 1, lower: 35, digit: 7
If debugging is the process of removing software bugs, then programming must be the process of putting them in. upper: 1, lower: 35, digit: 20
I am error.'-random fat dude in 'The Legend of Zelda 2' upper: 4, lower: 35, digit: 17
Pressione qualquer tecla para continuar. . .
```

Figure 309: Pass by Reference, Multiple Return Values Wrong Output.

The last string array is not being displayed correctly because of some symbols. To fix it, I wrote the following code:

```
#include <stdio.h>

void count_categories(char* p_cstring, int* p_upper_count, int*
p_lower_count, int* p_digit_count);

int main()
{
    char buffer1[] = "Hello Programming 1 Students";
    char buffer2[] = "Learn to program using arrays and pointers!";
    char buffer3[] = "If debugging is the process of removing
software bugs, then programming must be the process of putting them
in.";
    char buffer4[] = "'I am error.'-random fat dude in 'The Legend
of Zelda 2'";

    int upper_count = 0;
    int lower_count = 0;
    int digit_count = 0;

    count_categories(buffer1, &upper_count, &lower_count,
&digit_count);
    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer1,
upper_count, lower_count, digit_count);

    count_categories(buffer2, &upper_count, &lower_count,
&digit_count);
    count_categories(buffer3, &upper_count, &lower_count,
&digit_count);
    count_categories(buffer4, &upper_count, &lower_count,
&digit_count);
}
```

```

    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer2,
upper_count, lower_count, digit_count);

    count_categories(buffer3, &upper_count, &lower_count,
&digit_count);
    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer3,
upper_count, lower_count, digit_count);

    count_categories(buffer4, &upper_count, &lower_count,
&digit_count);
    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer4,
upper_count, lower_count, digit_count);

    printf("\n");

    return 0;
}
#include <stdio.h>
#include <string.h>

void count_categories(char* p_cstring, int* p_upper_count, int*
p_lower_count, int* p_digit_count)
{
    *p_upper_count = 0;
    *p_lower_count = 0;
    *p_digit_count = 0;

    for (int i = 0; i < strlen(p_cstring); i++)
    {
        if (p_cstring[i] >= 65 && p_cstring[i] <= 90)
        {
            *p_upper_count += 1;
        }
        else if (p_cstring[i] >= 97 && p_cstring[i] <= 122)
        {
            *p_lower_count += 1;
        }
        else
        {
            *p_digit_count += 1;
        }
    }
}

```

I changed the “” for a “”, but it is not displayed correctly as well.

```

C:\Windows\system32\cmd.exe
Hello Programming 1 Students] upper: 3, lower: 21, digit: 4
Learn to program using arrays and pointers!] upper: 1, lower: 35, digit: 7
If debugging is the process of removing software bugs, then programming must be the process of putting them in.] upper: 1, lower: 35, digit: 17
wer: 90, digit: 20
'I am error.-random fat dude in 'The Legend of Zelda 2]' upper: 4, lower: 35, digit: 17
Pressione qualquer tecla para continuar. . .

```

Figure 310: Pass by Reference, Multiple Return Values Wrong Output.

Then I decided to just remove the quotes:

```

#include <stdio.h>

void count_categories(char* p_cstring, int* p_upper_count, int*
p_lower_count, int* p_digit_count);

int main()
{
    char buffer1[] = "Hello Programming 1 Students";
    char buffer2[] = "Learn to program using arrays and pointers!";
    char buffer3[] = "If debugging is the process of removing
software bugs, then programming must be the process of putting them
in.";
    char buffer4[] = "I am error.-random fat dude in The Legend of
Zelda 2";

    int upper_count = 0;
    int lower_count = 0;
    int digit_count = 0;

    count_categories(buffer1, &upper_count, &lower_count,
&digit_count);
    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer1,
upper_count, lower_count, digit_count);

    count_categories(buffer2, &upper_count, &lower_count,
&digit_count);
    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer2,
upper_count, lower_count, digit_count);

    count_categories(buffer3, &upper_count, &lower_count,
&digit_count);
    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer3,
upper_count, lower_count, digit_count);

    count_categories(buffer4, &upper_count, &lower_count,
&digit_count);
    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer4,
upper_count, lower_count, digit_count);

```

```

    printf("\n");

    return 0;
}
#include <stdio.h>
#include <string.h>

void count_categories(char* p_cstring, int* p_upper_count, int*
p_lower_count, int* p_digit_count)
{
    *p_upper_count = 0;
    *p_lower_count = 0;
    *p_digit_count = 0;

    for (int i = 0; i < strlen(p_cstring); i++)
    {
        if (p_cstring[i] >= 65 && p_cstring[i] <= 90)
        {
            *p_upper_count += 1;
        }
        else if (p_cstring[i] >= 97 && p_cstring[i] <= 122)
        {
            *p_lower_count += 1;
        }
        else
        {
            *p_digit_count += 1;
        }
    }
}

```

This program outputs:

```

C:\Windows\system32\cmd.exe
Hello Programming 1 Students! upper: 3, lower: 21, digit: 4
Learn to program using arrays and pointers! upper: 1, lower: 35, digit: 7
If debugging is the process of removing software bugs, then programming must be the process of putting them in. upper: 1, lower: 35, digit: 13
upper: 90, digit: 20
I am error.unrandom fat dude in The Legend of Zelda 2! upper: 4, lower: 35, digit: 13
Pressione qualquer tecla para continuar...

```

Figure 311: Pass by Reference, Multiple Return Values Wrong Output.

I also realized that the program should count digits, and it was counting all symbols that are not uppercase or lowercase letters. To fix this problem, I wrote the following code:

```
#include <stdio.h>
```

```

void count_categories(char* p_cstring, int* p_upper_count, int*
p_lower_count, int* p_digit_count);

int main()
{
    char buffer1[] = "Hello Programming 1 Students";
    char buffer2[] = "Learn to program using arrays and pointers!";
    char buffer3[] = "If debugging is the process of removing
software bugs, then programming must be the process of putting them
in.";
    char buffer4[] = "I am error.-random fat dude in The Legend of
Zelda 2";

    int upper_count = 0;
    int lower_count = 0;
    int digit_count = 0;

    count_categories(buffer1, &upper_count, &lower_count,
&digit_count);
    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer1,
upper_count, lower_count, digit_count);

    count_categories(buffer2, &upper_count, &lower_count,
&digit_count);
    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer2,
upper_count, lower_count, digit_count);

    count_categories(buffer3, &upper_count, &lower_count,
&digit_count);
    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer3,
upper_count, lower_count, digit_count);

    count_categories(buffer4, &upper_count, &lower_count,
&digit_count);
    printf("[%s] upper: %d, lower: %d, digit: %d\n", buffer4,
upper_count, lower_count, digit_count);

    printf("\n");

    return 0;
}
#include <stdio.h>
#include <string.h>

void count_categories(char* p_cstring, int* p_upper_count, int*
p_lower_count, int* p_digit_count)
{

```

```

*p_upper_count = 0;
*p_lower_count = 0;
*p_digit_count = 0;

for (int i = 0; i < strlen(p_cstring); i++)
{
    if (p_cstring[i] >= 65 && p_cstring[i] <= 90)
    {
        *p_upper_count += 1;
    }
    else if (p_cstring[i] >= 97 && p_cstring[i] <= 122)
    {
        *p_lower_count += 1;
    }
    else if (p_cstring[i] >= 48 && p_cstring[i] <= 57)
    {
        *p_digit_count += 1;
    }
}
}

```

This program outputs:

```

C:\Windows\system32\cmd.exe
Hello Programming 1 Students! upper: 3, lower: 21, digit: 1
Learn to program using arrays and pointers! upper: 1, lower: 35, digit: 0
If debugging is the process of removing software bugs, then programming must be the process of putting them in. upper: 1, lower: 35, digit: 0
I am error.unrandom fat dude in The Legend of Zelda 23 upper: 4, lower: 35, digit: 1
Pressione qualquer tecla para continuar. . .

```

Figure 312: Pass by Reference, Multiple Return Values Durty Output.

The program runs correctly, but the output is not good, to make it better, I wrote the following code:

```

#include <stdio.h>

void count_categories(char* p_cstring, int* p_upper_count, int*
p_lower_count, int* p_digit_count);

int main()
{
    char buffer1[] = "Hello Programming 1 Students";
    char buffer2[] = "Learn to program using arrays and pointers!";
    char buffer3[] = "If debugging is the process of removing
software bugs, then programming must be the process of putting them
in.";

```

```
char buffer4[] = "I am error.-random fat dude in The Legend of
Zelda 2";

int upper_count = 0;
int lower_count = 0;
int digit_count = 0;

count_categories(buffer1, &upper_count, &lower_count,
&digit_count);
printf("[%s]\n", buffer1);
printf("upper: %d, lower: %d, digit: %d\n", upper_count,
lower_count, digit_count);
printf("\n");

count_categories(buffer2, &upper_count, &lower_count,
&digit_count);
printf("[%s]\n", buffer2);
printf("upper: %d, lower: %d, digit: %d\n", upper_count,
lower_count, digit_count);
printf("\n");

count_categories(buffer3, &upper_count, &lower_count,
&digit_count);
printf("[%s]\n", buffer3);
printf("upper: %d, lower: %d, digit: %d\n", upper_count,
lower_count, digit_count);
printf("\n");

count_categories(buffer4, &upper_count, &lower_count,
&digit_count);
printf("[%s]\n", buffer4);
printf("upper: %d, lower: %d, digit: %d\n", upper_count,
lower_count, digit_count);
printf("\n");

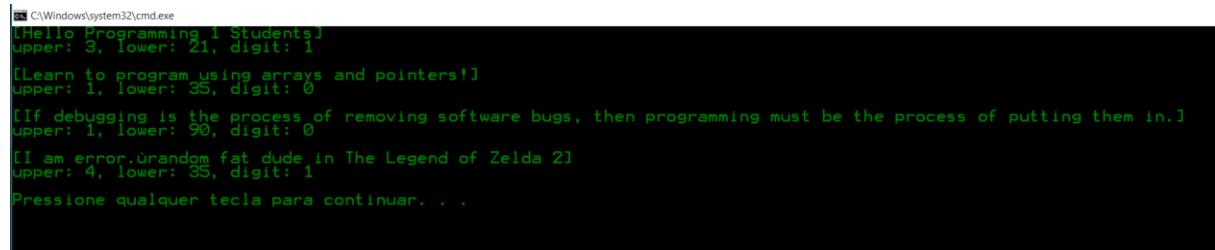
return 0;
}

#include <stdio.h>
#include <string.h>

void count_categories(char* p_cstring, int* p_upper_count, int*
p_lower_count, int* p_digit_count)
{
    *p_upper_count = 0;
    *p_lower_count = 0;
    *p_digit_count = 0;
```

```
for (int i = 0; i < strlen(p_cstring); i++)
{
    if (p_cstring[i] >= 65 && p_cstring[i] <= 90)
    {
        *p_upper_count += 1;
    }
    else if (p_cstring[i] >= 97 && p_cstring[i] <= 122)
    {
        *p_lower_count += 1;
    }
    else if (p_cstring[i] >= 48 && p_cstring[i] <= 57)
    {
        *p_digit_count += 1;
    }
}
}
```

This program outputs:



```
C:\Windows\system32\cmd.exe
[Hello Programming 1 Students]
upper: 3, lower: 21, digit: 1
[Learn to program using arrays and pointers!]
upper: 1, lower: 35, digit: 0
[If debugging is the process of removing software bugs, then programming must be the process of putting them in.]
upper: 1, lower: 90, digit: 0
[I am error.unrandom fat dude in The Legend of Zelda 2]
upper: 4, lower: 35, digit: 1
Pressione qualquer tecla para continuar. . .
```

Figure 313: Pass by Reference, Multiple Return Values Output.

8.3.8 Exercise 8

This program should have a function named palindrome that returns 1 if an input is a palindrome of itself, else, 0 is returned. Three C-String arrays will be created in the main function and called.

I had previously written a code in Java that does almost the same thing:

```
package ThePalindromeClass;

public class PalindromeApp {

    public static void main(String[] args)
    {
        Palindrome p1 = new Palindrome("Abba");
        System.out.println("Palindrome p1 = " + p1);

        Palindrome p2 = new Palindrome("Anne, I vote more cars
race Rome to Vienna");
        System.out.println("Palindrome p2 = " + p2);

        Palindrome p3 = new Palindrome();
        System.out.println("Palindrome p3 = " + p3);

        /*p3.setPalindrome("I'm not a palindrome!");
        System.out.println("Palindrome p3 = " + p3);*/

        /*p3.setPalindrome("If I had a hi-fi...");*
        System.out.println("Palindrome p3 = " + p3);
    }

}

package ThePalindromeClass;

public class Palindrome
{
    String palindrome;
    String teste = "teste";

    public String toString()
```

```
{  
    return  
        this.palindrome;  
}  
  
//-----  
//Constructor  
public Palindrome(String palindrome)  
{  
    this.palindrome = palindrome;  
}  
  
//-----  
//Empty constructor  
public Palindrome()  
{  
}  
  
//-----  
//Reverse  
private String reverse(String input)  
{  
    input = "";  
  
    for(int i = palindrome.length()-1; i >= 0; i--)  
    {  
        input = input + palindrome.charAt(i);  
    }  
  
    return input;  
}  
  
//-----  
//Takes out numbers and spaces  
private String alphaNumeric(String input)  
{  
    input = palindrome;  
    input = input.replaceAll("[^A-Za-z]", "");  
    return input;  
}  
  
//-----  
//Check if is a palindrome  
private boolean isPalindrome(String input)  
{  
    String lcInput = alphaNumeric(input).toLowerCase();  
    return reverse(lcInput).equals(lcInput);  
}
```

```

}

//-----
//Set method
//If it is a palindrome, assings the input to the palindrome
public void setPalindrome(String input)
{
    if(isPalindrome(input) == true)
    {
        this.palindrome = input;
    }
    else
    {
        this.palindrome = " ";
    }
}
}

```

I wrote this code:

```

#include <stdio.h>

int palindrome(char* p_cstring);

int main()
{
    char buffer1[] = "If debugging is the process of removing
software bugs, then programming must be the process of putting them
in.";
    char buffer2[] = "I am error.-random fat dude in The Legend of
Zelda 2";

    int palindrome_return = 0;

    palindrome_return = palindrome(buffer1);
    printf("[%s]\n", buffer1);
    if (palindrome_return == 1)
    {
        printf("Is a palindrome!");
    }
    else
    {
        printf("Is not a palindrome!");
    }
    printf("\n");

    palindrome_return = palindrome(buffer2);
    printf("[%s]\n", buffer2);

```

```

if (palindrome_return == 1)
{
    printf("Is a palindrome!");
}
else
{
    printf("Is not a palindrome!");
}
printf("\n");

printf("\n");

return 0;
}

```

This program does not compile. I decided to research on the internet how to make the program work, and found the following codes (Walia, D. (n. d.)):

```

#include <stdio.h>
#include <string.h>

int main()
{
    char a[100], b[100];

    printf("Enter the string to check if it is a palindrome\n");
    gets(a);

    strcpy(b,a);
    strrev(b);

    if (strcmp(a,b) == 0)
        printf("Entered string is a palindrome.\n");
    else
        printf("Entered string is not a palindrome.\n");

    return 0;
}

#include <stdio.h>

main()
{
    int n, reverse = 0, temp;

    printf("Enter a number to check if it is a palindrome or not\n");
    scanf("%d",&n);

```

```
temp = n;

while (temp != 0)
{
    reverse = reverse * 10;
    reverse = reverse + temp%10;
    temp     = temp/10;
}

if (n == reverse)
    printf("%d is a palindrome number.\n", n);
else
    printf("%d is not a palindrome number.\n", n);

return 0;
}

#include <stdio.h>
#include <string.h>

int main()
{
    char text[100];
    int begin, middle, end, length = 0;

    gets(text);

    while (text[length] != '\0')
        length++;

    end = length - 1;
    middle = length/2;

    for (begin = 0; begin < middle; begin++)
    {
        if (text[begin] != text[end])
        {
            printf("Not a palindrome.\n");
            break;
        }
        end--;
    }
    if (begin == middle)
        printf("Palindrome.\n");

    return 0;
}
```

```
#include <stdio.h>

int is_palindrome(char*);
void copy_string(char*, char*);
void reverse_string(char*);
int string_length(char*);
int compare_string(char*, char*);

int main()
{
    char string[100];
    int result;

    printf("Input a string\n");
    gets(string);

    result = is_palindrome(string);

    if ( result == 1 )
        printf("\"%s\" is a palindrome string.\n", string);
    else
        printf("\"%s\" is not a palindrome string.\n", string);

    return 0;
}

int is_palindrome(char *string)
{
    int check, length;
    char *reverse;

    length = string_length(string);
    reverse = (char*)malloc(length+1);

    copy_string(reverse, string);
    reverse_string(reverse);

    check = compare_string(string, reverse);

    free(reverse);

    if ( check == 0 )
        return 1;
    else
        return 0;
}
```

```
int string_length(char *string)
{
    int length = 0;

    while(*string)
    {
        length++;
        string++;
    }

    return length;
}

void copy_string(char *target, char *source)
{
    while(*source)
    {
        *target = *source;
        source++;
        target++;
    }
    *target = '\0';
}

void reverse_string(char *string)
{
    int length, c;
    char *begin, *end, temp;

    length = string_length(string);

    begin = string;
    end = string;

    for ( c = 0 ; c < ( length - 1 ) ; c++ )
        end++;

    for ( c = 0 ; c < length/2 ; c++ )
    {
        temp = *end;
        *end = *begin;
        *begin = temp;

        begin++;
        end--;
    }
}
```

```

}

int compare_string(char *first, char *second)
{
    while(*first==*second)
    {
        if ( *first == '\0' || *second == '\0' )
            break;

        first++;
        second++;
    }
    if( *first == '\0' && *second == '\0' )
        return 0;
    else
        return -1;
}

```

These programs are simple and efficient.

Using them as reference, I wrote the following code:

```

#include <stdio.h>

int palindrome(char* p_cstring);

int main()
{
    char buffer1[] = "If debugging is the process of removing
software bugs, then programming must be the process of putting them
in.";
    char buffer2[] = "I am error.-random fat dude in The Legend of
Zelda 2";

    int palindrome_return = 0;

    palindrome_return = palindrome(buffer1);
    printf("[%s]\n", buffer1);
    if (palindrome_return == 1)
    {
        printf("Is a palindrome!");
    }
    else
    {
        printf("Is not a palindrome!");
    }
    printf("\n");
}

```

```
palindrome_return = palindrome(buffer2);
printf("[%s]\n", buffer2);
if (palindrome_return == 1)
{
    printf("Is a palindrome!");
}
else
{
    printf("Is not a palindrome!");
}
printf("\n");

printf("\n");

return 0;
}

#include <stdio.h>
#include <string.h>

int palindrome(char* p_cstring)
{
    char cstring[999];

    strcpy(cstring, *p_string);
    strrev(cstring);

    if(strncmp(*p_string, cstring) == 0)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
```

But this code does not compile as well.

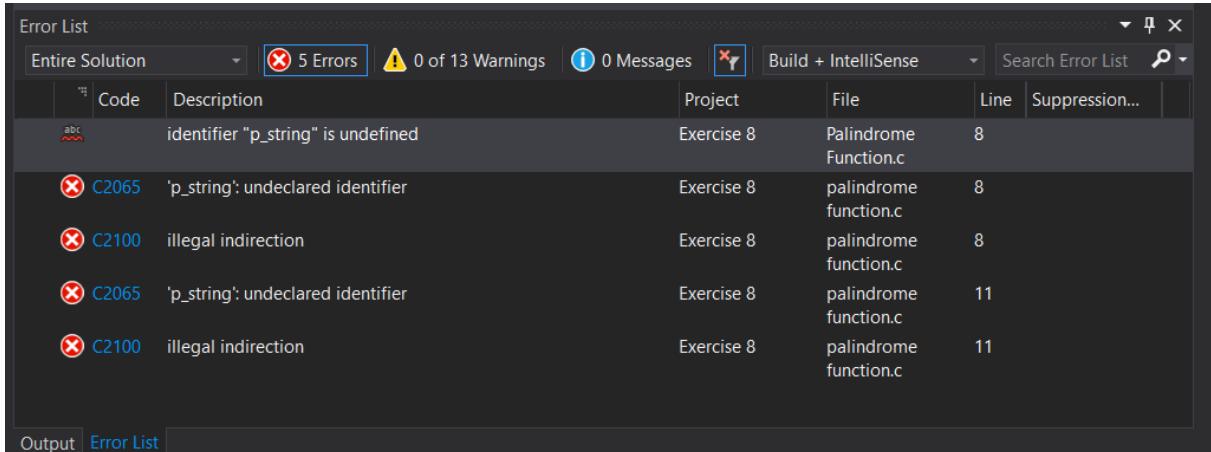


Figure 314: Compiling Errors.

I fixed some small spelling mistakes:

```
#include <stdio.h>

int palindrome(char* p_cstring);

int main()
{
    char buffer1[] = "If debugging is the process of removing
software bugs, then programming must be the process of putting them
in.";
    char buffer2[] = "I am error.-random fat dude in The Legend of
Zelda 2";

    int palindrome_return = 0;

    palindrome_return = palindrome(buffer1);
    printf("[%s]\n", buffer1);
    if (palindrome_return == 1)
    {
        printf("Is a palindrome!");
    }
    else
    {
        printf("Is not a palindrome!");
    }
    printf("\n");

    palindrome_return = palindrome(buffer2);
    printf("[%s]\n", buffer2);
```

```

if (palindrome_return == 1)
{
    printf("Is a palindrome!");
}
else
{
    printf("Is not a palindrome!");
}
printf("\n");

printf("\n");

return 0;
}
#include <stdio.h>
#include <string.h>

int palindrome(char* p_cstring)
{
    char cstring[999];

    strcpy(cstring, *p_cstring);
    strrev(cstring);

    if(strcmp(*p_cstring, cstring) == 0)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

```

This code compiles, but it does not run properly. To fix this problem, I wrote the following code:

```

#include <stdio.h>
#include <string.h>

int palindrome(char* p_cstring, char* string_copy);

int main()
{
    char buffer1[] = "Anne, I vote more cars race Rome to Vienna";
    char buffer2[] = "Anne, I vote more cars race Rome to Vienna";

```

```
char buffer3[] = "anneivotemorecarsracerometovienna";
char copy_array[999];
int palindrome_return = 0;

strcpy(copy_array, buffer1);
palindrome_return = palindrome(buffer1, copy_array);
printf("[%s]\n", buffer1);
if (palindrome_return == 1)
{
    printf("Is a palindrome!");
}
else
{
    printf("Is not a palindrome!");
}
printf("\n\n");

strcpy(copy_array, buffer2);
palindrome_return = palindrome(buffer2, copy_array);
printf("[%s]\n", buffer2);
if (palindrome_return == 1)
{
    printf("Is a palindrome!");
}
else
{
    printf("Is not a palindrome!");
}
printf("\n\n");

strcpy(copy_array, buffer3);
palindrome_return = palindrome(buffer3, copy_array);
printf("[%s]\n", buffer3);
if (palindrome_return == 1)
{
    printf("Is a palindrome!");
}
else
{
    printf("Is not a palindrome!");
}
printf("\n\n");

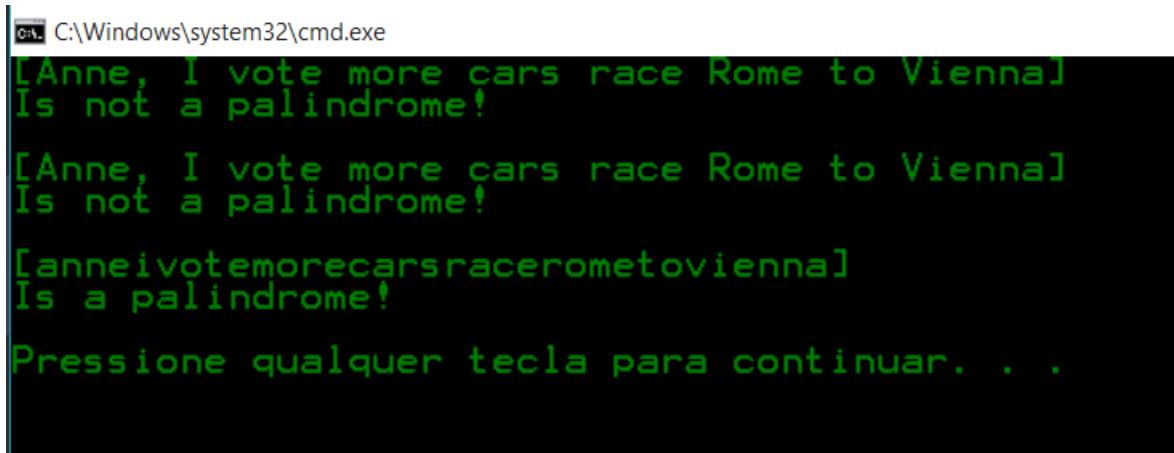
return 0;
}
```

```
#include <stdio.h>
#include <string.h>

int palindrome(char* p_cstring, char* string_copy)
{
    strrev(string_copy);

    if(strcmp(p_cstring, string_copy) == 0)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
```

This program outputs:



The screenshot shows a Windows Command Prompt window with the following text output:

```
C:\Windows\system32\cmd.exe
[Anne, I vote more cars race Rome to Vienna]
Is not a palindrome!

[Anne, I vote more cars race Rome to Vienna]
Is not a palindrome!

[anneivotemorecarsracerometovienna]
Is a palindrome!

Pressione qualquer tecla para continuar. . .
```

Figure 315: Palindrome Output.

8.3.9 Exercise 9

This exercise asks the student to:

Write the following functions:

- A function that counts and returns the number of vowels (a, e, i ,o, u) in a C-String.
- Write a function that counts and returns the number of consonants in a C-String.
- Write a function that converts a C-String to all lowercase.
- Write a function that converts a C-String to all uppercase.

Next, write a program in the `main` function which does the following, in order:

- Prompts the user to enter a string, and lets them type it in.
 - The string cannot be larger than 100 characters.
- Prints a menu of items for the user to choose from:
 1. Count the number of vowels.
 2. Count the number of consonants.
 3. Count the number of spaces.
 4. Convert to upper case.
 5. Convert to lower case.
 6. Echo input string.
 7. Enter another string.
 8. Quit.
- Prompts the user for a choice, then based upon the choice, execute the appropriate menu option, and display the result for the user to view.
- Finally, loops to repeat the displaying of the menu options until the user selected 8 to quit the program.

Using these instructions, I wrote the following code:

```
#include <stdio.h>

int number_of_vowels(char* c_string);
int number_of_consonants(char* c_string);
int number_of_spaces(char* c_string);
void to_uppercase(char* c_string);
void to_lowercase(char* c_string);
```

```
int main()
{
    char c_string[101];
    printf("Enter a string: ");
    scanf("%100s", c_string);
    printf("\n");

    int choice = 0;
    int vowels = 0;
    int consonants = 0;
    int spaces = 0;

    do
    {
        printf("Select an option:\n");
        printf("1. Count the number of vowels.\n");
        printf("2. Count the number of consonants.\n");
        printf("3. Count the number of spaces.\n");
        printf("4. Convert to upper case.\n");
        printf("5. Convert to lower case.\n");
        printf("6. Echo input string.\n");
        printf("7. Enter another string.\n");
        printf("8. Quit.\n");
        scanf("%d", &choice);
        printf("\n");

        switch (choice)
        {
            case 1:
                vowels = number_of_vowels(c_string);
                printf("The string has %d vowels.\n", vowels);
                break;
            case 2:
                consonants = number_of_consonants(c_string);
                printf("The string has %d consonants.\n",
consonants);
                break;
            case 3:
                spaces = number_of_spaces(c_string);
                printf("The string has %d spaces.\n", spaces);
                break;
            case 4:
                to_uppercase(c_string);
                printf("%s\n", c_string);
                break;
            case 5:
```

```

        to_lowercase(c_string);
        printf("%s\n", c_string);
        break;
    case 6:
        printf("%s\n", c_string);
        break;
    case 7:
        printf("Enter a string: ");
        scanf("%100s", c_string);
        printf("\n");
        break;
    }

    printf("\n");
}
while (choice != 8);

printf("\n\n");

return 0;
}

#include <stdio.h>
#include <string.h>

int number_of_vowels(char* c_string)
{
    int vowel = 0;

    for (int i = 0; i < strlen(c_string); i++)
    {
        if (c_string[i] == 65 || c_string[i] == 69 || c_string[i]
== 73 || c_string[i] == 79 || c_string[i] == 85 || c_string[i] == 97
|| c_string[i] == 101 || c_string[i] == 105 || c_string[i] == 111 || c_
string[i] == 117)
        {
            vowel++;
        }
    }

    return vowel;
}
#include <stdio.h>
#include <string.h>

int number_of_consonants(char* c_string)
{
    int consonant = 0;

```

```

for (int i = 0; i < strlen(c_string); i++)
{
    if ((c_string[i] >= 66 && c_string[i] <= 68) ||
(c_string[i] >= 70 && c_string[i] <= 72) || (c_string[i] >= 74 &&
c_string[i] <= 78) || (c_string[i] >= 80 && c_string[i] <= 84) || 
(c_string[i] >= 86 && c_string[i] <= 90) || (c_string[i] >= 98 &&
c_string[i] <= 100) || (c_string[i] >= 102 && c_string[i] <= 104) || 
(c_string[i] >= 106 && c_string[i] <= 110) || (c_string[i] >= 112 &&
c_string[i] <= 116) || (c_string[i] >= 118 && c_string[i] <= 122))
    {
        consonant++;
    }
}

return consonant;
}
#include <stdio.h>
#include <string.h>

int number_of_spaces(char* c_string)
{
    int spaces = 0;

    for (int i = 0; i < strlen(c_string); i++)
    {
        if (c_string[i] == 255)
        {
            spaces++;
        }
    }

    return spaces;
}
#include <stdio.h>
#include <string.h>

void to_uppercase(char* c_string)
{
    int vowel = 0;

    for (int i = 0; i < strlen(c_string); i++)
    {
        if (c_string[i] >= 65 && c_string[i] <= 90)
        {
            c_string[i] += 32;
        }
    }
}

```

```

        }
    }
#include <stdio.h>
#include <string.h>

void to_lowercase(char* c_string)
{
    int vowel = 0;

    for (int i = 0; i < strlen(c_string); i++)
    {
        if (c_string[i] >= 65 && c_string[i] <= 90)
        {
            c_string[i] -= 32;
        }
    }
}

```

The program compiles, but it crashes when the user inputs the choice, I realized that I was forgetting the &. To fix this problem, I wrote the following code:

```

#include <stdio.h>

int number_of_vowels(char* c_string);
int number_of_consonants(char* c_string);
int number_of_spaces(char* c_string);
void to_uppercase(char* c_string);
void to_lowercase(char* c_string);

int main()
{
    char c_string[101];
    printf("Enter a string: ");
    scanf("%100s", c_string);
    printf("\n");

    int choice = 0;
    int vowels = 0;
    int consonants = 0;
    int spaces = 0;

    do
    {
        printf("Select an option:\n");
        printf("1. Count the number of vowels.\n");
        printf("2. Count the number of consonants.\n");

```

```
printf("3. Count the number of spaces.\n");
printf("4. Convert to upper case.\n");
printf("5. Convert to lower case.\n");
printf("6. Echo input string.\n");
printf("7. Enter another string.\n");
printf("8. Quit.\n");
scanf("%d", &choice);
printf("\n");

switch (choice)
{
case 1:
    vowels = number_of_vowels(c_string);
    printf("The string has %d vowels.\n", vowels);
    break;
case 2:
    consonants = number_of_consonants(c_string);
    printf("The string has %d consonants.\n",
consonants);
    break;
case 3:
    spaces = number_of_spaces(c_string);
    printf("The string has %d spaces.\n", spaces);
    break;
case 4:
    to_uppercase(c_string);
    printf("%s\n", c_string);
    break;
case 5:
    to_lowercase(c_string);
    printf("%s\n", c_string);
    break;
case 6:
    printf("%s\n", c_string);
    break;
case 7:
    printf("Enter a string: ");
    scanf("%100s", c_string);
    printf("\n");
    break;
}

printf("\n");
}
while (choice != 8);

printf("\n\n");
```

```

        return 0;
}
#include <stdio.h>
#include <string.h>

int number_of_vowels(char* c_string)
{
    int vowel = 0;

    for (int i = 0; i < strlen(c_string); i++)
    {
        if (c_string[i] == 65 || c_string[i] == 69 || c_string[i]
== 73 || c_string[i] == 79 || c_string[i] == 85 || c_string[i] == 97
|| c_string[i] == 101 || c_string[i] == 105 || c_string[i] == 111 || c_string[i] == 117)
        {
            vowel++;
        }
    }

    return vowel;
}
#include <stdio.h>
#include <string.h>

int number_of_consonants(char* c_string)
{
    int consonant = 0;

    for (int i = 0; i < strlen(c_string); i++)
    {
        if ((c_string[i] >= 66 && c_string[i] <= 68) ||
(c_string[i] >= 70 && c_string[i] <= 72) || (c_string[i] >= 74 && c_string[i] <= 78) || (c_string[i] >= 80 && c_string[i] <= 84) || (c_string[i] >= 86 && c_string[i] <= 90) || (c_string[i] >= 98 && c_string[i] <= 100) || (c_string[i] >= 102 && c_string[i] <= 104) || (c_string[i] >= 106 && c_string[i] <= 110) || (c_string[i] >= 112 && c_string[i] <= 116) || (c_string[i] >= 118 && c_string[i] <= 122))
        {
            consonant++;
        }
    }

    return consonant;
}
#include <stdio.h>
```

```
#include <string.h>

int number_of_spaces(char* c_string)
{
    int spaces = 0;

    for (int i = 0; i < strlen(c_string); i++)
    {
        if (c_string[i] == 255)
        {
            spaces++;
        }
    }

    return spaces;
}

#include <stdio.h>
#include <string.h>

void to_uppercase(char* c_string)
{
    int vowel = 0;

    for (int i = 0; i < strlen(c_string); i++)
    {
        if (c_string[i] >= 65 && c_string[i] <= 90)
        {
            c_string[i] += 32;
        }
    }
}

#include <stdio.h>
#include <string.h>

void to_lowercase(char* c_string)
{
    int vowel = 0;

    for (int i = 0; i < strlen(c_string); i++)
    {
        if (c_string[i] >= 65 && c_string[i] <= 90)
        {
            c_string[i] -= 32;
        }
    }
}
```

Running and testing the program, I realized two problems:

Both 4 and 5 options converts the string to lowercase;

The option 7 does not prints the new string.

```
C:\Windows\system32\cmd.exe
1 Enter another string.
2 Quit.

The string has 3 vowels.
Select an option:
1. Count the number of vowels.
2. Count the number of consonants.
3. Count the number of spaces.
4. Convert to upper case.
5. Convert to lower case.
6. Echo input string.
7. Enter another string.
8. Quit.
2

The string has 3 consonants.
Select an option:
1. Count the number of vowels.
2. Count the number of consonants.
3. Count the number of spaces.
4. Convert to upper case.
5. Convert to lower case.
6. Echo input string.
7. Enter another string.
8. Quit.
3

The string has 0 spaces.
Select an option:
1. Count the number of vowels.
2. Count the number of consonants.
3. Count the number of spaces.
4. Convert to upper case.
5. Convert to lower case.
6. Echo input string.
7. Enter another string.
8. Quit.
4

paludo
Select an option:
1. Count the number of vowels.
2. Count the number of consonants.
3. Count the number of spaces.
```

Figure 316: String Manipulator Menu Wrong Output.

To fix these problems, I wrote the following code:

```
#include <stdio.h>

int number_of_vowels(char* c_string);
int number_of_consonants(char* c_string);
int number_of_spaces(char* c_string);
void to_uppercase(char* c_string);
void to_lowercase(char* c_string);

int main()
{
    char c_string[101];
    printf("Enter a string: ");
    scanf("%100s", c_string);
    printf("\n");

    int choice = 0;
```

```
int vowels = 0;
int consonants = 0;
int spaces = 0;

do
{
    printf("Select an option:\n");
    printf("1. Count the number of vowels.\n");
    printf("2. Count the number of consonants.\n");
    printf("3. Count the number of spaces.\n");
    printf("4. Convert to upper case.\n");
    printf("5. Convert to lower case.\n");
    printf("6. Echo input string.\n");
    printf("7. Enter another string.\n");
    printf("8. Quit.\n");
    scanf("%d", &choice);
    printf("\n");

    switch (choice)
    {
        case 1:
            vowels = number_of_vowels(c_string);
            printf("The string has %d vowels.\n", vowels);
            break;
        case 2:
            consonants = number_of_consonants(c_string);
            printf("The string has %d consonants.\n",
consonants);
            break;
        case 3:
            spaces = number_of_spaces(c_string);
            printf("The string has %d spaces.\n", spaces);
            break;
        case 4:
            to_uppercase(c_string);
            printf("%s\n", c_string);
            break;
        case 5:
            to_lowercase(c_string);
            printf("%s\n", c_string);
            break;
        case 6:
            printf("%s\n", c_string);
            break;
        case 7:
            printf("Enter a string: ");
            scanf("%100s", c_string);
    }
}
```

```

        printf("%s\n", c_string);
        break;
    }

    printf("\n");
}
while (choice != 8);

return 0;
}

#include <stdio.h>
#include <string.h>

int number_of_vowels(char* c_string)
{
    int vowel = 0;

    for (int i = 0; i < strlen(c_string); i++)
    {
        if (c_string[i] == 65 || c_string[i] == 69 || c_string[i]
== 73 || c_string[i] == 79 || c_string[i] == 85 || c_string[i] == 97
|| c_string[i] == 101 || c_string[i] == 105 || c_string[i] == 111 || c_string[i] == 117)
        {
            vowel++;
        }
    }

    return vowel;
}
#include <stdio.h>
#include <string.h>

int number_of_consonants(char* c_string)
{
    int consonant = 0;

    for (int i = 0; i < strlen(c_string); i++)
    {
        if ((c_string[i] >= 66 && c_string[i] <= 68) ||
(c_string[i] >= 70 && c_string[i] <= 72) || (c_string[i] >= 74 && c_string[i] <= 78) || (c_string[i] >= 80 && c_string[i] <= 84) ||
(c_string[i] >= 86 && c_string[i] <= 90) || (c_string[i] >= 98 && c_string[i] <= 100) || (c_string[i] >= 102 && c_string[i] <= 104) ||
(c_string[i] >= 106 && c_string[i] <= 110) || (c_string[i] >= 112 && c_string[i] <= 116) || (c_string[i] >= 118 && c_string[i] <= 122))
        {

```

```
        consonant++;
    }

    return consonant;
}

#include <stdio.h>
#include <string.h>

int number_of_spaces(char* c_string)
{
    int spaces = 0;

    for (int i = 0; i < strlen(c_string); i++)
    {
        if (c_string[i] == 255)
        {
            spaces++;
        }
    }

    return spaces;
}

#include <stdio.h>
#include <string.h>

void to_uppercase(char* c_string)
{
    int vowel = 0;

    for (int i = 0; i < strlen(c_string); i++)
    {
        if (c_string[i] >= 97 && c_string[i] <= 122)
        {
            c_string[i] += 32;
        }
    }
}

#include <stdio.h>
#include <string.h>

void to_lowercase(char* c_string)
{
    int vowel = 0;
```

```
    if (c_string[i] >= 65 && c_string[i] <= 90)
    {
        c_string[i] -= 32;
    }
}
```

But the program still not running correctly when converting to uppercase.

```
C:\Windows\system32\cmd.exe
Enter a string: Paludex
Select an option:
1. Count the number of vowels.
2. Count the number of consonants.
3. Count the number of spaces.
4. Convert to upper case.
5. Convert to lower case.
6. Echo input string.
7. Enter another string.
8. Quit.
4

Púloáay
Select an option:
1. Count the number of vowels.
2. Count the number of consonants.
3. Count the number of spaces.
4. Convert to upper case.
5. Convert to lower case.
6. Echo input string.
7. Enter another string.
8. Quit.
7

Enter a string: String
String
Select an option:
1. Count the number of vowels.
2. Count the number of consonants.
3. Count the number of spaces.
4. Convert to upper case.
5. Convert to lower case.
6. Echo input string.
7. Enter another string.
8. Quit.
8

Pressione qualquer tecla para continuar. . .
```

Figure 317: Convert to Uppercase Wrong Output.

To fix this problem, I wrote the following code:

```
#include <stdio.h>

int number_of_vowels(char* c_string);
int number_of_consonants(char* c_string);
int number_of_spaces(char* c_string);
void to_uppercase(char* c_string);
void to_lowercase(char* c_string);

int main()
{
    char c_string[101];
    printf("Enter a string: ");
    scanf(" %100s", c_string);
    printf("\n");
```

```
int choice = 0;
int vowels = 0;
int consonants = 0;
int spaces = 0;

do
{
    printf("Select an option:\n");
    printf("1. Count the number of vowels.\n");
    printf("2. Count the number of consonants.\n");
    printf("3. Count the number of spaces.\n");
    printf("4. Convert to upper case.\n");
    printf("5. Convert to lower case.\n");
    printf("6. Echo input string.\n");
    printf("7. Enter another string.\n");
    printf("8. Quit.\n");
    scanf("%d", &choice);
    printf("\n");

    switch (choice)
    {
        case 1:
            vowels = number_of_vowels(c_string);
            printf("The string has %d vowels.\n", vowels);
            break;
        case 2:
            consonants = number_of_consonants(c_string);
            printf("The string has %d consonants.\n",
consonants);
            break;
        case 3:
            spaces = number_of_spaces(c_string);
            printf("The string has %d spaces.\n", spaces);
            break;
        case 4:
            to_uppercase(c_string);
            printf("\n");
            break;
        case 5:
            to_lowercase(c_string);
            printf("\n");
            break;
        case 6:
            printf("%s\n", c_string);
            break;
        case 7:
            printf("Enter a string: ");
    }
}
```

```

        scanf("%100s", c_string);
        printf("%s\n", c_string);
        break;
    }

    printf("\n");
}
while (choice != 8);

return 0;
}

#include <stdio.h>
#include <string.h>

int number_of_vowels(char* c_string)
{
    int vowel = 0;

    for (int i = 0; i < strlen(c_string); i++)
    {
        if (c_string[i] == 65 || c_string[i] == 69 || c_string[i]
== 73 || c_string[i] == 79 || c_string[i] == 85 || c_string[i] == 97
|| c_string[i] == 101 || c_string[i] == 105 || c_string[i] == 111 || c_string[i] == 117)
        {
            vowel++;
        }
    }

    return vowel;
}

#include <stdio.h>
#include <string.h>

int number_of_consonants(char* c_string)
{
    int consonant = 0;

    for (int i = 0; i < strlen(c_string); i++)
    {
        if (((c_string[i] >= 66 && c_string[i] <= 68) ||
(c_string[i] >= 70 && c_string[i] <= 72) || (c_string[i] >= 74 && c_string[i] <= 78) || (c_string[i] >= 80 && c_string[i] <= 84) ||
(c_string[i] >= 86 && c_string[i] <= 90) || (c_string[i] >= 98 && c_string[i] <= 100) || (c_string[i] >= 102 && c_string[i] <= 104) ||
(c_string[i] >= 106 && c_string[i] <= 110) || (c_string[i] >= 112 && c_string[i] <= 116) || (c_string[i] >= 118 && c_string[i] <= 122)))

```

```
        {
            consonant++;
        }
    }

    return consonant;
}

#include <stdio.h>
#include <string.h>

int number_of_spaces(char* c_string)
{
    int spaces = 0;

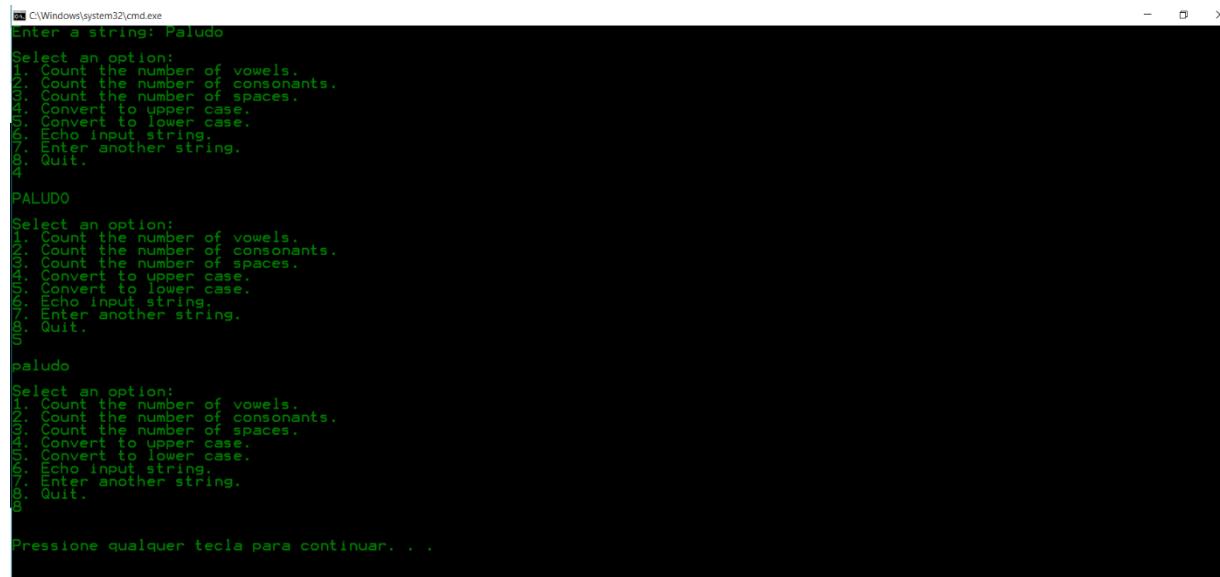
    for (int i = 0; i < strlen(c_string); i++)
    {
        if (c_string[i] == 255)
        {
            spaces++;
        }
    }

    return spaces;
}
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void to_uppercase(char* c_string)
{
    for (int i = 0; i < strlen(c_string); i++)
    {
        putchar(toupper(c_string[i]));
    }
}
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void to_lowercase(char* c_string)
{
    for (int i = 0; i < strlen(c_string); i++)
    {
        putchar(tolower(c_string[i]));
    }
}
```

This program outputs:



```
C:\Windows\system32\cmd.exe
Enter a string: Paludo
Select an option:
1. Count the number of vowels.
2. Count the number of consonants.
3. Count the number of spaces.
4. Convert to upper case.
5. Convert to lower case.
6. Echo input string.
7. Enter another string.
8. Quit.
4
PALUDO
Select an option:
1. Count the number of vowels.
2. Count the number of consonants.
3. Count the number of spaces.
4. Convert to upper case.
5. Convert to lower case.
6. Echo input string.
7. Enter another string.
8. Quit.
5
paludo
Select an option:
1. Count the number of vowels.
2. Count the number of consonants.
3. Count the number of spaces.
4. Convert to upper case.
5. Convert to lower case.
6. Echo input string.
7. Enter another string.
8. Quit.
3

Pressione qualquer tecla para continuar. . .
```

Figure 318: String Manipulator Menu Output.

9 Nine

9.1 Lectures

Implicit type conversion is performed by the compiler, explicit type conversion is performed by the programmer using a cast operator. It is a good practice to use explicit type cast operator when converting between two types because it highlights to other programmers that data conversion is occurring.

Type casting is a way to convert a variable from one data type to another.

Void variables can store any type of information.

Dynamic memory allocation (malloc) can be used with special operators. This allows the programmer to have a dynamic sized array.

The assert library was also introduced, the programmer can use assert to verify assumptions. This feature is very useful to debug the code.

9.2 Examples

9.2.1 Example 1

This program is an example of how to declare and initialize multi-dimensional arrays.

This was the written code:

```
#include <stdio.h>

int main()
{
    int the_array[5][4] =
    {
        { 0, 1, 2, 3 },
        { 4, 5, 6, 7 },
        { 8, 9, 10, 11 },
        { 12, 13, 14, 15 },
        { 16, 17, 18, 19 }
    };

    for (int k = 0; k < 5; ++k)
    {
        for (int i = 0; i < 4; ++i)
        {
            printf("%d ", the_array[k][i]);
        }

        printf("\n");
    }

    printf("\n\n");

    return 0;
}
```

This program outputs:



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window displays a 2D array of integers arranged in four rows and five columns. The numbers are colored green. A message at the bottom of the window reads "Pressione qualquer tecla para continuar. . .".

0	1	2	3	
4	5	6	7	
8	9	10	11	
12	13	14	15	
16	17	18	19	

Pressione qualquer tecla para continuar. . .

Figure 319: Multi-Dimensional Array.

9.2.2 Example 2

This program does almost the same as the previous one, but with a character array.

This was the written code:

```
#include <stdio.h>

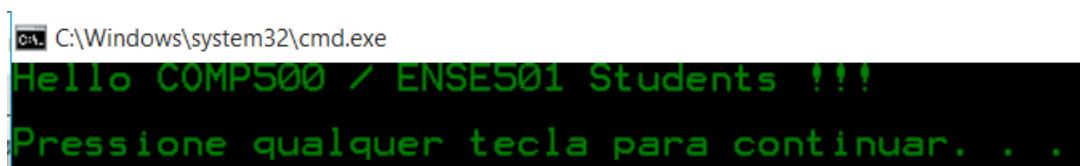
int main()
{
    char* strings[] =
    {
        "Hello",
        "COMP500",
        "/",
        "ENSE501",
        "Students",
        "!!!!"
    };

    for (int k = 0; k < 6; ++k)
    {
        printf("%s ", strings[k]);
    }

    printf("\n\n");

    return 0;
}
```

The program outputs:



```
C:\Windows\system32\cmd.exe
Hello COMP500 / ENSE501 Students !!!
Pressione qualquer tecla para continuar. . .
```

Figure 320: Multi-Dimensional Character Array.

9.2.3 Example 3

This program should be tested with some changes in the Microsoft Visual Studio debugging settings.

This was the written code:

```
#include <stdio.h>

int main(int argc, char* argv[])
{
    printf("There are %d arguments supplied.\n", argc);

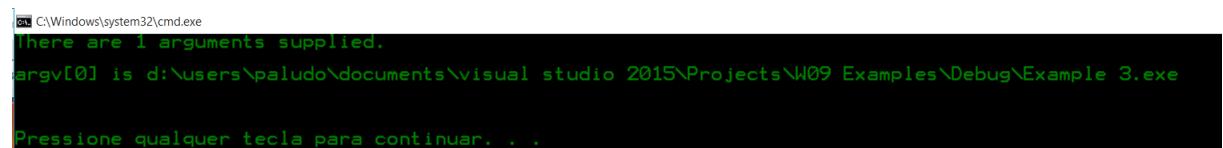
    printf("\n");

    for (int k = 0; k < argc; ++k)
    {
        printf("argv[%d] is %s\n", k, argv[k]);
        printf("\n");
    }

    printf("\n\n");

    return 0;
}
```

Running the program, it outputs:



```
C:\Windows\system32\cmd.exe
There are 1 arguments supplied.
argv[0] is d:\users\paludo\documents\visual studio 2015\Projects\W09 Examples\Debug\Example 3.exe

Pressione qualquer tecla para continuar. . .
```

Figure 321: The Parameters of main.

9.2.4 Example 4

This program examples how does the dynamic memory allocation (malloc) works. If the malloc allocates the memory correctly, the user will be asked to input how old he/she is.

This was the written code:

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

int main()
{
    int* p_age = 0;

    p_age = (int*)malloc(sizeof(int));

    if (0 == p_age)
    {
        printf("Malloc failed to allocate memory!\n");
    }
    else
    {
        printf("How old are you? ");

        scanf("%d", p_age);

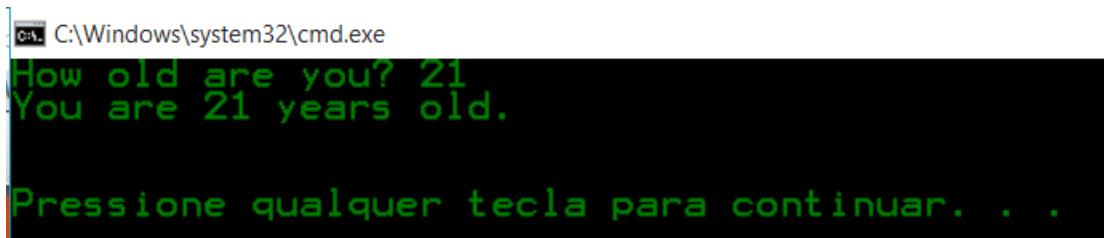
        printf("You are %d years old.\n", *p_age);

        free(p_age);
        p_age = 0;
    }

    printf("\n\n");

    return 0;
}
```

This program outputs:



A screenshot of a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
How old are you? 21
You are 21 years old.

Pressione qualquer tecla para continuar. . .

Figure 322: Simple malloc.

9.2.5 Example 5

This program uses malloc to take user text input.

This was the written code:

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

int main()
{
    char* p_buffer = 0;

    p_buffer = (char*)malloc(sizeof(char) * 1024);

    if (0 == p_buffer)
    {
        printf("Malloc failed to allocate memory!\n");
    }
    else
    {
        printf("Enter some text: ");

        scanf("%1023s", p_buffer);

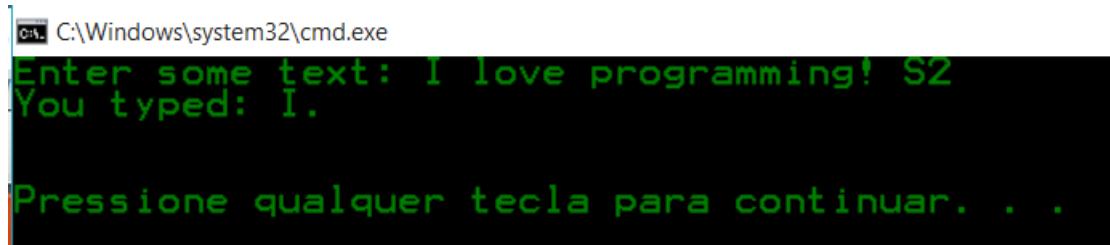
        printf("You typed: %s.\n", p_buffer);

        free(p_buffer);
        p_buffer = 0;
    }

    printf("\n\n");

    return 0;
}
```

The program outputs:



A screenshot of a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
Enter some text: I love programming! S2
You typed: I.

Pressione qualquer tecla para continuar. . .

Figure 323: Input Buffer.

9.3 Exercises

9.3.1 Exercise 1

Given the following code:

```
int main(int argc, char* argv[])
{
    int data[7] = { 9, 2, 7, 1, 8, 4, 5 };
    int temp = 0;
    int j = 0;

    for (int i = 1; i < 7; ++i)
    {
        temp = data[i];
        j = i - 1;

        while (temp < data[j] && j >= 0)
        {
            data[j + 1] = data[j];
            j = j - 1;
        }

        data[j + 1] = temp;
    }

    return 0;
}
```

The programmer should modularize the program such that the sorting algorithm is reusable with an int array of any size.

To do so, a void function that takes an int array via pointer, and the dimension of the array that is being passed in should be created.

Another function might be created to print the array.

This was the written code:

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

void print_array(int* pointer_array, int array_size);
void sort_array(int* pointer_array, int array_size);

int main(int argc, char* argv[])
{
    int array_size;
    printf("Enter the array size: ");
    scanf("%d", &array_size);
    printf("\n\n");

    int *int_array = (int*)malloc(array_size, sizeof(int));

    print_array(int_array, array_size);

    sort_array(int_array, array_size);

    print_array(int_array, array_size);

    printf("\n\n");

    return 0;
}

#include <stdio.h>
#include <stdlib.h>

void print_array(int* pointer_array, int array_size)
{
    for (int i = 0; i < array_size; i++)
    {
        printf("%d, ", pointer_array[i]);
    }

    printf("\n\n");
}

#include <stdio.h>
#include <stdlib.h>

void sort_array(int* pointer_array, int array_size)
{
    printf("Sorting array.");
}
```

```
for (int i = 0; i < array_size; i++)
{
    pointer_array[i] = 1;
}

printf("\n\n");
}
```

This program runs correctly, but crashes in the end.

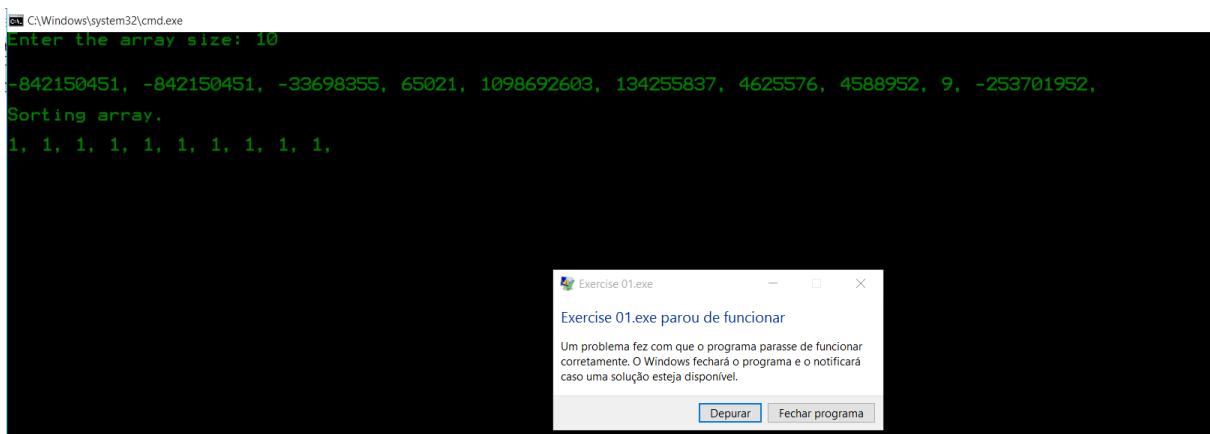


Figure 324: Simple malloc Crash.

To solve the problem, I just cleaned the solution and restarted my computer.



Figure 325: Simple malloc Output.

9.3.2 Exercise 2

This program should declare a five by five int array, populate each element in the array with a value of zero, then fill the array diagonally, from top left to bottom right, with the following values:

2				
	4			
		8		
			16	
				32

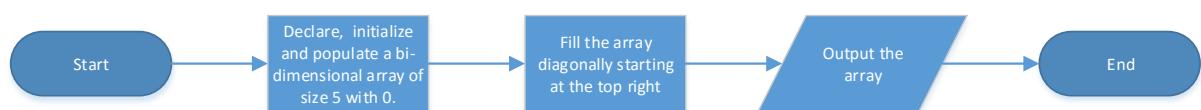
Figure 326: Diagonal Square Array Fill Table.

The program should output:

```
2  0  0  0  0
0  4  0  0  0
0  0  8  0  0
0  0  0  16 0
0  0  0  0  32
```

Figure 327: Diagonal Square Array Fill.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

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

```
int main()
{
    int diagonal_array[5][5] = { 0 };

    for (int i = 0; i < 5; i++)
    {
        for (int n = 0; n < 5; n++)
        {
            printf("%d, ", diagonal_array[i][n]);
        }
        printf("\n\n");
    }

    printf("Populating array.\n\n");

    int diagonal_variable = 1;

    for (int row = 0; row < 5; row++)
    {
        diagonal_variable = diagonal_variable * 2;

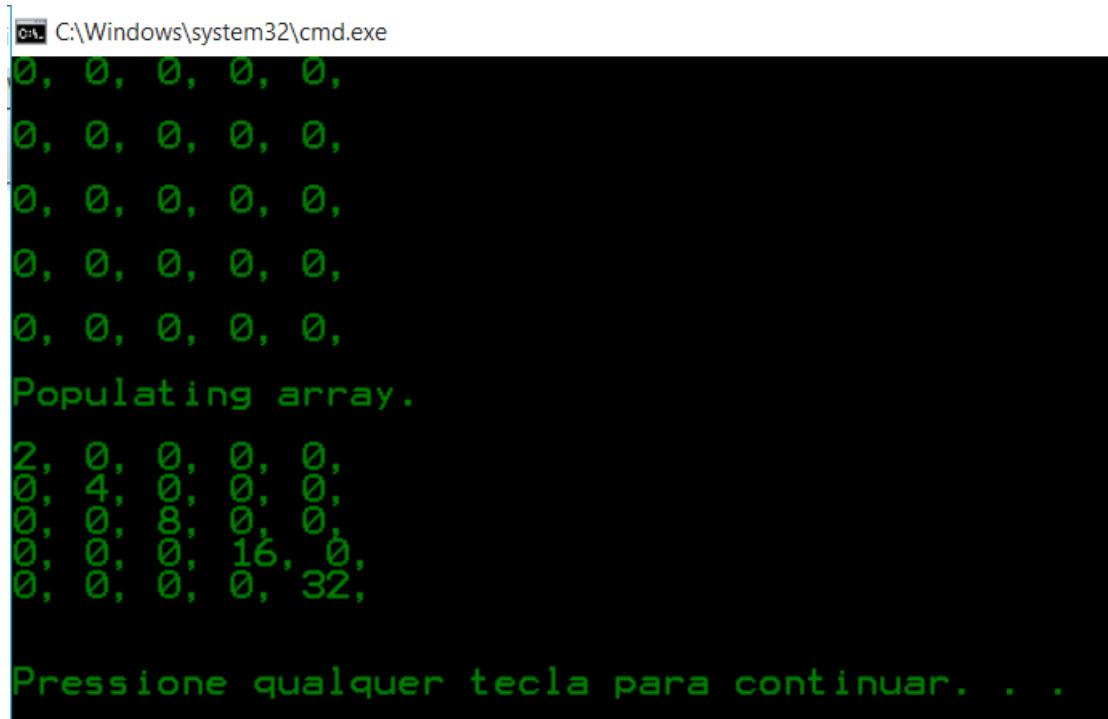
        for (int column = 0; column < 5; column++)
        {
            if (row == column)
            {
                diagonal_array[row][column] =
diagonal_variable;
            }
        }
    }

    for (int i = 0; i < 5; i++)
    {
        for (int n = 0; n < 5; n++)
        {
            printf("%d, ", diagonal_array[i][n]);
        }
        printf("\n");
    }

    printf("\n\n");

    return 0;
}
```

This program outputs:



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The window displays the following text:

```
0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
Populating array.
2, 0, 0, 0, 0,
0, 4, 0, 0, 0,
0, 0, 8, 0, 0,
0, 0, 0, 16, 0,
0, 0, 0, 0, 32,
Pressione qualquer tecla para continuar. . .
```

Figure 328: Diagonal Square Array Fill Output.

9.3.3 Exercise 3

This program does the almost the same thing as the previous one. But with this pattern:

				80
				40
			20	
		10		
5				

Figure 329: Diagonal Square Array Fill Again Pattern.

The program should output:

```
0 0 0 0 80
0 0 0 40 0
0 0 20 0 0
0 10 0 0 0
5 0 0 0 0
```

Figure 330: Diagonal Square Array Fill Again.

This was the written code:

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

int main()
{
    int diagonal_array[5][5] = { 0 };

    for (int i = 0; i < 5; i++)
    {
        for (int n = 0; n < 5; n++)
        {
            printf("%d, ", diagonal_array[i][n]);
        }
    }
}
```

```
        }
        printf("\n\n");
    }

printf("Populating array.\n\n");

int diagonal_variable = 160;
int x = 5;

for (int row = 0; row < 5; row++)
{
    diagonal_variable = diagonal_variable / 2;
    x -= 1;

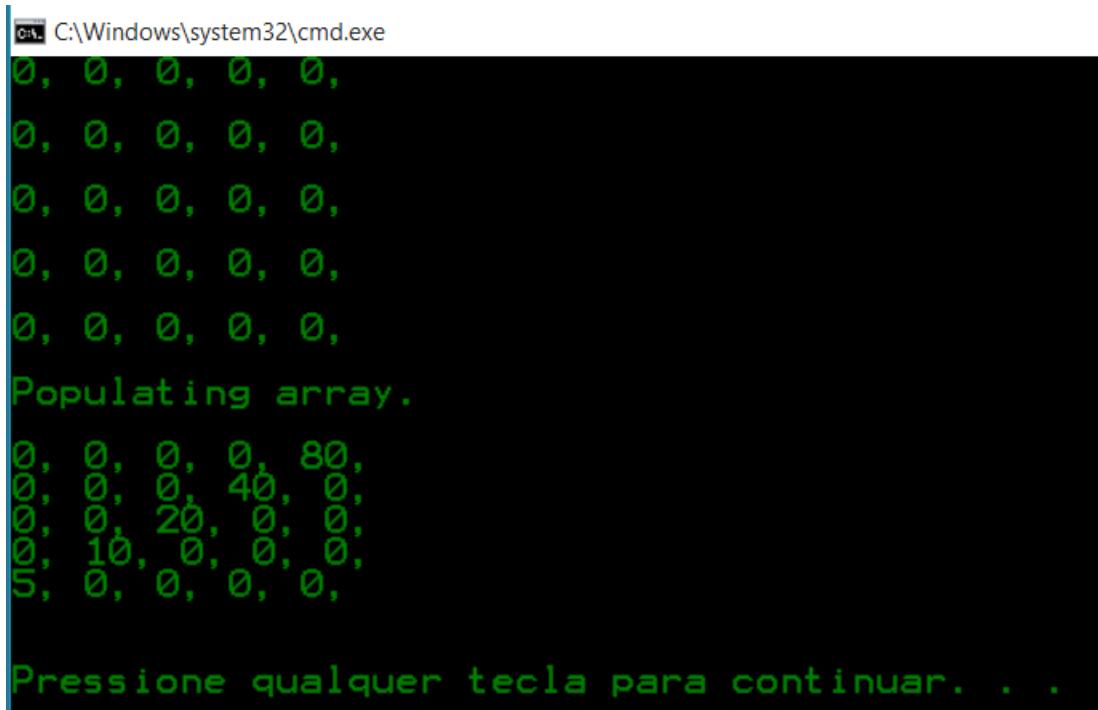
    for (int column = 0; column < 5; column++)
    {
        if (column == x)
        {
            diagonal_array[row][column] =
diagonal_variable;
        }
    }
}

for (int i = 0; i < 5; i++)
{
    for (int n = 0; n < 5; n++)
    {
        printf("%d, ", diagonal_array[i][n]);
    }
    printf("\n");
}

printf("\n\n");

return 0;
}
```

The program outputs:



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The window displays the following text:

```
0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
Populating array.
0, 0, 0, 0, 80,
0, 0, 0, 40, 0,
0, 0, 20, 0, 0,
0, 10, 0, 0, 0,
5, 0, 0, 0, 0,
Pressione qualquer tecla para continuar. . .
```

Figure 331: Diagonal Square Array Fill Again Output.

9.3.4 Exercise 4

This program does the same as the previous one, but with the following pattern:

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

Figure 332: Square Array Fill Pattern.

The program should output:

```
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
```

Figure 333: Square Array Fill.

This was the written code:

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

int main()
{
    int diagonal_array[5][5] = { 0 };

    for (int i = 0; i < 5; i++)
    {
        for (int n = 0; n < 5; n++)
        {
            printf("%d, ", diagonal_array[i][n]);
        }
    }
}
```

```
        }
        printf("\n\n");
    }

printf("Populating array.\n\n");

int element = 1;

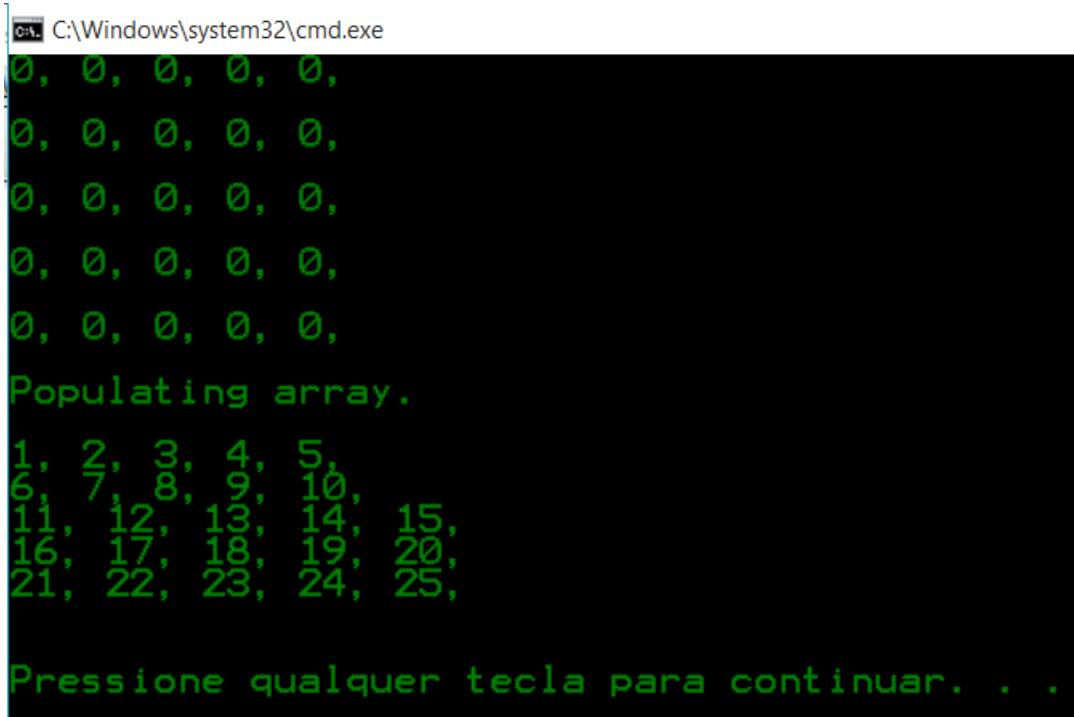
for (int row = 0; row < 5; row++)
{
    for (int column = 0; column < 5; column++)
    {
        diagonal_array[row][column] = element;
        element++;
    }
}

for (int i = 0; i < 5; i++)
{
    for (int n = 0; n < 5; n++)
    {
        printf("%d, ", diagonal_array[i][n]);
    }
    printf("\n");
}

printf("\n\n");

return 0;
}
```

The program outputs:



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The output displays the filling of a 5x5 square array. It starts with five rows of zeros, followed by a message 'Populating array.', then five rows of integers from 1 to 25. Finally, it prompts the user to press any key to continue.

```
0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
Populating array.
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,
Pressione qualquer tecla para continuar. . .
```

Figure 334: Square Array Fill Output.

9.3.5 Exercise 5

This program should have a matrix “a”, with these values:

1	3	2	5
4	5	2	4
2	1	2	2
5	3	3	1

Figure 335: Matrix A.

And a matrix “b”:

2	2	4	1
1	-5	2	3
4	3	-2	5
6	7	3	2

Figure 336: Matrix B.

And a third matrix that adds each corresponding element of these two matrix.

The program should output:

Matrix A:

1	3	2	5
4	5	2	4
1	1	2	2
5	3	3	1

Matrix B:

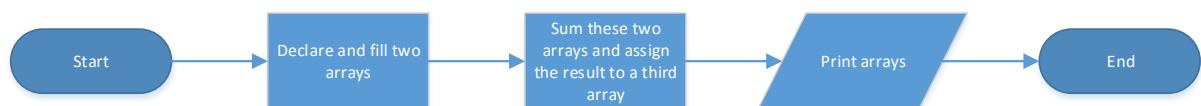
2	2	4	1
1	-5	2	3
3	3	-2	5
6	7	3	2

Result:

3	5	6	6
5	0	4	7
4	4	0	7
11	10	6	3

Figure 337: Matrix Addition.

To develop this program, I draw the following flowchart:



This was the written code:

```

#include <stdio.h>
#include <stdlib.h>

int main()
{
    int matrix_a[4][4] =
    {
        { 1, 3, 2, 5 },
        { 4, 5, 2, 4 },
        { 2, 1, 2, 2 },
        { 5, 3, 3, 1 }
    }
}
  
```

```
        { 5, 3, 3, 1 }
};

printf("Matrix A:\n\n");

printf("\n\n");

for (int i = 0; i < 4; i++)
{
    for (int n = 0; n < 4; n++)
    {
        printf("%d, ", matrix_a[i][n]);
    }
    printf("\n\n");
}

printf("\n\n");

int matrix_b[4][4] =
{
    { 2, 2, 4, 1 },
    { 1, -5, 2, 3 },
    { 4, 3, -2, 5 },
    { 6, 7, 3, 2 }
};

printf("Matrix B:\n\n");

for (int i = 0; i < 4; i++)
{
    for (int n = 0; n < 4; n++)
    {
        printf("%d, ", matrix_b[i][n]);
    }
    printf("\n\n");
}

printf("\n\n");

int result[4][4];

for (int row = 0; row < 4; row++)
{
    for (int column = 0; column < 4; column++)
    {
        result[row][column] = matrix_a[row][column] +
matrix_b[row][column];
```

```
        }
        printf("\n");
    }

printf("Result:\n\n");

for (int i = 0; i < 4; i++)
{
    for (int n = 0; n < 4; n++)
    {
        printf("%d, ", result[i][n]);
    }
    printf("\n\n");
}

printf("\n\n");

return 0;
}
```

This program outputs:

```
C:\Windows\system32\cmd.exe
Matrix A:
1, 3, 2, 5,
4, 5, 2, 4,
2, 1, 2, 2,
5, 3, 3, 1,

Matrix B:
2, 2, 4, 1,
1, -5, 2, 3,
4, 3, -2, 5,
6, 7, 3, 2,

Result:
3, 5, 6, 6,
5, 0, 4, 7,
6, 4, 0, 7,
11, 10, 6, 3,

Pressione qualquer tecla para continuar. . .
```

Figure 338: Matrix Addition Output.

9.3.6 Exercise 6

This program does the same as the previous one, but it multiplies the arrays.

The program should output:

Matrix A:

1	3	2	5
4	5	2	4
1	1	2	2
5	3	3	1

Matrix B:

2	2	4	1
1	-5	2	3
3	3	-2	5
6	7	3	2

Result:

41	28	21	30
43	17	34	37
21	17	8	18
28	11	23	31

Figure 339: Matrix Multiplication.

This was the written code:

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

int main()
{
    int matrix_a[4][4] =
    {
        { 1, 3, 2, 5 },
        { 4, 5, 2, 4 },
        { 2, 1, 2, 2 },
        { 5, 3, 3, 1 }
    }
```

```
        { 5, 3, 3, 1 }
};

printf("Matrix A:\n\n");

printf("\n\n");

for (int i = 0; i < 4; i++)
{
    for (int n = 0; n < 4; n++)
    {
        printf("%d, ", matrix_a[i][n]);
    }
    printf("\n\n");
}

printf("\n\n");

int matrix_b[4][4] =
{
    { 2, 2, 4, 1 },
    { 1, -5, 2, 3 },
    { 4, 3, -2, 5 },
    { 6, 7, 3, 2 }
};

printf("Matrix B:\n\n");

for (int i = 0; i < 4; i++)
{
    for (int n = 0; n < 4; n++)
    {
        printf("%d, ", matrix_b[i][n]);
    }
    printf("\n\n");
}

printf("\n\n");

int result[4][4];

for (int row = 0; row < 4; row++)
{
    for (int column = 0; column < 4; column++)
    {
        result[row][column] = matrix_a[row][column] *
matrix_b[row][column];
```

```
        }
        printf("\n");
    }

printf("Result:\n\n");

for (int i = 0; i < 4; i++)
{
    for (int n = 0; n < 4; n++)
    {
        printf("%d, ", result[i][n]);
    }
    printf("\n\n");
}

printf("\n\n");

return 0;
}
```

This program outputs:

```
C:\Windows\system32\cmd.exe
Matrix A:
1, 3, 2, 5,
4, 5, 2, 4,
2, 1, 2, 2,
5, 3, 3, 1,

Matrix B:
2, 2, 4, 1,
1, -5, 2, 3,
4, 3, -2, 5,
6, 7, 3, 2,

Result:
2, 6, 8, 5,
4, -25, 4, 12,
8, 3, -4, 10,
30, 21, 9, 2,

Pressione qualquer tecla para continuar. . .
```

Figure 340: Matrix Multiplication.

9.3.7 Exercise 7

This program should allocate space for an array of three int values on the heap using a single malloc. After allocating, the array should be populated as follows:

```
First Element: 150
Second Element: 275
Third Element: 400
```

Figure 341: Small malloc.

To develop this program I draw the following flowchart:



This was the written code:

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

int main()
{
    int *int_array = (int*)malloc(3, sizeof(int));

    int_array[0] = 150;
    int_array[1] = 275;
    int_array[2] = 400;

    printf("First Element: %d\n", int_array[0]);
    printf("Second Element: %d\n", int_array[1]);
    printf("Third Element: %d", int_array[2]);

    printf("\n\n");

    return 0;
}
```

This program outputs:



A screenshot of a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
First Element: 150
Second Element: 275
Third Element: 400
Pressione qualquer tecla para continuar. . .

Figure 342: Small malloc Output.

9.3.8 Exercise 8

This program asks the user to input an array size and populates each element with user input. The program should output:

```
How many int elements will you enter? 7

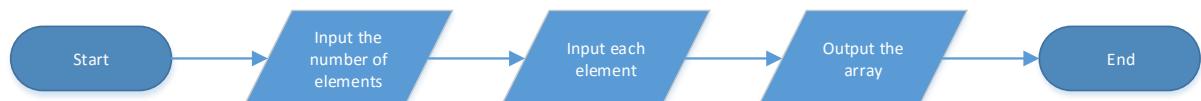
Element 1: 12
Element 2: 22
Element 3: 14
Element 4: 7
Element 5: 453
Element 6: 43
Element 7: 695

The array stores the following values:

12, 22, 14, 7, 453, 43, 695
```

Figure 343: User Defined Array Size.

To develop this program, I draw the following flowchart:



This was the written code:

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

int main(int argc, char* argv[])
{
    int array_size;
    printf("How many int elements will you enter? ");
    scanf("%d", &array_size);
    printf("\n\n");
```

```
int *int_array = (int*)malloc(array_size, sizeof(int));

for (int size = 0; size < array_size; size++)
{
    printf("Element %d: ", size + 1);
    scanf("%d", &int_array[size]);
}

printf("\n\n");

printf("The array stores the following values:");

printf("\n\n");

for (int size = 0; size < array_size; size++)
{
    printf("%d, ", int_array[size]);
}

printf("\n\n");

return 0;
}
```

This program outputs:

```
C:\Windows\system32\cmd.exe
How many int elements will you enter? 5

Element 1: 1
Element 2: 2
Element 3: 3
Element 4: 4
Element 5: 5

The array stores the following values:
1, 2, 3, 4, 5,
Pressione qualquer tecla para continuar. . .
```

Figure: 344: User Defined Array Size Output.

9.3.9 Exercise 9

Given the following code:

```
#include <stdio.h>

int main()
{
    char* input = 0;

    printf("Enter your name: ");
    scanf("%s", input);

    printf("Your name is %s.\n", input);

    return 0;
}
```

The programmer should fix it using malloc.

This was the fixed code:

```
#include <stdio.h>

int main()
{
    char* input = 0;

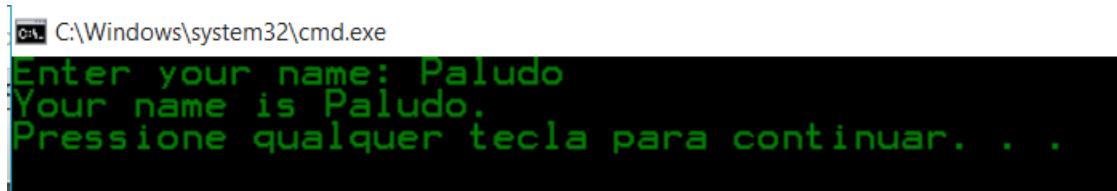
    input = (char*)malloc(sizeof(char) * 100);

    printf("Enter your name: ");
    scanf("%99s", input);

    printf("Your name is %s.\n", input);

    return 0;
}
```

This program outputs:



A screenshot of a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
Enter your name: Paludo
Your name is Paludo.
Pressione qualquer tecla para continuar. . .

Figure 345: Broken C-String Output.

9.3.10 Exercise 10

This program is almost the same as the previous one.

Using malloc, the programmer should fix the following code:

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

int main()
{
    char input[256];

    printf("Enter your name: ");
    scanf("%255s", input);

    char* p_heap_buffer = (char*)malloc(strlen(input));
    strcpy(p_heap_buffer, input);

    printf("Your name is %s.\n", p_heap_buffer);

    return 0;
}
```

This is the fixed code:

```
#include <stdio.h>

int main()
{
    char* input = 0;

    input = (char*)malloc(sizeof(char) * 256);

    printf("Enter your name: ");
    scanf("%255s", input);

    printf("Your name is %s.\n", input);

    return 0;
}
```

This program outputs:



A screenshot of a Windows command prompt window titled 'cmd C:\Windows\system32\cmd.exe'. The window contains the following text:
Enter your name: Paludo
Your name is Paludo.
Pressione qualquer tecla para continuar. . .

Figure 346: Another Broken C-String Output.

9.3.11 Exercise 11

The exercise gives a program that leaks heap memory:

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

int* create_array(int size)
{
    int* p_data = (int*)malloc(size * sizeof(int));

    return p_data;
}

void fill_array(int* p_array, int size)
{
    for (int k = 0; k < size; ++k)
    {
        printf("Enter a value: ");
        scanf("%d", &(p_array[k]));
    }
}

int main()
{
    int* first_array = create_array(5);
    fill_array(first_array, 5);

    int* second_array = create_array(3);
    fill_array(second_array, 3);

    int* third_array = create_array(3);
    fill_array(third_array, 3);

    return 0;
}
```

Using crtdebug.h, I detected the memory leak.

```
#include <stdio.h>
#include <stdlib.h>
#include <crtdebug.h>

int* create_array(int size)
```

```

{
    int* p_data = (int*)malloc(size * sizeof(int));

    return p_data;
}

void fill_array(int* p_array, int size)
{
    for (int k = 0; k < size; ++k)
    {
        printf("Enter a value: ");
        scanf("%d", &(p_array[k]));
    }
}

int main()
{
    _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);

    int* first_array = create_array(5);
    fill_array(first_array, 5);

    int* second_array = create_array(3);
    fill_array(second_array, 3);

    int* third_array = create_array(3);
    fill_array(third_array, 3);

    return 0;
}

```

The program outputs:

```

The thread 0x1d40 has exited with code 0 (0x0).
The thread 0x2740 has exited with code 0 (0x0).
The thread 0x2cc0 has exited with code 0 (0x0).
Detected memory leaks!
Dumping objects ->
{80} normal block at 0x0030A030, 12 bytes long.
  Data: <     A      y      > 8F 0C 00 00 41 01 00 00 79 11 05 00
{79} normal block at 0x0030A728, 12 bytes long.
  Data: <I     A      > 49 E9 04 00 41 01 00 00 8B 02 00 00
{76} normal block at 0x00309C58, 20 bytes long.
  Data: <>     A      > 3E FD 09 00 41 00 00 00 8B 02 00 00 E7 00 00 00
Object dump complete.
The program '[4448] Exercise 11.exe' has exited with code 0 (0x0).

```

Figure 347: Detect the Memory Leak.

9.3.12 Exercise 12

Given the following code, the user should detect and fix memory leak.

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

int* create_array(int size)
{
    int* p_data = (int*)malloc(size * sizeof(int));

    return p_data;
}

void fill_array(int* p_array, int size)
{
    for (int k = 0; k < size; ++k)
    {
        printf("Enter a value: ");
        scanf("%d", &(p_array[k]));
    }
}

int main()
{
    int* first_array = create_array(5);
    fill_array(first_array, 5);

    int* second_array = create_array(3);
    fill_array(second_array, 3);

    int* third_array = create_array(3);
    fill_array(third_array, 3);

    return 0;
}
```

To detect memory leak, I wrote the following code:

```
#include <stdio.h>
#include <stdlib.h>
#include <crtdbg.h>

int* create_array(int size)
{
```

```

int* p_data = (int*)malloc(size * sizeof(int));

return p_data;
}

void fill_array(int* p_array, int size)
{
    for (int k = 0; k < size; ++k)
    {
        printf("Enter a value: ");
        scanf("%d", &(p_array[k]));
    }
}

int main()
{
    _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);

    int* first_array = create_array(5);
    fill_array(first_array, 5);

    int* second_array = create_array(3);
    fill_array(second_array, 3);

    int* third_array = create_array(3);
    fill_array(third_array, 3);

    return 0;
}

```

This was Microsoft Visual Studio's Output:

```

The thread 0x2cc8 has exited with code 0 (0x0).
The thread 0x1c88 has exited with code 0 (0x0).
The thread 0xe48 has exited with code 0 (0x0).
Detected memory leaks!
Dumping objects ->
{80} normal block at 0x00E825B0, 12 bytes long.
  Data: <          > 01 00 00 00 01 00 00 00 01 00 00 00
{79} normal block at 0x00E82540, 12 bytes long.
  Data: <          > 01 00 00 00 01 00 00 00 01 00 00 00
{76} normal block at 0x00E821E8, 20 bytes long.
  Data: <          > 01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
Object dump complete.
The program '[7604] Exercise 12.exe' has exited with code 0 (0x0).

```

Figure 348: Fix the Memory Leaks Output.

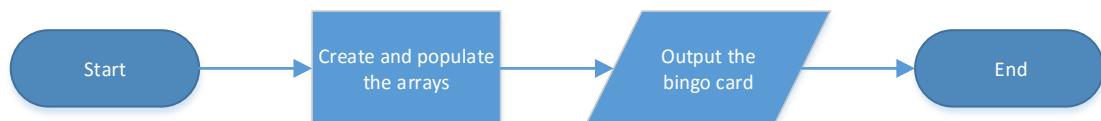
9.3.13 Exercise 13

This program should generate random Bingo Boards, as in the example:

	B	I	N	G	O
11	18	45	49	70	
5	16	32	47	75	
7	27	42	59	61	
12	22	44	51	65	
1	30	37	60	69	

Figure 349: Bingo Board.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following program:

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
    srand(time(0));

    int bingo[5][5] = { 0 };

    int temp = 0;
  
```

```

for (int row = 0; row < 5; row++)
{
    for (int column = 0; column < 5; column++)
    {
        if (column == 0)
        {
            do
            {
                temp = (rand() % 14) + 1;
            } while (temp == bingo[0][column] || temp ==
bingo[1][column] || temp == bingo[2][column] || temp ==
bingo[3][column] || temp == bingo[4][column]);

                bingo[row][column] = temp;
        }
        else if (column == 1)
        {
            do
            {
                temp = (rand() % 14) + 16;
            } while (temp == bingo[0][column] || temp ==
bingo[1][column] || temp == bingo[2][column] || temp ==
bingo[3][column] || temp == bingo[4][column]);

                bingo[row][column] = temp;
        }
        else if (column == 2)
        {
            do
            {
                temp = (rand() % 14) + 31;
            } while (temp == bingo[0][column] || temp ==
bingo[1][column] || temp == bingo[2][column] || temp ==
bingo[3][column] || temp == bingo[4][column]);

                bingo[row][column] = temp;
        }
        else if (column == 3)
        {
            do
            {
                temp = (rand() % 14) + 46;
            } while (temp == bingo[0][column] || temp ==
bingo[1][column] || temp == bingo[2][column] || temp ==
bingo[3][column] || temp == bingo[4][column]);
        }
    }
}

```

```

        bingo[row][column] = temp;
    }
else
{
    do
    {
        temp = (rand() % 14) + 61;
    } while (temp == bingo[0][column] || temp ==
bingo[1][column] || temp == bingo[2][column] || temp ==
bingo[3][column] || temp == bingo[4][column]);

    bingo[row][column] = temp;
}
}

printf("-----+-----+-----+-----+-----+\n");
printf(" |--B--|--I--|--N--|--G--|--O--| \n");
printf("-----+-----+-----+-----+-----+\n");
printf(" | %d | %d | %d | %d | %d | \n", bingo[0][0],
bingo[0][1], bingo[0][2], bingo[0][3], bingo[0][4]);
printf("-----+-----+-----+-----+-----+\n");
printf(" | %d | %d | %d | %d | %d | \n", bingo[1][0],
bingo[1][1], bingo[1][2], bingo[1][3], bingo[1][4]);
printf("-----+-----+-----+-----+-----+\n");
printf(" | %d | %d | %d | %d | %d | \n", bingo[2][0],
bingo[2][1], bingo[2][2], bingo[2][3], bingo[2][4]);
printf("-----+-----+-----+-----+-----+\n");
printf(" | %d | %d | %d | %d | %d | \n", bingo[3][0],
bingo[3][1], bingo[3][2], bingo[3][3], bingo[3][4]);
printf("-----+-----+-----+-----+-----+\n");
printf(" | %d | %d | %d | %d | %d | \n", bingo[4][0],
bingo[4][1], bingo[4][2], bingo[4][3], bingo[4][4]);
printf("-----+-----+-----+-----+-----+\n");

printf("\n\n");

return 0;
}

```

This program outputs:

```
C:\Windows\system32\cmd.exe
+---+---+---+---+
|--B--|--I--|--N--|--G--|--O--|
+=====+=====+=====+=====+=====
|   11  |   25  |   32  |   54  |   63  |
+-----+-----+-----+-----+
|   12  |   23  |   33  |   55  |   68  |
+-----+-----+-----+-----+
|    4  |   21  |   37  |   50  |   64  |
+-----+-----+-----+-----+
|    5  |   19  |   42  |   48  |   72  |
+-----+-----+-----+-----+
|   10  |   16  |   34  |   47  |   62  |
+-----+-----+-----+-----+
Pressione qualquer tecla para continuar. . .
```

Figure 350: Bingo Board Output.

9.3.14 Exercise 14

This program allows the user to play a Hangman game. The game should have 10 secret words and select one of them randomly. If the letter is in the text, reveal the letter in the text, otherwise, reveal another part of the hangman. If the hangman is fully drawn, the player loses, if the word is completed, the player wins.

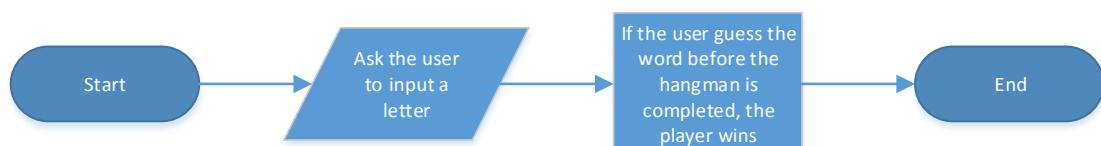
The program should output:

A screenshot of a terminal window on a black background. At the top, there is a partially drawn hangman figure with a head (circle), a body (vertical line), and two arms (one \, one /). Below the figure is a dashed rectangular frame. In the center of the frame, the word "SECRET WORD" is displayed in all caps. To its right, a dash-separated string of letters is shown: "- R O - R - M M - - -". Below this, the text "INCORRECT GUESSES: Z X C V B N L Q" is displayed. At the bottom, the prompt "NEXT GUESS?" is visible.

```
SECRET WORD: - R O - R - M M - - -
INCORRECT GUESSES: Z X C V B N L Q
NEXT GUESS?
```

Figure 351: Hangman.

To develop this program, I wrote the following flowchart:



10 Ten

10.1 Lectures

Structures are groups of several pieces of information together, they can store complex pieces of data and a collection of different variables under a single name. The declaration can be compared with a blueprint, it tells the compiler what this new type of data is. Operators allows the user to access members of a structure via pointer.

10.2 Examples

10.2.1 Example 1

This program examples how assert works.

This was the written code:

```
#include <stdio.h>
#include <assert.h>

void print_grade(char grade)
{
    assert(grade == 'A' ||
           grade == 'B' ||
           grade == 'C' ||
           grade == 'D');

    printf("Grade: %c\n", grade);
}

int main()
{
    print_grade('D');

    print_grade('C');

    print_grade('x');

    print_grade('B');

    print_grade('A');

    return 0;
}
```

10.2.2 Example 2

This program shows how to declare and examples an use for structures.

This was the written code:

```
#include <stdio.h>

struct Car
{
    char manufacturer[20];
    char model[20];
    int year;
    float price;
};

void print_car(struct Car a_car)
{
    printf("Manufacturer: %s\n", a_car.manufacturer);
    printf("Model: %s\n", a_car.model);
    printf("Year: %d\n", a_car.year);
    printf("Price: %0.2f\n", a_car.price);
    printf("\n");
}

int main()
{
    struct Car nissan;

    nissan.year = 2008;
    nissan.price = 38070.0;
    sprintf(nissan.manufacturer, "Nissan");
    sprintf(nissan.model, "Fairlady Z Z33");

    struct Car mazda;

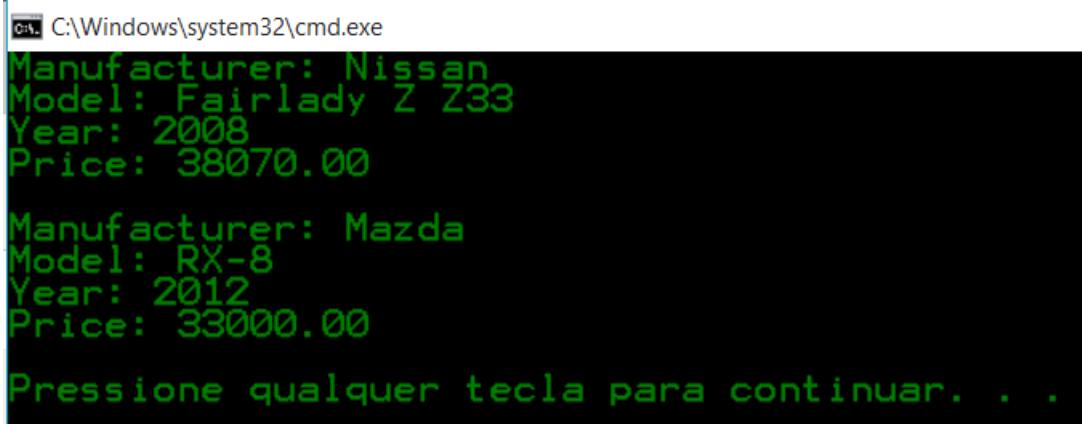
    mazda.year = 2012;
    mazda.price = 33000.0;
    sprintf(mazda.manufacturer, "Mazda");
    sprintf(mazda.model, "RX-8");

    print_car(nissan);

    print_car(mazda);
```

```
    return 0;  
}
```

This program outputs:



```
C:\Windows\system32\cmd.exe  
Manufacturer: Nissan  
Model: Fairlady Z Z33  
Year: 2008  
Price: 38070.00  
  
Manufacturer: Mazda  
Model: RX-8  
Year: 2012  
Price: 33000.00  
  
Pressione qualquer tecla para continuar. . .
```

Figure 352: Declaring a Structure.

10.2.3 Example 3

This program shows how to pass structures by reference.

This was the written code:

```
#include <stdio.h>
#include <assert.h>

struct Car
{
    char manufacturer[20];
    char model[20];
    int year;
    float price;
};

void print_car(struct Car* p_car)
{
    assert(p_car);

    printf("Manufacturer: %s\n", p_car->manufacturer);
    printf("Model: %s\n", p_car->model);
    printf("Year: %d\n", p_car->year);
    printf("Price: %.2f\n", p_car->price);
    printf("\n");
}

int main()
{
    struct Car nissan;

    nissan.year = 2008;
    nissan.price = 38070.0;
    sprintf(nissan.manufacturer, "Nissan");
    sprintf(nissan.model, "Fairlady Z Z33");

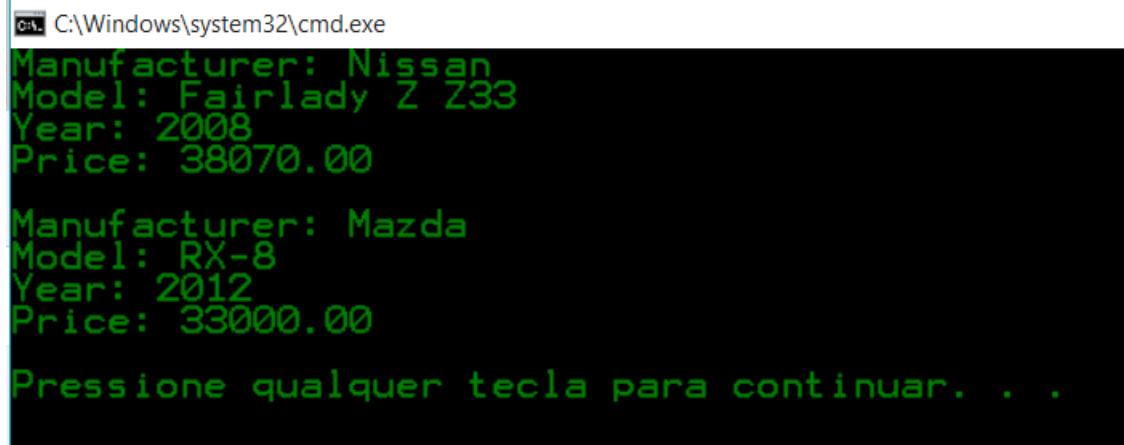
    struct Car mazda;

    mazda.year = 2012;
    mazda.price = 33000.0;
    sprintf(mazda.manufacturer, "Mazda");
    sprintf(mazda.model, "RX-8");

    print_car(&nissan);
```

```
    print_car(&mazda);  
  
    return 0;  
}
```

This program outputs:



The screenshot shows a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The output of the program is displayed, showing two sets of car details. The first set is for a Nissan Fairlady Z Z33 from 2008 at \$38,070.00. The second set is for a Mazda RX-8 from 2012 at \$33,000.00. A message at the bottom prompts the user to press any key to continue.

```
C:\Windows\system32\cmd.exe  
Manufacturer: Nissan  
Model: Fairlady Z Z33  
Year: 2008  
Price: 38070.00  
  
Manufacturer: Mazda  
Model: RX-8  
Year: 2012  
Price: 33000.00  
  
Pressione qualquer tecla para continuar. . .
```

Figure 353: Passing a Structure by Reference.

10.2.4 Example 4

This program shows how to store a structure on the heap.

This was the written code:

```
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>

struct Car
{
    char manufacturer[20];
    char model[20];
    int year;
    float price;
};

void print_car(struct Car* p_car)
{
    assert(p_car);

    printf("Manufacturer: %s\n", p_car->manufacturer);
    printf("Model: %s\n", p_car->model);
    printf("Year: %d\n", p_car->year);
    printf("Price: %0.2f\n", p_car->price);
    printf("\n");
}

int main()
{
    struct Car* p_nissan = 0;
    struct Car* p_mazda = 0;

    p_nissan = malloc(sizeof(struct Car));
    assert(p_nissan);

    p_nissan->year = 2008;
    p_nissan->price = 38070.0;
    sprintf(p_nissan->manufacturer, "Nissan");
    sprintf(p_nissan->model, "Fairlady Z Z33");

    p_mazda = malloc(sizeof(struct Car));
    assert(p_mazda);
```

```
p_mazda->year = 2012;
p_mazda->price = 33000.0;
sprintf(p_mazda->manufacturer, "Mazda");
sprintf(p_mazda->model, "RX-8");

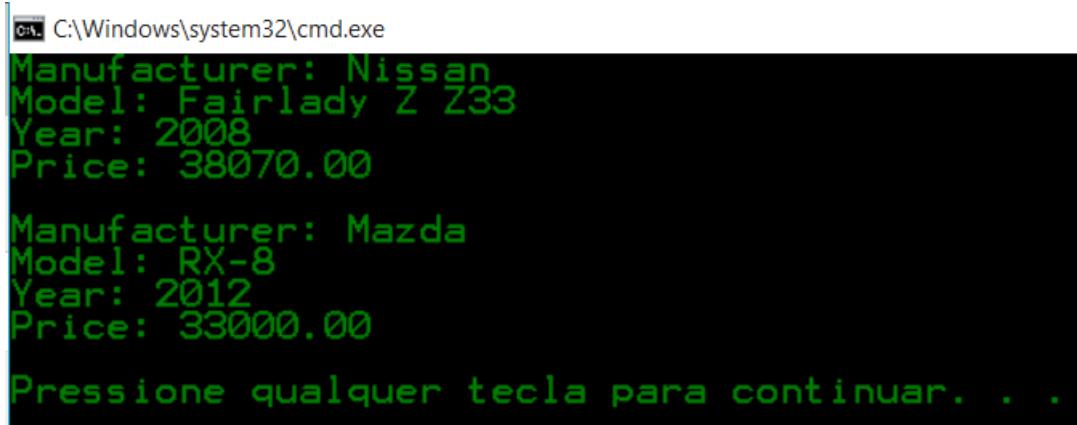
print_car(p_nissan);
print_car(p_mazda);

free(p_nissan);
p_nissan = 0;

free(p_mazda);
p_mazda = 0;

return 0;
}
```

This program outputs:



```
C:\Windows\system32\cmd.exe
Manufacturer: Nissan
Model: Fairlady Z Z33
Year: 2008
Price: 38070.00

Manufacturer: Mazda
Model: RX-8
Year: 2012
Price: 33000.00

Pressione qualquer tecla para continuar. . .
```

Figure 354: Storing a structure on the Heap.

10.3 Exercises

10.3.1 Exercise 1

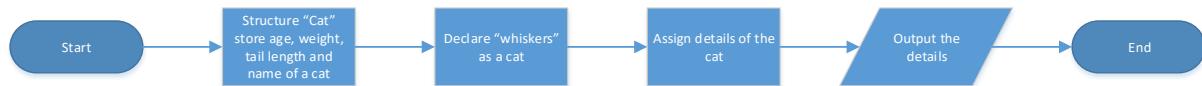
This program has a “Cat” structure with details of a cat, a Cat is declared in the main and printed using a function. The program should output:

A cat...

Name: ?
 Age: ?
 Weight: ?
 Tail Length: ?

Figure 355: The Cat Structure.

To develop this program, I draw the following flowchart:



This was the written code:

```

#include <stdio.h>

struct Cat
{
    int age;
    int weight;
    int tail_length;
    char name[100];
};

int main()
{
    struct Cat whiskers;
    whiskers.age = 7;
  
```

```
whiskers.weight = 8;
whiskers.tail_length = 15;
sprintf(whiskers.name, "Whiskers");

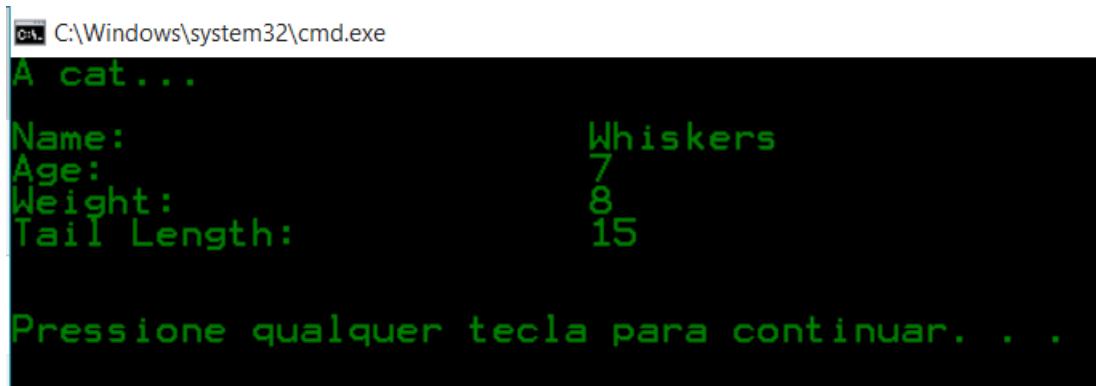
printf("A cat...") ;
printf("\n\n") ;

printf("Name:           %s\n", whiskers.name);
printf("Age:            %d\n", whiskers.age);
printf("Weight:          %d\n", whiskers.weight);
printf("Tail Length:     %d\n", whiskers.tail_length);

printf("\n\n") ;

return 0;
}
```

This program outputs:



```
C:\Windows\system32\cmd.exe
A cat...
Name:           Whiskers
Age:            7
Weight:          8
Tail Length:     15

Pressione qualquer tecla para continuar. . .
```

Figure 356: The Cat Structure Output.

10.3.2 Exercise 2

Given the following structure:

```
struct Person
{
    char first_initial;
    char last_initial;
    int age;
    int weight;
    char sex;
    int height;
    char blood_type;
    int shoe_type;
};
```

The program should output the size of the structure, and another structure with the same variables should be created to have a smaller size.

This was the written code:

```
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>

struct Person
{
    char first_initial;
    char last_initial;
    int age;
    int weight;
    char sex;
    int height;
    char blood_type;
    int shoe_type;
};

struct OptimisedPerson
{
    int age;
    int weight;
    int height;
    int shoe_type;
    char first_initial;
```

```
char last_initial;
char sex;
char blood_type;
};

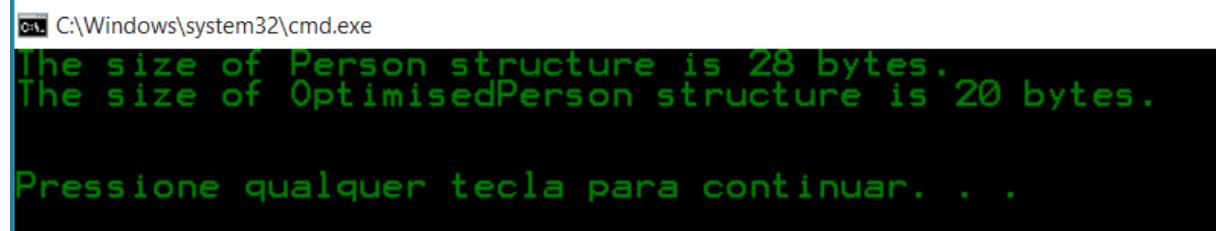
int main()
{
    struct Person a_person;
    printf("The size of Person structure is %d bytes.\n",
sizeof(a_person));

    struct OptimisedPerson another_person;
    printf("The size of OptimisedPerson structure is %d bytes.\n",
sizeof(another_person));

    printf("\n\n");

    return 0;
}
```

This program outputs:



C:\Windows\system32\cmd.exe

```
The size of Person structure is 28 bytes.
The size of OptimisedPerson structure is 20 bytes.

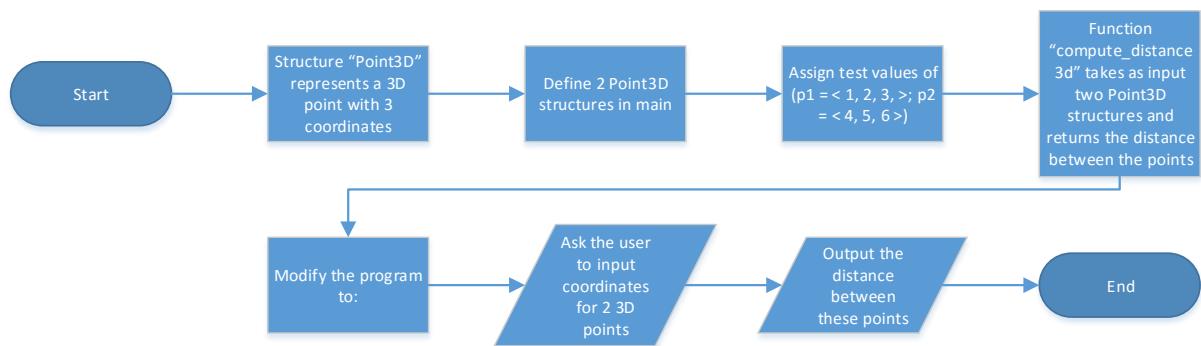
Pressione qualquer tecla para continuar. . .
```

Figure 357: sizeof Structures.

10.3.3 Exercise 3

This program has a structure which represents a 3D point and 2 of these points are declared in the main, a function computes the distance between these points.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>
#include <math.h>

double compute_distance3d(struct Point3D* p1, struct Point3D* p2);

struct Point3D
{
    double x;
    double y;
    double z;
};

double compute_distance3d(struct Point3D* p1, struct Point3D* p2)
{
    return (sqrt((pow(((p2->x) - (p1->x)), (2))) + (pow(((p2->y) - (p1->y)), (2))) + (pow(((p2->z) - (p1->z)), (2)))));
}

int main()
{
    struct Point3D p1;
    
```

```
p1.x = 1;
p1.y = 2;
p1.z = 3;

struct Point3D p2;
p2.x = 4;
p2.y = 5;
p2.z = 6;

double distance3d = compute_distance3d(&p1, &p2);

printf("The 3d distance between p1(%f, %f, %f) and p2 (%f, %f,
%f) is: %f.", p1.x, p1.y, p1.z, p2.x, p2.y, p2.z, distance3d);

printf("\n\n");

printf("Enter data for 2 3D points:\n");

printf("p1 x: ");
scanf("%lf", &p1.x);
printf("p1 y: ");
scanf("%lf", &p1.y);
printf("p1 z: ");
scanf("%lf", &p1.z);

printf("\n");

printf("p2 x: ");
scanf("%lf", &p2.x);
printf("p2 y: ");
scanf("%lf", &p2.y);
printf("p2 z: ");
scanf("%lf", &p2.z);

printf("\n");

distance3d = compute_distance3d(&p1, &p2);

printf("The 3d distance between p1(%f, %f, %f) and p2 (%f, %f,
%f) is: %f.", p1.x, p1.y, p1.z, p2.x, p2.y, p2.z, distance3d);

printf("\n\n");

return 0;
}
```

This program outputs:

```
C:\Windows\system32\cmd.exe
The 3d distance between p1(1.000000, 2.000000, 3.000000) and p2 (4.000000, 5.000000, 6.000000) is: 5.196152.
Enter data for 2 3D points:
p1 x: 5
p1 y: 9
p1 z: 8
p2 x: 1
p2 y: 2
p2 z: 5
The 3d distance between p1(5.000000, 9.000000, 8.000000) and p2 (1.000000, 2.000000, 5.000000) is: 8.602325.
Pressione qualquer tecla para continuar. . .
```

Figure 358: Distance 3D Output.

10.3.4 Exercise 4

This program has a structure with two numbers and functions to add, subtract, multiply, divide and print fractions. In the main, functions are declared.

This was the written code:

```
#include <stdio.h>

struct Fraction add_fractions(struct Fraction* f1, struct Fraction* f2);
struct Fraction subtract_fractions(struct Fraction* f1, struct Fraction* f2);
struct Fraction multiply_fractions(struct Fraction* f1, struct Fraction* f2);
struct Fraction divide_fractions(struct Fraction* f1, struct Fraction* f2);
void print_fraction(struct Fraction* f1);

struct Fraction
{
    int numerator;
    int denominator;
};

struct Fraction add_fractions(struct Fraction* f1, struct Fraction* f2)
{
    struct Fraction f3;
    f3.numerator = (((f1->numerator) * (f2->denominator)) + ((f2->numerator) * (f1->denominator)));
    f3.denominator = ((f1->denominator) * (f2->denominator));
    return f3;
}

struct Fraction subtract_fractions(struct Fraction* f1, struct Fraction* f2)
{
    struct Fraction f3;
    f3.numerator = (((f1->numerator) * (f2->denominator)) - ((f2->numerator) * (f1->denominator)));
    f3.denominator = ((f1->denominator) * (f2->denominator));
}
```

```
    return f3;
}

struct Fraction multiply_fractions(struct Fraction* f1, struct
Fraction* f2)
{
    struct Fraction f3;
    f3.numerator = ((f1->numerator)*(f2->numerator));
    f3.denominator = ((f1->denominator)*(f2->denominator));

    return f3;
}

struct Fraction divide_fractions(struct Fraction* f1, struct
Fraction* f2)
{
    struct Fraction f3;
    f3.numerator = ((f1->numerator)*(f2->denominator));
    f3.denominator = ((f1->denominator)*(f2->numerator));

    return f3;
}

void print_fraction(struct Fraction* f1)
{
    printf("%d/%d", f1->numerator, f1->denominator);
}

int main()
{
    struct Fraction f1;
    f1.numerator = 2;
    f1.denominator = 3;

    struct Fraction f2;
    f2.numerator = 5;
    f2.denominator = 8;

    printf("Function f1:\n");
    print_fraction(&f1);
    printf("\n\n");

    printf("Function f2:\n");
    print_fraction(&f2);
    printf("\n\n");

    printf("f1 + f2:\n");
}
```

```
struct Fraction f3 = add_fractions(&f1, &f2);
print_fraction(&f3);
printf("\n\n");

printf("f1 - f2:\n");
f3 = subtract_fractions(&f1, &f2);
print_fraction(&f3);
printf("\n\n");

printf("f1 * f2:\n");
f3 = multiply_fractions(&f1, &f2);
print_fraction(&f3);
printf("\n\n");

printf("f1 / f2:\n");
f3 = divide_fractions(&f1, &f2);
print_fraction(&f3);
printf("\n\n");

printf("\n\n");

return 0;
}
```

This program outputs:

```
C:\Windows\system32\cmd.exe
Function f1:
2/3

Function f2:
5/8

f1 + f2:
31/24

f1 - f2:
1/24

f1 * f2:
10/24

f1 / f2:
16/15

Pressione qualquer tecla para continuar. . .
```

Figure 359: Fractions Output.

10.3.5 Exercise 5

Given the structure:

```
struct Thing
{
    int a;
    short b;
    char c;
    double d;
    long e;
    float f;
};
```

A struct Thing variable is declared in the main, the address of this variable is printed, then the variables values of the Thing variable and its respective addresses are printed in the main and in two functions, one passing by reference, and another passing by value.

This was the written code:

```
#include <stdio.h>

void pass_by_value(struct Thing thing);
void pass_by_reference(struct Thing* thing);

struct Thing
{
    int a;
    short b;
    char c;
    double d;
    long e;
    float f;
};

void pass_by_value(struct Thing thing)
{
    printf("Address of thing: %p\n", &thing);
    printf("thing.a holds: '%d' and it's stored at '%p'\n",
    thing.a, &thing.a);
```

```
    printf("thing.b holds: '%hi' and it's stored at '%p'\n",
thing.b, &thing.b);
    printf("thing.c holds: '%c' and it's stored at '%p'\n",
thing.c, &thing.c);
    printf("thing.d holds: '%lf' and it's stored at '%p'\n",
thing.d, &thing.d);
    printf("thing.e holds: '%li' and it's stored at '%p'\n",
thing.e, &thing.e);
    printf("thing.f holds: '%f' and it's stored at '%p'\n",
thing.f, &thing.f);
}

void pass_by_reference(struct Thing* thing)
{
    printf("Address of thing: %p\n", thing);
    printf("thing.a holds: '%d' and it's stored at '%p'\n", thing-
>a, &thing->a);
    printf("thing.b holds: '%hi' and it's stored at '%p'\n", thing-
>b, &thing->b);
    printf("thing.c holds: '%c' and it's stored at '%p'\n", thing-
>c, &thing->c);
    printf("thing.d holds: '%lf' and it's stored at '%p'\n", thing-
>d, &thing->d);
    printf("thing.e holds: '%li' and it's stored at '%p'\n", thing-
>e, &thing->e);
    printf("thing.f holds: '%f' and it's stored at '%p'\n", thing-
>f, &thing->f);
}

int main()
{
    struct Thing thing;
    thing.a = 1;
    thing.b = 5;
    thing.c = 'a';
    thing.d = 3.095;
    thing.e = 5;
    thing.f = 7.124;

    printf("Address of thing: %p\n", &thing);
    printf("thing.a holds: '%d' and it's stored at '%p'\n",
thing.a, &thing.a);
    printf("thing.b holds: '%hi' and it's stored at '%p'\n",
thing.b, &thing.b);
    printf("thing.c holds: '%c' and it's stored at '%p'\n",
thing.c, &thing.c);
```

```
    printf("thing.d holds: '%lf' and it's stored at '%p'\n",
thing.d, &thing.d);
    printf("thing.e holds: '%li' and it's stored at '%p'\n",
thing.e, &thing.e);
    printf("thing.f holds: '%f' and it's stored at '%p'\n",
thing.f, &thing.f);

printf("\n");

printf("Calling pass_by_value:\n");
pass_by_value(thing);

printf("\n");

printf("Calling pass_by_reference:\n");
pass_by_reference(&thing);

printf("\n\n");

return 0;
}
```

This program outputs:

```
C:\Windows\system32\cmd.exe
Address of thing: 00FFFE74
thing.a holds: '1' and it's stored at '00FFFE74'
thing.b holds: '5' and it's stored at '00FFFE78'
thing.c holds: 'a' and it's stored at '00FFFE7A'
thing.d holds: '3;095000' and it's stored at '00FFFE7C'
thing.e holds: '5' and it's stored at '00FFFE84'
thing.f holds: '7.124000' and it's stored at '00FFFE88'

Calling pass_by_value:
Address of thing: 00FFFD8C
thing.a holds: '1' and it's stored at '00FFFD8C'
thing.b holds: '5' and it's stored at '00FFFD90'
thing.c holds: 'a' and it's stored at '00FFFD92'
thing.d holds: '3;095000' and it's stored at '00FFFD94'
thing.e holds: '5' and it's stored at '00FFFD9C'
thing.f holds: '7.124000' and it's stored at '00FFFDA0'

Calling pass_by_reference:
Address of thing: 00FFFE74
thing.a holds: '1' and it's stored at '00FFFE74'
thing.b holds: '5' and it's stored at '00FFFE78'
thing.c holds: 'a' and it's stored at '00FFFE7A'
thing.d holds: '3;095000' and it's stored at '00FFFE7C'
thing.e holds: '5' and it's stored at '00FFFE84'
thing.f holds: '7.124000' and it's stored at '00FFFE88'

Pressione qualquer tecla para continuar. . .
```

Figure 360: Pass by Value vs Pass by Reference, with Structures Output.

10.3.6 Exercise 6

This program has a structure that describes a game. Three game structures are declared in the main and have their details printed with a function.

This was the written code:

```
#include <stdio.h>

void print_video_game_details(struct Video_Game* game1);

struct Video_Game
{
    int in_app_purchase;
    int age_limit;
    int year_release;
    float price;
    char title[100];
    char genre[100];
    char developer[100];
    char platform[200];
};

void print_video_game_details(struct Video_Game* game1)
{
    printf("Title: %s\n", game1->title);
    printf("Genre: %s\n", game1->genre);
    printf("Developer: %s\n", game1->developer);
    printf("Platform: %s\n", game1->platform);
    printf("Price: %f\n", game1->price);
    printf("Lower Age Limit: %d\n", game1->age_limit);
    printf("Year of release: %d\n", game1->year_release);
    printf("In-app Purchase: ");
    if (game1->in_app_purchase) printf("Yes");
    else printf("No");

    printf("\n\n");
}

int main()
{
```

```
struct Video_Game game1;
sprintf(game1.title, "The Elder Scrolls V: Skyrim");
sprintf(game1.genre, "Action RPG");
sprintf(game1.developer, "Bethesda Studios");
sprintf(game1.platform, "PlayStation 3, Xbox 360, Microsoft Windows");
game1.price = 23.99;
game1.age_limit = 13;
game1.year_release = 2011;
game1.in_app_purchase = 0;

struct Video_Game game2;
sprintf(game2.title, "Fallout 4");
sprintf(game2.genre, "Action RPG");
sprintf(game2.developer, "Bethesda Studios");
sprintf(game2.platform, "PlayStation 4, Xbox One, Microsoft Windows");
game2.price = 99.95;
game2.age_limit = 18;
game2.year_release = 2015;
game2.in_app_purchase = 0;

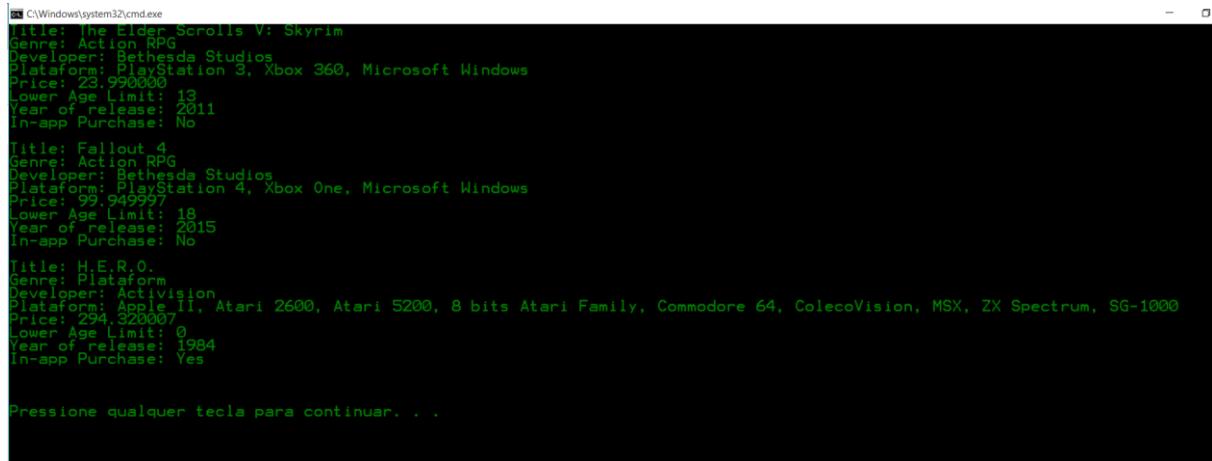
struct Video_Game game3;
sprintf(game3.title, "H.E.R.O.");
sprintf(game3.genre, "Platform");
sprintf(game3.developer, "Activision");
sprintf(game3.platform, "Apple II, Atari 2600, Atari 5200, 8 bits Atari Family, Commodore 64, ColecoVision, MSX, ZX Spectrum, SG-1000");
game3.price = 294.32;
game3.age_limit = 0;
game3.year_release = 1984;
game3.in_app_purchase = 1;

print_video_game_details(&game1);
print_video_game_details(&game2);
print_video_game_details(&game3);

printf("\n\n");

return 0;
}
```

This program outputs:



C:\Windows\System32\cmd.exe

```
Title: The Elder Scrolls V: Skyrim
Genre: Action RPG
Developer: Bethesda Studios
Platform: PlayStation 3, Xbox 360, Microsoft Windows
Price: 23.990000
Lower Age Limit: 13
Year of release: 2011
In-app Purchase: No

Title: Fallout 4
Genre: Action RPG
Developer: Bethesda Studios
Platform: PlayStation 4, Xbox One, Microsoft Windows
Price: 99.949997
Lower Age Limit: 18
Year of release: 2015
In-app Purchase: No

Title: H.E.R.O.
Genre: Platform
Developer: Activision
Platform: Apple II, Atari 2600, Atari 5200, 8 bits Atari Family, Commodore 64, ColecoVision, MSX, ZX Spectrum, SG-1000
Price: 294.320007
Lower Age Limit: 0
Year of release: 1984
In-app Purchase: Yes

Pressione qualquer tecla para continuar. . .
```

Figure 361: The Video_Game Structure Output.

10.3.7 Exercise 7

This program has a structure with details of a card, and a function that prints this card. Another function generates cards randomly.

This was the written code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <assert.h>

struct Playing_Card card_factory();
void print_playing_card(struct Playing_Card card);

struct Playing_Card
{
    int rank;
    int suit;
};

struct Playing_Card card_factory()
{
    srand(time(0));

    struct Playing_Card card;
    card.rank = (rand() % 13) + 1;
    card.suit = (rand() % 3);

    //assert(card);

    return card;
}

void print_playing_card(struct Playing_Card card)
{
    printf("Printing playing card:\n");
    printf("Rank: ");
        if (card.rank == 1) printf("Ace");
        else if (card.rank == 11) printf("Jack");
        else if (card.rank == 12) printf("Queen");
        else if (card.rank == 13) printf("King");
        else printf("%d", card.rank);
    printf("\n");
    printf("Suit: ");
}
```

```
    if (card.suit == 0) printf("Diamonds");
    else if (card.suit == 1) printf("Hearts");
    else if (card.suit == 2) printf("Clubs");
    else printf("Spades");
    printf("\n\n");
}

int main()
{
    struct Playing_Card card[5];

    card[0] = card_factory();
    card[1] = card_factory();
    card[2] = card_factory();
    card[3] = card_factory();
    card[4] = card_factory();

    print_playing_card(card[0]);
    print_playing_card(card[1]);
    print_playing_card(card[2]);
    print_playing_card(card[3]);
    print_playing_card(card[4]);

    printf("\n\n");

    return 0;
}
```

This program outputs:

```
C:\Windows\system32\cmd.exe
Printing playing card:
Rank: 8
Suit: Hearts

Pressione qualquer tecla para continuar. . .
```

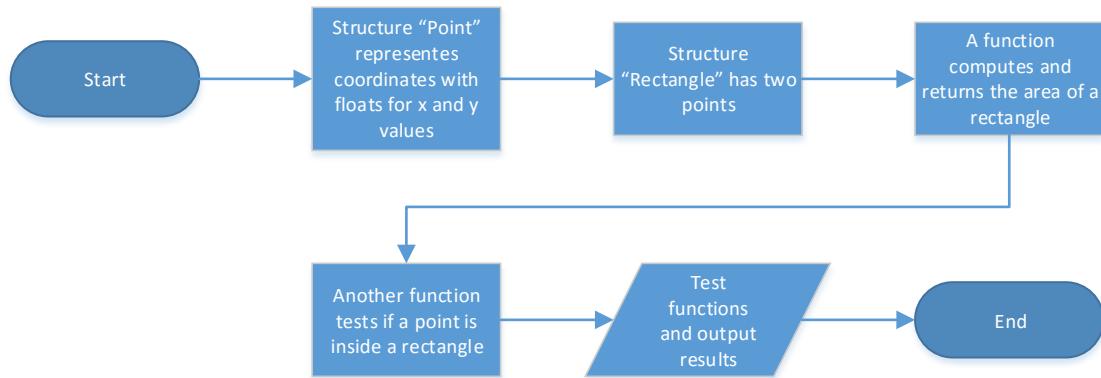
Figure 362: Playing Cards Factory Output.

This program has a problem, it always generates the same cards.

10.3.8 Exercise 8

This program has a structure for a 2D point, a rectangle structure with two 2D points, a function to calculate the rectangle's area, and another function to return either or not a point is inside a rectangle.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

float rectangle_area(struct Rectangle* rec);
int is_inside(struct Rectangle* rec, struct Point* point);

struct Point
{
    float x;
    float y;
};

struct Rectangle
{
    struct Point bottom_left;
    struct Point top_right;
};

float rectangle_area(struct Rectangle* rec)
  
```

```

{
    return ((rec->top_right.x - rec->bottom_left.x) * (rec-
>top_right.y - rec->bottom_left.y));
}

int is_inside(struct Rectangle* rec, struct Point* point)
{
    if ((point->x >= rec->bottom_left.x && point->x <= rec-
>top_right.x) && (point->y >= rec->bottom_left.y && point->y <= rec-
>top_right.y)) return 1;
    else return 0;
}

int main()
{
    struct Point bottom_left;
    bottom_left.x = 0;
    bottom_left.y = 0;
    struct Point top_right;
    top_right.x = 7.124;
    top_right.y = 7.124;
    struct Point inside;
    inside.x = 5.123;
    inside.y = 6.235;
    struct Point outside;
    outside.x = 3.1235;
    outside.y = 12.1245;
    struct Rectangle rec;
    rec.bottom_left = bottom_left;
    rec.top_right = top_right;

    float area = rectangle_area(&rec);
    printf("The area of the rectangle is: %f", area);
    printf("\n\n");

    int point_inside = is_inside(&rec, &inside);
    printf("The point (%f, %f) is ", inside.x, inside.y);
    if (point_inside == 1);
    else if (point_inside == 0)
        printf("not ");
    printf("inside the rectangle");
    printf("\n\n");

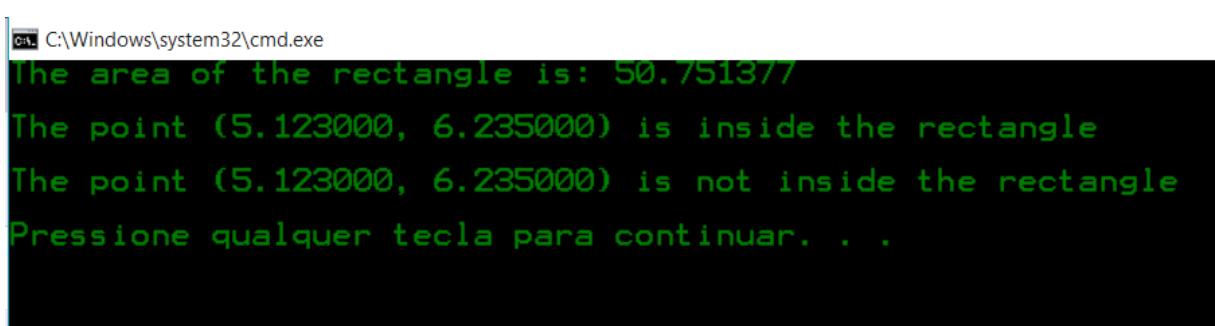
    int point_outside = is_inside(&rec, &point_outside);
    printf("The point (%f, %f) is ", inside.x, inside.y);
    if (point_outside == 1);
    else if (point_outside == 0)

```

```
    printf("not ");
    printf("inside the rectangle");

    printf("\n\n");
}

This program outputs:
```



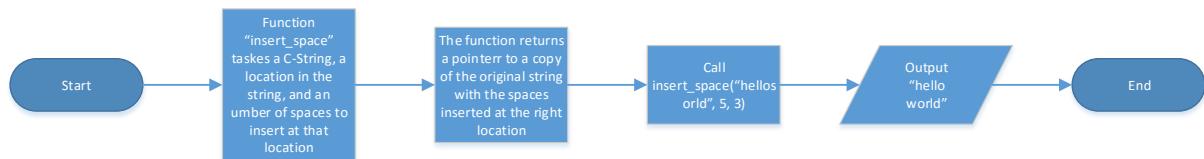
```
C:\Windows\system32\cmd.exe
The area of the rectangle is: 50.751377
The point (5.123000, 6.235000) is inside the rectangle
The point (5.123000, 6.235000) is not inside the rectangle
Pressione qualquer tecla para continuar. . .
```

Figure 363: Rectangle Structure Output.

10.3.9 Exercise 9

This program has a function that takes a string and two integers as parameter, the function should insert a number of spaces in a determinated position in the string.

To develop this program, I draw the following flowchart:



Using this flowchart, I wrote the following code:

```

#include <stdio.h>

void insert_space(char* string, int place, int spaces);

void insert_space(char* string, int place, int spaces)
{
    char temporary[100] = string;

    for (int i = place; i < place + spaces; i++)
    {
        string[i] = ' ';
    }

    for (int i = place + spaces + 1; temporary[i] != '\n'; i++)
    {
        string[i] = temporary[i - place - spaces - 1];
    }
}

int main()
{
    char string[] = "helloworld";
    insert_space(string, 5, 3);
}

```

```
printf("%s", string);

printf("\n\n");

return 0;
}
```

But this code does not work properly. It does not compile and I could not find the reason, I tried to debug it, but I was unsuccessful on finding the error. I also tried cleaning the solution and restarting my computer, but it did not work as well.

11 Eleven

11.1 Lectures

This week we saw how function pointers work. We also saw how to use preprocessors and their multiple uses. They can be used to declare header files and to call multiple constructors.

On Friday we learned how Application Programming Interface (API) works, they are very useful to create programs with an interface.

11.2 Examples

11.2.1 Example 1

This example simply shows how to save text into a file.

This was the written code:

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

void save_text_file(char* filename)
{
    FILE* fp = fopen(filename, "w");

    fprintf(fp, "Saving this text to file...\n");

    fclose(fp);
}

char* load_text_file(char* filename)
{
    char* buffer = malloc(sizeof(char) * 80);

    FILE* fp = fopen(filename, "r");

    fscanf(fp, "%79[a-zA-Z ]", buffer);

    fclose(fp);

    return buffer;
}

int main()
{
    save_text_file("example.txt");

    char* loaded = load_text_file("example.txt");

    printf("From file: %s\n", loaded);
```

```
    free(loaded) ;
    loaded = 0 ;

    return 0;
}
```

The program generates a text file in the program's directory.

11.2.2 Example 2

This program does some examples with binary files, such as saving and loading. This was the written code:

```
#include <stdio.h>

struct My_Data
{
    int whole_number;
    float real_number;
};

void save_binary_file(char* filename, struct My_Data d)
{
    FILE* fp = fopen(filename, "wb");

    fwrite(&d.whole_number, 1, sizeof(int), fp);
    fwrite(&d.real_number, 1, sizeof(float), fp);

    fclose(fp);
}

struct My_Data load_binary_file(char* filename)
{
    struct My_Data data;

    FILE* fp = fopen(filename, "rb");

    fread(&data.whole_number, 1, sizeof(int), fp);
    fread(&data.real_number, 1, sizeof(float), fp);

    fclose(fp);

    return data;
}

int main()
{
    struct My_Data data;
    data.whole_number = 47;
    data.real_number = 3.14f;
    save_binary_file("example.bin", data);
```

```
struct My_Data loaded;
loaded = load_binary_file("example.bin");

printf("From file: %d\n", loaded.whole_number);
printf("From file: %f\n", loaded.real_number);

return 0;
}
```

11.2.3 Example 3

This program shows how to use function pointers. This was the written code:

```
#include <stdio.h>

typedef void(*p_call_me) () ;

void print_hello()
{
    printf("Hello\n");
}

void print_goodbye()
{
    printf("Goodbye\n");
}

int main()
{
    p_call_me function_pointer = 0;

    function_pointer = print_hello;

    function_pointer();

    function_pointer = print_goodbye;

    function_pointer();

    return 0;
}
```

11.3 Exercises

11.3.1 Exercise 1

This program creates a file, writes some text on it, then closes the file. This was the written code:

```
#include <stdio.h>

int main()
{
    FILE* file_pointer = fopen("exercise.txt", "w");

    fprintf(file_pointer, "Hello File World!");

    fclose(file_pointer);

    printf("\n\n");

    return 0;
}
```

11.3.2 Exercise 2

This program takes a text file with some text on it, opens the file, scans the text, print the content of the file, and closes the file. This was the written code:

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

int main()
{
    FILE* p_text = fopen("input.txt", "r");

    char char_array;

    int line_count = 1;

    printf("%d ", line_count);

    while (fscanf(p_text, "%c", &char_array) == 1)
    {
        printf("%c", char_array);

        if (char_array == '\n')
        {
            printf("%d ", line_count);
            line_count++;
        }
    }

    fclose(p_text);

    printf("\n\n");

    return 0;
}
```

11.3.3 Exercise 3

This program simply prints out the values of the following preprocessor macros:

```
#include <stdio.h>
intmain()
{
    printf("File: %s\n", __FILE__);
    printf("Line: %d\n", __LINE__);
    printf("Date: %s\n", __DATE__);
    printf("Time: %s\n", __TIME__);
    printf("\n\n");
    return 0;
}
```

11.3.4 Exercise 4

This exercise gives different codes, which, put together, results on this program:

Multi-File Project.c:

```
#include <stdio.h>
#include "my.h"

int main()
{
    run_my();

    printf("\n\n");

    return 0;
}

my.c:

#include <stdio.h>
#include "my.h"

void run_my()
{
    printf("run_my() in my.c called!\n");
}

my.h

#ifndef MY_H_GUARD
#define MY_H_GUARD

void run_my();

#endif // MY_H_GUARD
```

11.3.5 Exercise 5

This program simulates a countdown timer. It asks the user for a positive whole number.

It keeps asking for a positive whole number until the user enters it. This was the written code:

```
#include <stdio.h>

int main()
{
    int input;
    printf("Enter a positive whole number: ");
    scanf("%d", &input);

    for (int i = input; i > 0; i--)
    {
        printf("%d", input);

        for (int n = input; n > 0; n--)
        {
            printf(".");
        }

        printf("\n");
        input--;
    }

    printf("The counter has finished!");

    printf("\n\n");

    return 0;
}
```

11.3.6 Exercise 6

This program prints a student ID using a function. This was the written code:

```
#include <stdio.h>

void print_student_id();

void print_student_id()
{
    printf("14885857");
}

int main()
{
    print_student_id();
    printf("\n\n");
    return 0;
}
```

11.3.7 Exercise 7

This program has a function which takes a learning outcome as parameter. The function prints the assessments according to the number. This was the written code:

```
#include <stdio.h>

void print_assessments(int learning_outcome);
void print_assessments(int learning_outcome)
{
    printf("Reporting Journal\n");
    printf("Final Practical Exam\n");

    if (learning_outcome <= 9)
    {
        printf("Practical Test 3\n");

        if (learning_outcome <= 8)
        {
            printf("Practical Test 2\n");

            if (learning_outcome <= 6)
            {
                printf("Practical Test 1\n");
            }
        }
    }
}

int main()
{
    int input;
    printf("Learning Outcome? ");
    scanf("%d", &input);

    printf("\n");
    print_assessments(input);

    printf("\n\n");
    return 0;
}
```

11.3.8 Exercise 8

This program has a function which takes a grade as parameter and output the respective letter grade. This was the written code:

```
#include <stdio.h>

char convert_percent_to_grade(int grade);
char convert_percent_to_grade(int grade)
{
    if (grade >= 80)
    {
        return 'A';
    }
    else if (grade >= 65)
    {
        return 'B';
    }
    else if (grade >= 50)
    {
        return 'C';
    }
    else
    {
        return 'D';
    }
}

int main()
{
    int grade;
    printf("Please, enter your grade: ");
    scanf("%d", &grade);

    char grade_letter = convert_percent_to_grade(grade);

    printf("%d evaluates a %c", grade, grade_letter);

    printf("\n\n");

    return 0;
}
```

11.3.9 Exercise 9

This program simply calls a function which prints hello. This was the written code:

```
#include <stdio.h>

void print_hello()
{
    printf("Hello!!!\n");
}

int main()
{
    print_hello();

    return 0;
}
```

11.3.10 Exercise 10

This program had a memory leak, to fix it, I wrote the following code:

```
#include <stdlib.h>

struct Memory_Buffer
{
    unsigned char buffer[64];
};

struct Memory_Buffer* get_memory_block()
{
    return (malloc(sizeof(struct Memory_Buffer)));
}

int main()
{
    struct Memory_Buffer* p_buffer = 0;

    p_buffer = get_memory_block();

    for (int k = 0; k < 64; ++k)
    {
        p_buffer->buffer[k] = k;
    }

    return 0;
}
```

11.3.11 Exercise 11

This program has an array with whole numbers, and counts how many of these elements are greater than 25. This was the written code:

```
#include <stdio.h>

int main()
{
    int data[] = { 12, 28, 33, 18, 27, 10, 13, 91, 15 };

    int counter = 0;

    for (int i = 0; i <= 9; i++)
    {
        if (data[i] > 25)
        {
            counter++;
        }
    }

    printf("There are %d elements bigger than 25.", counter);

    printf("\n\n");

    return 0;
}
```

11.3.12 Exercise 12

This program passes an array to a function with the size of the array. The function prints all the elements in the array. This was the written code:

```
#include <stdio.h>

void print_int_array(int* int_array, int elements);

void print_int_array(int* int_array, int elements)
{
    for (int i = 0; i < elements; i++)
    {
        printf("%d; ", int_array[i]);
    }

    printf("\n\n");
}

int main()
{
    int numbers[] = { 5, 3, 14, 3, 2, 1, 8, 12, 0, 2, 7, 8 };

    print_int_array(&numbers, 12);

    printf("\n\n");

    return 0;
}
```

11.3.13 Exercise 13

This program has a function which returns the number of even numbers in an array.

This was the written code:

```
#include <stdio.h>

int count_evens(int* int_array, int elements);

int count_evens(int* int_array, int elements)
{
    int counter = 0;

    for (int i = 0; i < elements; i++)
    {
        if (int_array[i] % 2 == 0)
        {
            counter++;
        }
    }

    return counter;
}

int main()
{
    int test_data[] = { 5, 3, 2, 4, 1, 8, 10, 2, 7 };

    int result = count_evens(&test_data, 9);

    printf("There are %d even numbers in the array.", result);

    printf("\n\n");

    return 0;
}
```

11.3.14 Exercise 14

This program has a function which return the number of vowels in an array. This was the written code:

```
#include <stdio.h>

int count_vowels(char* char_array);

int count_vowels(char* char_array)
{
    int counter = 0;
    int count_steps = 0;

    do
    {
        if (char_array[count_steps] == 'a' ||
            char_array[count_steps] == 'e' || char_array[count_steps] == 'i' ||
            char_array[count_steps] == 'o' || char_array[count_steps] == 'u' ||
            char_array[count_steps] == 'A' || char_array[count_steps] == 'E' ||
            char_array[count_steps] == 'I' || char_array[count_steps] == 'O' ||
            char_array[count_steps] == 'U')
        {
            counter++;
        }

        count_steps++;
    }
    while (char_array[count_steps] != '\0');

    return counter;
}

int main()
{
    char test_string[] = "CaN_i_progrAM?";
    int result = count_vowels(&test_string);
    printf("There are %d vowels in the array.", result);
    printf("\n\n");
}
```

```
    return 0;  
}
```

11.3.15 Exercise 15

This program prints a structure with some information. This was the written code:

```
#include <stdio.h>

void print_song(struct Song song);

struct Song
{
    char name[30];
    int runtime_seconds;
    int likes;
    int dislikes;
};

void print_song(struct Song song)
{
    printf("A song...\n\n");

    printf("Name: %s\n", song.name);
    printf("Runtime: %d seconds\n", song.runtime_seconds);
    printf("Likes: %d thumbs up\n", song.likes);
    printf("Dislikes: %d thumbs down\n", song.dislikes);
}

int main()
{
    struct Song p1_theme_music;
    sprintf(p1_theme_music.name, "COMP500/ENSE501 Remix");
    p1_theme_music.runtime_seconds = 250;
    p1_theme_music.likes = 105;
    p1_theme_music.dislikes = 315;

    print_song(p1_theme_music);

    printf("\n\n");

    return 0;
}
```

11.3.16 Exercise 16

This program outputs a representation of another structure. This was the written code:

```
#include <stdio.h>

void print_post(struct Post post);

struct Post
{
    char name[16];
    char date[20];
    char time[8];
    int likes;
    int shares;
};

void print_post(struct Post post)
{
    printf("A social media post...\n\n");

    printf("Name: %s people\n", post.name);
    printf("Likes: %d people\n", post.likes);
    printf("Shares: %d people\n", post.shares);
    printf("Date: %s\n", post.date);
    printf("Time: %s\n", post.time);
}

int main()
{
    struct Post comment;
    sprintf(comment.name, "Secret Admirer");
    sprintf(comment.date, "5 May 2016");
    sprintf(comment.time, "11:22am");
    comment.likes = 27;
    comment.shares = 2;

    print_post(comment);

    printf("\n\n");

    return 0;
}
```

11.3.17 Exercise 17

This program converts uppercase letters to lowercase letters. This was the written code:

```
#include <stdio.h>

void convert_to_lowercase(char char_array[]);
void convert_to_lowercase(char* char_array)
{
    for (int i = 0; char_array[i] != '\0'; i++)
    {
        if (char_array[i] >= 65 && char_array[i] <= 90)
        {
            char_array[i] += 32;
        }
    }
}

int main()
{
    char name[21];
    printf("Please enter your name: ");
    scanf(" %21s", &name);

    convert_to_lowercase(&name);

    printf("%s", name);

    printf("\n\n");
    return 0;
}
```

11.3.18 Exercise 18

This program returns a representations of some structures. This was the written code:

```
#include <stdio.h>

void print_movie(struct Movie movie);

struct Movie
{
    char name[30];
    char sequel[30];
    int release;
    int rotten_tomatoes;
    int is_sequel;
};

void print_movie(struct Movie movie)
{
    printf("Name: %s\n", movie.name);
    printf("Year of Release: %d\n", movie.release);
    printf("Rotten Tomatoes: %d%\n", movie.rotten_tomatoes);
    if (movie.is_sequel)
    {
        printf("%s's sequel was %s", movie.name, movie.sequel);
    }
    else
    {
        printf("%s has no sequel, yet!", movie.name);
    }

    printf("\n\n");
}

int main()
{
    struct Movie jurassic_1;
    sprintf(jurassic_1.name, "Jurassic Park");
    jurassic_1.release = 1993;
    jurassic_1.rotten_tomatoes = 93;
    jurassic_1.is_sequel = 1;

    struct Movie jurassic_2;
```

```
        sprintf(jurassic_2.name, "The Lost World: Jurassic
Park");
        sprintf(jurassic_1.sequel, jurassic_2.name);
jurassic_2.release = 1997;
jurassic_2.rotten_tomatoes = 51;
jurassic_2.is_sequel = 1;

struct Movie jurassic_3;
        sprintf(jurassic_3.name, "Jurassic Park III");
        sprintf(jurassic_2.sequel, jurassic_3.name);
jurassic_3.release = 2001;
jurassic_3.rotten_tomatoes = 50;
jurassic_3.is_sequel = 1;

struct Movie jurassic_world;
        sprintf(jurassic_world.name, "Jurassic Park");
        sprintf(jurassic_3.sequel, jurassic_world.name);
jurassic_world.release = 1993;
jurassic_world.rotten_tomatoes = 93;
jurassic_world.is_sequel = 0;

print_movie(jurassic_1);
print_movie(jurassic_2);
print_movie(jurassic_3);
print_movie(jurassic_world);

printf("\n\n");

return 0;
}
```

11.3.19 Exercise 19

This program also outputs a representation of some structures:

```
#include <stdio.h>

void print_season(struct Tv_Season season);

struct Tv_Season
{
    char premier_date[30];
    char season_finale_date[30];
    int season;
    int number_of_episodes;
    int is_next_season;
};

void print_season(struct Tv_Season season)
{
    printf("Season: %d\n", season.season);
    printf("Number of Episodes: %d\n", season.number_of_episodes);
    printf("Premier Date: %s\n", season.premier_date);
    printf("Season Finale Date: %s\n", season.season_finale_date);

    if (season.is_next_season)
    {
        printf("The next season was...\\n");
    }
    else
    {
        printf("Then the show was cancelled.");
    }
}

int main()
{
    struct Tv_Season season_1;
    season_1.season = 1;
    season_1.number_of_episodes = 13;
    sprintf(season_1.premier_date, "14 January 2005");
    sprintf(season_1.season_finale_date, "1 April 2005");
    season_1.is_next_season = 1;
```

```
struct Tv_Season season_2;
    season_2.season = 2;
    season_2.number_of_episodes = 20;
    sprintf(season_2.premier_date, "15 July 2005");
    sprintf(season_2.season_finale_date, "10 March 2006");
    season_2.is_next_season = 1;

struct Tv_Season season_3;
    season_3.season = 3;
    season_3.number_of_episodes = 20;
    sprintf(season_3.premier_date, "6 October 2006");
    sprintf(season_3.season_finale_date, "25 March 2007");
    season_3.is_next_season = 1;

struct Tv_Season season_4;
    season_4.season = 4;
    season_4.number_of_episodes = 20;
    sprintf(season_4.premier_date, "4 April 2008");
    sprintf(season_4.season_finale_date, "20 March 2009");
    season_4.is_next_season = 0;

print_season(season_1);
print_season(season_2);
print_season(season_3);
print_season(season_4);

printf("\n\n");

return 0;
}
```

12 Twelve

12.1 Lectures

This week we were introduced on how to use Microsoft Windows Software Development Kit (Microsoft Windows SDK). The Windows SDK includes Dialog Boxes, Window Menus, Check Box Controls, and other functionalities.

12.2 Examples

12.2.1 Example 1

This program creates colored columns using Microsoft Windows API. This was the written code:

```
#include <Windows.h>
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

enum Colours
{
    BLACK = 0,
    BLUE = 1,
    GREEN = 2,
    CYAN = 3,
    RED = 4,
    MAGENTA = 5,
    BROWN = 6,
    LIGHTGRAY = 7,
    DARKGRAY = 8,
    LIGHTBLUE = 9,
    LIGHTGREEN = 10,
    LIGHTCYAN = 11,
    LIGHTRED = 12,
    LIGHTMAGENTA = 13,
    YELLOW = 14,
    WHITE = 15
};

void set_text_colour(int foreground_colour, int background_colour)
{
    HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);

    int colour = ((background_colour & 0x0F) << 4) |
(foreground_colour & 0x0F);

    SetConsoleTextAttribute(console, colour);
}
```

```
}

void move_cursor_to(int column, int row)
{
    HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);

    COORD position;
    position.X = column;
    position.Y = row;

    SetConsoleCursorPosition(console, position);
}

int main()
{
    int colour = WHITE;
    int row = 23;
    int column = 76;

    srand((unsigned int)time(0));

    while (1)
    {
        for (row = 23; row > 0; --row)
        {
            move_cursor_to(column, row);

            set_text_colour(colour, colour);

            printf(" ");

            Sleep(1);

            --colour;

            if (colour <= 0)
            {
                colour = WHITE;
            }
        }

        column -= 4;

        if (column < 0)
        {
            column = 76;
        }
    }
}
```

```
    }  
  
    return 0;  
}
```

12.2.2 Example 2

This program simply outputs a message box. This was the written code:

```
#include <Windows.h>

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
LPSTR lpCmdLine, int nCmdShow)
{
    MessageBox(NULL, L"The last week of labs!", L"COMP500/ENSE501",
MB_OK);

    return 0;
}
```

12.2.3 Example 3

This program outputs a message box with two buttons. The program gives different outputs for each button. This was the written code:

```
#include <Windows.h>

#define MY_BUTTON_LEFT 100
#define MY_BUTTON_RIGHT 200

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
    switch (msg)
    {
        case WM_CREATE:
        {
            CreateWindow(L"Button", L"Left", WS_CHILD | WS_VISIBLE,
5, 5, 100, 30, hwnd, (HMENU)MY_BUTTON_LEFT, 0, 0);

            CreateWindow(L"Button", L"Right", WS_CHILD | WS_VISIBLE,
110, 5, 100, 30, hwnd, (HMENU)MY_BUTTON_RIGHT, 0, 0);
        }
        break;
        case WM_COMMAND:
        {
            if (LOWORD(wParam) == MY_BUTTON_LEFT)
            {
                MessageBox(hwnd, L"Left Button!", L"You pressed...", MB_OK);
            }
            else if (LOWORD(wParam) == MY_BUTTON_RIGHT)
            {
                MessageBox(hwnd, L"Right Button!", L"You
pressed...", MB_OK);
            }
        }
        break;
        case WM_CLOSE:
        {
            DestroyWindow(hwnd);
        }
        break;
    }
}
```

```

case WM_DESTROY:
{
    PostQuitMessage(0);
}
break;
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"LabWeek12";
    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"COMP500/ENSE501", WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
CW_USEDEFAULT, 235, 80, 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;
}
```

12.2.4 Example 4

This program is an example of a drawing tool. This was the written code:

```
#include <Windows.h>
#define MY_BUTTON_LEFT 100
#define MY_BUTTON_RIGHT 200
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg,
    WPARAM wParam, LPARAM lParam)
{
    switch (msg)
    {
        case WM_PAINT:
        {
            PAINTSTRUCT ps;
            HDC hdc = BeginPaint(hwnd, &ps);
            HBRUSH blue = CreateSolidBrush(RGB(0, 0, 255));
            HBRUSH green = CreateSolidBrush(RGB(0, 255, 0));
            HBRUSH red = CreateSolidBrush(RGB(255, 0, 0));
            HBRUSH old = SelectObject(hdc, blue);
            Rectangle(hdc, 20, 20, 150, 430);
            SelectObject(hdc, green);
            Rectangle(hdc, 170, 20, 300, 430);
            SelectObject(hdc, red);
            Rectangle(hdc, 320, 20, 460, 430);
            SelectObject(hdc, old);
            DeleteObject(blue);
            DeleteObject(green);
            DeleteObject(red);
            EndPaint(hwnd, 0);
        }
        break;
        case WM_CLOSE:
        {
            DestroyWindow(hwnd);
        }
        break;
        case WM_DESTROY:
        {
            PostQuitMessage(0);
        }
        break;
        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"LabWeek12";
    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"COMP500/ENSE501",
                         WS_OVERLAPPEDWINDOW,
                         CW_USEDEFAULT, CW_USEDEFAULT,
                         235, 80,
                         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;  
}
```

12.3 Exercises

12.3.1 Exercise 1

This program uses a first week's code and the example codes of this week to output different colours in the text. This was the written code:

Colour Console Experiment.c :

```
#include <stdio.h>
#include "Functions.h"

enum Colours
{
    BLACK = 0,
    BLUE = 1,
    GREEN = 2,
    CYAN = 3,
    RED = 4,
    MAGENTA = 5,
    BROWN = 6,
    LIGHTGRAY = 7,
    DARKGRAY = 8,
    LIGHTBLUE = 9,
    LIGHTGREEN = 10,
    LIGHTCYAN = 11,
    LIGHTRED = 12,
    LIGHTMAGENTA = 13,
    YELLOW = 14,
    WHITE = 15
};

int main()
{
    int foreground_colour = LIGHTRED;
    int background_colour = BLUE;

    set_text_colour(foreground_colour, background_colour);
```

```

printf("      + \n", 92);
printf("      /| \n", 92);
printf("      / |+ \n", 92);
printf("      / ||%c \n", 92);
printf("      / || %c \n", 92);
printf("      / ||  %c \n", 92);
printf("      / ||   %c \n", 92);
printf("      +-----+ \n");
printf("      _____%c|_____ \n", 37);
printf("      %c THE DRAGONBORN/ \n", 92);
printf("%c%c%c%c%c%c%c%c%c%c%c%c \n", 126, 126,
126, 126, 126, 126, 126, 126, 126, 126, 126);
return 0;
}
Change Colour.c

#include <Windows.h>
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

enum Colours
{
    BLACK = 0,
    BLUE = 1,
    GREEN = 2,
    CYAN = 3,
    RED = 4,
    MAGENTA = 5,
    BROWN = 6,
    LIGHTGRAY = 7,
    DARKGRAY = 8,
    LIGHTBLUE = 9,
    LIGHTGREEN = 10,
    LIGHTCYAN = 11,
    LIGHTRED = 12,
    LIGHTMAGENTA = 13,
    YELLOW = 14,
    WHITE = 15
};

void set_text_colour(int foreground_colour, int background_colour)
{
    HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
    int colour = ((background_colour & 0x0F) << 4) |
(foreground_colour & 0x0F);
}

```

```
    SetConsoleTextAttribute(console, colour);  
}  
Functions.h  
  
#ifndef MYFUNCTIONS_H_GUARD  
#define MYFUNCTIONS_H_GUARD  
  
void set_text_colour(int foreground_colour, int background_colour);  
#endif
```

12.3.2 Exercise 2

This program simply outputs a message box which asks if the user is ready for the Final Practical Exam, if the user answers Yes, he/she gets a message “Excellent!”, otherwise, the user gets a message “More study required!”, this was the written code:

```
#include <Windows.h>

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
LPSTR lpCmdLine, int nCmdShow)
{
    int answer = MessageBox(NULL, L"Are you ready for the
COMP500/ENSE501 Final Practical Exam?", L"COMP500/ENSE501",
MB_YESNO);

    if (IDYES == answer)
    {
        MessageBox(NULL, L"Excellent!", L"Excellent", MB_OK);
    }
    else if (IDNO == answer)
    {
        MessageBox(NULL, L"More study required!", L"More study
required", MB_OK);
    }

    return 0;
}
```

14885857

Reporting Journal #4

14885857

Reporting Journal #4

14885857

Reporting Journal #4

Paludo

Page 680 of 688

13 Hours of study

In the weeks on to three, it was hard to document all the activity time and duration because I just took notes about the whole time for each day of study and a general idea of what I did in each respective day, to write this reporting journal I had to look at all my files and try to figure out what I have exactly done in each day based on my routine.

In the weeks three to six, I took notes of every independent study that I have done, here follows the table with all the recorded information:

Date	Starting time	Finishing time	Activity
February 29th	12 p.m.	2 p.m.	Lecture.
February 29th	5 p.m.	6 p.m.	Reading lecture notes.
February 29th	9 p.m.	10 p.m.	Reading books.
March 1st	2 p.m.	4 p.m.	Laboratory. Exercises one to five.
March 1st	5 p.m.	6 p.m.	Reading books.
March 1st	6 p.m.	7 p.m.	Going through examples.
March 1st	7 p.m.	8 p.m.	Rewriting exercises one to five.
March 4th	9 a.m.	10 a.m.	Lecture.
March 4th	11 a.m.	12 p.m.	Reading lecture notes.
March 4th	2 p.m.	4 p.m.	Sharing information with classmates, reading my own codes, reading books and lecture notes.
March 4th	7 p.m.	8 p.m.	Writing reporting journal.
March 5th	6 p.m.	7 p.m.	Reading books, lecture notes, previous codes and researching.
March 5th	7 p.m.	10 p.m.	Week one exercises six to eight.
March 6th	2 p.m.	5 p.m.	Writing reporting journal.
March 7th	12 p.m.	2 p.m.	Lecture.
March 7th	5 p.m.	6 p.m.	Reading lecture notes.
March 7th	9 p.m.	10 p.m.	Reading books and researching.
March 8th	2 p.m.	4 p.m.	Laboratory. Reading books, going through lecture notes, internet researching, looking at examples, coding exercises one and two.
March 8th	5 p.m.	6 p.m.	Reading books.
March 8th	6 p.m.	8 p.m.	Going through examples.

March 8th	9 p.m.	11 p.m.	Reviewing codes, writing codes and drawing flowcharts for exercises one to four.
March 10th	5 p.m.	6 p.m.	Reading books, lecture notes and researching.
March 11th	9 a.m.	10 a.m.	Lecture.
March 11th	11 a.m.	12 p.m.	Reading lecture notes.
March 11th	1 p.m.	3 p.m.	Looking at examples, written codes, reading books, lecture notes, and researching.
March 11th	5 p.m.	7 p.m.	Drawing flowcharts and coding exercises five to seven.
March 12th	7 p.m.	8 p.m.	Looking at examples, written codes, reading books, lecture notes and researching.
March 12th	8 p.m.	9 p.m.	Drawing flowchart and coding exercise eight.
March 12th	9 p.m.	10 p.m.	Drawing flowchart, researching and sharing information about exercise nine.
March 12th	10 p.m.	12 a.m.	Coding exercise nine.
March 13th	9 a.m.	12 p.m.	Writing reporting journal.
March 13th	8 p.m.	10 p.m.	Writing reporting journal.
March 14th	12 p.m.	2 p.m.	Lecture.
March 14th	5 p.m.	6 p.m.	Reading lecture notes.
March 14th	9 p.m.	10 p.m.	Reading books.
March 15th	2 p.m.	4 p.m.	Laboratory. Looking at examples and writing codes for exercises one to three.
March 15th	5 p.m.	7 p.m.	Going through examples.
March 15th	7 p.m.	8 p.m.	Looking at written codes, drawing flowcharts, and reprogramming exercises one to three.
March 15th	8 p.m.	9 p.m.	Reading books and researching.
March 16th	5 p.m.	9 p.m.	Writing reporting journal.
March 17th	1 p.m.	6 p.m.	Writing reporting journal, reading books, internet researching, looking at examples, going through written codes, reading lecture notes, and sharing information with classmates.
March 17th	7 p.m.	11 p.m.	Writing reporting journal.
March 18th	9 a.m.	10 a.m.	Lecture.
March 18th	10 a.m.	12 p.m.	Writing reporting journal.
March 18th	6 p.m.	7 p.m.	Reading lecture notes and books.
March 18th	8 p.m.	9 p.m.	Writing reporting journal.
March 18th	10 p.m.	11 p.m.	Writing reporting journal.
March 19th	9 a.m.	11 a.m.	Writing reporting journal.
March 19th	12 p.m.	2 p.m.	Writing reporting journal.
March 19th	3 p.m.	4 p.m.	Reading books.
March 19th	4 p.m.	6 p.m.	Drawing flowcharts and writing codes for exercises four and five, and starting exercise six.
March 19th	7 p.m.	8 p.m.	Reading books, internet researching, reading lecture notes and going through examples.
March 19th	8 p.m.	10 p.m.	Exercise six.

March 20th	1 p.m.	6 p.m.	Drawing flowcharts and codes for exercises seven to nine, writing the reporting journal, reading lecture notes and books, and internet researching.
March 20th	8 p.m.	10 p.m.	Writing reporting journal.
March 21st	6 a.m.	8 a.m.	Writing reporting journal.
March 21st	12 p.m.	2 p.m.	Lecture.
March 21st	6 p.m.	7 p.m.	Reading lecture notes.
March 21st	8 p.m.	9 p.m.	Reading books.
March 22nd	7 a.m.	8 a.m.	Writing reporting journal.
March 22nd	2 p.m.	4 p.m.	Laboratory, exercises one to 15.
March 22nd	4 p.m.	5 p.m.	Exercise 16.
March 22nd	6 p.m.	8 p.m.	Writing reporting journal.
March 23rd	8 a.m.	12 p.m.	Drawing flowcharts, rewriting the codes for all exercises and writing the reporting journal.
March 23rd	3 p.m.	8 p.m.	Writing reporting journal.
April 1st	9 a.m.	10 a.m.	Lecture.
April 1st	11 a.m.	14 p.m.	Reading lecture notes, books and researching.
April 1st	5 p.m.	8 p.m.	Reading lecture notes, books and writing the reporting journal.
April 2nd	12 p.m.	8 p.m.	Reading Lecture notes, books, researching, going through examples and writing the reporting journal.
April 2nd	9 p.m.	11 p.m.	Writing codes for exercises 1 to 3, drawing flowcharts and writing the reporting journal.
April 3rd	7 a.m.	9 a.m.	Writing codes for exercises 4 to 7, drawing flowcharts and writing the reporting journal.
April 3rd	3 p.m.	11 p.m.	Researching, reading books, reading lecture notes, writing codes for exercises 8 to 16, drawing flowcharts and writing the reporting journal.
April 4th	12 p.m.	2 p.m.	Lecture
April 4th	6 p.m.	7 p.m.	Reading lecture notes.
April 4th	8 p.m.	10 p.m.	Reading books and researching.
April 5th	2 p.m.	4 p.m.	Laboratory, exercises 1 to 9.
April 5th	5 p.m.	8 p.m.	Reading books, researching and reading lecture notes.
April 8th	9 a.m.	10 a.m.	Lecture.
April 8th	11 a.m.	12 p.m.	Reading lecture notes.
April 9th	2 p.m.	10 p.m.	Going through examples, reading books, internet researching, reading lecture notes, and writing reporting journal.
April 10th	8 a.m.	11 a.m.	Rewriting codes, drawing flowcharts for exercises 1 to 9 and writing reporting journal.

April 10th	1 p.m.	11 p.m.	Writing codes for exercises 10 to 24 and writing the reporting journal.
April 11th	12 p.m.	2 p.m.	Lecture
April 11th	6 p.m.	7 p.m.	Reading lecture notes.
April 11th	8 p.m.	10 p.m.	Reading books and researching.
April 12 th	2 p.m.	4 p.m.	Laboratory. Reading books, going through lecture notes, internet researching, looking at examples, coding exercises 1 to 3.
April 12th	5 p.m.	6 p.m.	Reading books.
April 12th	6 p.m.	8 p.m.	Going through examples.
April 15th	9 a.m.	10 a.m.	Lecture.
April 15th	11 a.m.	12 p.m.	Reading lecture notes.
April 15th	1 p.m.	3 p.m.	Looking at examples, written codes, reading books, lecture notes, and researching.
April 16th	2 p.m.	5 p.m.	Drawing flowcharts and coding exercises 1 to 3.
April 16th	6 p.m.	8 p.m.	Finishing exercise 3 and writing reporting journal.
April 17th	8 a.m.	11 p.m.	Writing codes for exercises 4 to 22 and writing the reporting journal
May 2nd	12 p.m.	2 p.m.	Lecture
May 2nd	6 p.m.	7 p.m.	Reading lecture notes.
May 2nd	8 p.m.	10 p.m.	Reading books and researching.
May 3rd	2 p.m.	4 p.m.	Laboratory. Writing reporting journal.
May 3rd	5 p.m.	6 p.m.	Reading books.
May 6th	9 a.m.	10 a.m.	Lecture.
May 6th	11 a.m.	12 p.m.	Reading lecture notes.
May 7th	2 p.m.	4 p.m.	Going through examples and writing the reporting journal.
May 8th	8 a.m.	10 p.m.	Reading books, going through lecture notes, internet researching, looking at examples, coding exercises, and writing the reporting journal.
May 9th	12 p.m.	2 p.m.	Lecture
May 9th	6 p.m.	7 p.m.	Reading lecture notes.
May 9th	8 p.m.	10 p.m.	Reading books and researching.
May 10th	2 p.m.	4 p.m.	Laboratory. Writing reporting journal.
May 13th	9 a.m.	10 a.m.	Lecture.
May 13th	11 a.m.	12 p.m.	Reading lecture notes.
May 13th	2 p.m.	5 p.m.	Going through examples, internet researching, reading lecture notes, reading books, and writing reporting journal.
May 14th	1 p.m.	8 p.m.	Writing codes for exercises, going through examples, internet researching, reading lecture notes, reading books, and writing reporting journal.
May 16th	12 p.m.	2 p.m.	Lecture
May 16th	6 p.m.	7 p.m.	Reading lecture notes.

May 16th	8 p.m.	10 p.m.	Reading books and researching.
May 17th	2 p.m.	4 p.m.	Laboratory. Writing codes for exercises 1 to 3.
May 18th	3 p.m.	9 p.m.	Reading lecture notes, internet researching, reading books, going through examples, writing codes and drawing flowcharts for exercises.
May 19th	3 p.m.	6 p.m.	Writing reporting journal.
May 20th	9 p.m.	10 p.m.	Lecture.
May 20th	1 p.m.	2 p.m.	Reading lecture notes.
May 23rd	12 p.m.	2 p.m.	Lecture.
May 23rd	6 p.m.	7 p.m.	Reading lecture notes.
May 23rd	8 p.m.	10 p.m.	Reading books and researching.
May 24th	2 p.m.	4 p.m.	Laboratory.
May 25th	4 p.m.	10 p.m.	Reading books, internet researching, reading lecture notes, going through examples, writing codes for exercises, studying for the test.
May 27th	9 a.m.	10 a.m.	Lecture.
May 27th	11 a.m.	12 p.m.	Reading lecture notes.
May 30th	12 p.m.	2 p.m.	Lecture.
May 31st	2 p.m.	4 p.m.	Laboratory.
May 31st	8 p.m.	10 p.m.	Writing reporting journal.
June 1st	6 p.m.	11 p.m.	Writing reporting journal.

References

- Ahmad, U. (n. d.). *Convert Hexadecimal to Decimal C*. Retrieved April 08, 2016, from <http://robustprogramming.com/convert-hexadecimal-to-decimal-c/>
- Davis, S. R. (2014). *C++ For Dummies* (7th ed.). Hoboken, New Jersey: John Wiley & Sons, Inc.
- Deitel, P. & Deitel, H. (2010). *C How to Program* (6th ed.). New Jersey, United States of America: Pearson Education.
- Horton, I. (2013). *Begginning C* (5th ed.). New York, United States of America: Apress.
- Perry, G. (1992). *C++ by Example*. United States of America: Lloyd Short.
- Sempf, B., Sphar, C. & Davis, S. R. (2010). *C# 2010 All-in-One For Dummies*. Indianapolis, Indiana: Wiley Publishing, Inc.
- Walia, D. (n. d.). C Palindrome Program: C Program for Palindrome. Retrieved May 11, 2016, from <http://www.programmingsimplified.com/c-program-find-palindrome>