

DUBLIN CITY UNIVERSITY

SCHOOL OF ELECTRONIC ENGINEERING

**SCHOOL OF MECHANICAL AND
MANUFACTURING ENGINEERING**

**SOFTWARE DEVELOPMENT FOR ENGINEERS
(EM108)**

Laboratory Manual

Author:

Dr. Gabriel-Miro Muntean

2016

Section 1

Introduction

This is the laboratory manual for the module EM108 Software Development for Engineers. EM108 is a required module for all students enrolled for the following academic programmes offered in Dublin City University, School of Electronic Engineering and School of Mechanical and Manufacturing Engineering.

- Electronic and Computer Engineering
- Mechanical and Manufacturing Engineering
- Medical and Mechanical Engineering
- Manufacturing Engineering with Business Studies
- Mechatronic Engineering
- Common Entry to Engineering

The aim of the EM108 module is to develop familiarity with basic concepts and techniques of software engineering in C. The notion of a programming language will be introduced, as well as compiling and debugging issues. A selection of simple C programs will be developed first and then more complex programming issues will be introduced along with more complex programs.

The module is preparatory in nature, providing necessary foundations for a variety of modules in subsequent stages of the relevant programmes. It is delivered via two lectures and one three-hour laboratory session per week. At the end of most lab sessions, the students are required to submit lab reports via e-mail. In total there are seven reports to submit. Each of the reports is separately assessed and accounts for either 5% or 10% of the final mark as follows: session 1, 2, 3, 6, 7 and the computer-based exam simulation from session 10 reports are worth 5%, whereas sessions 4 and 5, and sessions 8 and 9 reports account for 10% each. The remaining 50% of the mark is awarded after the DCU Registry-organised computer-based exam which takes place during the exam session.

Section 2

Laboratory Exercises

Introduction

The laboratory exercises are very important for both your understanding of the material taught during lectures and your building practical experience in writing, compiling, debugging and testing computer programs in general and C programs in particular.

Workload

Each student attends ten three-hour lab sessions, scheduled every week, starting from the second week. Last week meeting is reserved for those of you who want to do a lab exercise they missed. Only one lab exercise can be submitted at the end of the last week session.

The lab attendance is compulsory. The lab exercises are getting increasingly extensive and demanding as the semester progresses. You will not be able to read the laboratory requirements for first time and carry out the required tasks within the three hours of the lab. It is strongly recommended that you study and prepare for the lab in advance, as this will help you understand better the problems you have to solve, design good solutions, write and test properly the programs properly and write good reports. It is also advised to spend at least three further hours to study privately the material from the lecture notes, textbooks, online tutorials and to write programs and test them on a computer every week.

The lab sessions are such structured that each student has a separate PC. If for private study you find easier to work together with one or two colleagues, this is permitted. However the lab reports should be written individually and they should reflect your own work only.

Logbook

It is advisable to have a new copybook or notebook for use as a logbook for this module. Bring this along to all module activities – lectures and labs - and use it to keep your own personal notes and questions. In contrast to most other subjects, this logbook will be used to hold formal lab reports. Instead, you will maintain it for your own personal notes, drawings and code excerpts, which will serve as the basis for writing formal reports, which will actually be submitted (and archived) electronically.

Report Guidelines

You are requested to submit formal lab reports at the end of each lab session via e-mail. You should consult section 4 of this Laboratory manual for an example of how

to prepare a lab report. If the lab report does not include your details and the declaration that states that the report presents your own work only, the report will be marked with 0. The report should include a separate section for each problem to be solved during that lab session, but part of the same document. Only Microsoft Word formatted submissions are accepted. The code written must be submitted in separate C files, but attached to the same e-mail. No other files should be submitted such as executables or object files.

Report Submission

The lab reports and the code (only – not the exe file) must be submitted via the DCU *loop* system at:

<http://loop.dcu.ie>

It is your responsibility to make sure that the documents have been uploaded. Allow 20 minutes at the end of each lab session to do that. Send a copy to your e-mail address so it will be in the Inbox e-mail folder for further reference. The name of the files should indicate the lab number: "Lab3_report". All lab report submissions must be made before the end of the lab session. No late submission will be accepted.

Laboratory Session 1

Introduction

In this session you will develop and test an actual computer program for the first time. You will be using the C language. As this is a high-level language (which the computer cannot understand as is), you will need to translate the program into a low-level or machine language that the Central Processing Unit (CPU) of your computer can actually execute. A special program called compiler performs this translation.

There are a wide variety of C compilers available for the Windows environment, each with their own particular strengths and weaknesses. You will be using Borland C++ compiler. Details about the compiling process are given in section 4 of this laboratory manual.

This session is also the first to require you to prepare a formal lab report. Section 3 of this laboratory manual provides an example of a report submission. The same format will be required also for the final computer-based exam report.

You will receive individual grades for each lab report and they will contribute to your overall assessment result for this module. 50% of the overall module marks is awarded for the labs. It is advisable to write the report as you progress with the work during the lab session.

Create in your home directory (the H: drive) a working directory called for example **em108** and in that folder - a directory called **lab1**. It is strongly recommended to save all the materials related to the current lab session in **lab1** directory.

Copy the template of the lab report in **lab1** directory and open it using Microsoft Word. Keep the Word editor running throughout the lab session, so that you can add to it as you go along. Remember to save every time you add any significant amount of text, otherwise you run the risk of losing this text completely if, for example, there is a power failure in the lab! It is your responsibility to prevent this from happening.

Write your programs using “Textpad”. Never use Word for this, as it saves the content in a different format than plain text as required by the C compiler. Save them using name representative for their content. For example the file that stores the program, which prints the “Hello World” message, can be called “hello.c”.

Exercise 1: Hello World! (50%)

Here is the minimal “Hello World!” program discussed in the lectures

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

int main(void)
{
    printf("Hello World!\n");
    return(EXIT_SUCCESS);
}
```

Use the text editor - Textpad - to create a file called “hello.c” in your working directory (H: /em108/lab1) containing the program listed above.

Compile your program by giving the following command:

```
bcc32 hello.c
```

Automatically the compiler creates the compiled executable, calls it “hello.exe” and places it in the same directory. More details about compiling procedure are given in Section 4 of this manual.

Of course, “hello.c” should compile without any problems - if you have entered it correctly. But if any problems are encountered try to determine what was inputted wrong and fix the problem(s). If you have problems, seek help from the demonstrators by raising your hand.

To execute the program generated you simply give its name as a command:

```
hello.exe  
or  
hello
```

Check that the program executes as you expect. If it does not, try to figure out why, and fix the problem(s). Again, if you encounter problems, ask for help from the demonstrators.

If the program executes correctly, then congratulations - you have just entered, compiled, and tested your first C program!

Do not forget to write your report for exercise 1. A template is provided in Section 3.

Exercise 2: Branching out! (50%)

Here is the program called “insult.c”:

```
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
int main(void)  
{  
    char name[100];  
  
    printf("Welcome to INSULT ...\n");  
    printf("Please enter your name:\n");  
    scanf("%s", name);  
  
    if(strcmp(name, "John") == 0)  
        printf("Hello, %s. Welcome to my program!\n", name);  
    else  
        printf("I don't like you, %s. Good bye!\n", name);  
  
    printf("Bye from INSULT ...\n");  
  
    return(EXIT_SUCCESS);  
}
```

Using this as a basis, or template, develop (plan, code, test, correct) a program called “sport.c” which will behave as follows:

- Displays a suitable welcome/identification message.
- Ask the user to type in his or her favourite sport (one word only).
- If the topic is different than “Soccer”, display the message: “What? SPORT? Not good enough! I like Soccer!”, where in the place of “SPORT” you display the sport entered by the user.
- If the topic is “Soccer”, display the message “Good Choice! I like SPORT, too!” instead. In the place of “SPORT” you display the sport entered by the user (“Soccer” in this case!)
- Finally display some suitable closing message.

Record in your report your experiences in developing this program - especially whatever problems you encounter and how did you solve them.

Test the program with different inputs and record in your report the program’s output.

Conclusion

The major work of the lab session is now complete. The formal lab report, which you have prepared during the session, must be submitted by email. Perform the following steps:

- Read quickly back through the report, correcting any obvious errors, and possibly adding some brief concluding remarks.
- Exit from the text editor, saving the final version of the report
- Go to the DCU loop website and upload the files: one code file for each of the exercises and the report with sections for each exercise:

loop.dcu.ie

- Email to yourself the same files. Fill in the subject of the message as “Lab1 report”
- Now attach your report file to the e-mail. Also attach the source files: “hello.c” and “sport.c”
- Send the message.

Laboratory Session 2

Exercise 1: Variables, Operators, Expressions and Statements (20%)

State whether the following are TRUE or FALSE. Explain why and if the answer is FALSE indicate the correct solution.

- a) The expression $(x > y \ \&\& \ a < b)$ is TRUE if either $x > y$ is TRUE or $a < b$ is TRUE.
- b) An expression containing `||` operator is TRUE if either or both of its operands are TRUE.
- c) The following are correct variable names: `sum`, `4me`, `_result`, `a*b`, `m67`
- d) Assuming that `a` and `b` are variables, the following are correct expressions: `ab`, `a && b`, `a_ * b`, `a + a / b`, `a == b`
- e) The following expression tests for equality between variables `x` and `b`: `x = b`
- f) The following statement correctly prints `x` of type integer: `printf("%f", x);`
- g) The following statement correctly reads `x` of type float: `printf("%f", &x);`
- h) The following expression tests if variable `x` is greater or equal than variable `y`:
`x => y`
- i) After the evaluation of `x = 7 + 3 * 6 / 2 - 1`; `x` has value 29
- j) Given the equation $y = ax^3 + 7$, the following is a correct C statement: `y = (a * x * x * (x + 7))`

Exercise 2: Water Basin (20%)

Write a program that reads the dimensions of a water basin: depth, length and width. It should also read the amount of paint required to cover one square meter of a surface. The program should compute the amount of paint required to paint this basin on the outside as well as the amount of water it could hold. Provide nice welcome and goodbye messages.

Exercise 3: Geometric Figures (20%)

Write a program that uses the following “print” statement:

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

What does the program print?

Next, print a square and a diamond. Use your initial instead of the “*” character.

Exercise 4: Exchange Office (40%)

Write a program that reads the exchange rates between EURO and US DOLLAR and BRITISH POUND respectively. Print the following table for EURO values from 1 to 10. Ask the user to input an amount in EURO and express his interest in exchanging it in either USD or GBP. Convert the amount in the right currency and print the value on the screen. Provide nice welcome and goodbye messages.

EURO	USD	GBP
1	1.293	0.693
2	2.586	1.386
and so on		

Laboratory Session 3

Exercise 1: Variables, Operators, Expressions and Statements (20%)

State whether the following are TRUE or FALSE. Explain why and if the answer is FALSE indicate the correct solution.

- a) If x is 3 and y is 4, the expression $(x+1 \geq --y \ \&\& \ x == y)$ is FALSE.
- b) If x is 3.5 and y is 4.5, after $y = x+y$, x is 3.5 and y is 4.5.
- c) The following are correct declarations: integer _a, B; float 1c; double \$\$\$;
- d) Assuming that a and b are integer variables and their values are 6 and 4 respectively the result of a/b is 1.5
- e) After the execution of the following sequence the value of b is 1.
int a = 4;
float b = 3;
b = a/b;

Exercise 2: Weed Killing (20%)

Write a program that reads the dimensions of a rectangular field expressed in meters: length and width as well as the quantity of weed killer required to treat the field expressed in liters/square feet. Knowing that 1 meter = 3.28 feet, compute and display how many liters of weed killer are required. Provide nice messages.

Exercise 3: Sum and Average Calculation – 3 numbers (40%)

Write a program that reads from the keyboard three numbers, computes and displays their sum, their product and their average as well as the minimum and the maximum value of the three numbers. Use “if” statement only.

Exercise 4: Sum and Average Calculation – N numbers (20%)

Write a program that reads from the keyboard a positive integer N representing how many numbers we have to process. Making use of a loop, the program should then read N numbers, compute and display their sum, their product and their average.

Laboratory Sessions 4 and 5

Exercise 1: Variables, Operators, Expressions and Statements (20%)

State whether the following are TRUE or FALSE. Explains why and if the answer is FALSE indicate the correct solution.

- a) An float variable occupies 4 or 8 bytes of memory, whereas an integer variable 4 bytes and a char variable 1 byte
- b) When a new value is stored in a variable, the variable's resulted value is the old value plus the new value.
- c) It is possible to assign a float value to an integer variable and nothing wrong happens.
- d) There are no mistakes in the following statements:
`char name[30];`
`scanf("%s", &name);`
`printf("Your name is: %d\n", name);`
- e) If a is equal to 3, all following statements – executed separately and not one after the other - print "a = 4".
`printf("a = %d", a++);`
`printf("a = %d", ++a);`
`printf("a = %d", a+1);`

Exercise 2: Problem A (20%)

Select a problem marked * or ** from Sets 1, 2, 3 and 4.

Exercise 3: Problem B (20%)

Select a problem marked *** from Sets 1, 2, 3 and 4.

Exercise 4: Problem C (40%)

Select a problem marked **** or ***** from Sets 1, 2, 3 and 4.

Laboratory Sessions 6 and 7

Exercise 1: Soccer Tickets or Star Printing (20%)

Solve one of the following two problems (A OR B – not both):

- A) Write a program that first reads the number of people who bought soccer tickets. For each person, it reads the number of tickets bought and prints it in the following table:

CLIENT NO.	NO. TICKETS
1	2
2	3
3	1
4	4

The program should then compute and print the average numbers of tickets bought by soccer fans.

- B) Write a program that prints the following patterns separately, one below the other. Use “for” loops.

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

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

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

Exercise 2: The Caesar Cipher (40%)

Encryption is the process of disguising a text so that its meaning will not be apparent to anyone who may intercept it, accidentally or not. Encryption is a fundamental technology in the development of secure electronic payment systems, which, among other things, are allowing the increasing commercial exploitation of the Internet.

In this exercise you will work with a very simple encryption procedure. This is called a substitution cipher, in which each letter of the original message (plaintext) is replaced by a different letter to generate the coded message (ciphertext).

To simplify matters further, we will restrict our plaintext to use only the upper case letters: A to Z, together with the space and newline characters; furthermore, space and newline characters will be left unchanged by the encryption process.

In each case, the replacement letter will be a fixed number of letters later in the alphabet compared to the original letter (and wrapping around back to A again after Z, as necessary). The number of letters to count forward will be called the encryption key. For example, with a key of 3, we would replace A by D, B by E and so on, with, finally, W being replaced by Z, X by A, Y by B and Z by C.

This simple kind of cipher is also called Caesar Cipher after Julius Caesar, the Roman emperor, who is said to have used it for secure battlefield messages.

To implement a Caesar cipher with a C program, note that, when accessing individual characters in C, what you are really dealing with are numeric codes for the characters. As such, you can do any normal arithmetic (addition, subtraction etc.) and comparison (greater-than, less-than etc.) operations on them. The relationship between the characters, as displayed on your screen or in your editor, and the numeric codes accessed within your C program is not specified by the C language standard. It is thus open to anyone developing a C compiler to adopt any character numbering scheme they like; you cannot reliably assume that the numbering scheme in effect for one C compiler will be the same as for another. However, there is a particular numbering scheme which is the most widely used, and to which most C compilers conform (including our compiler): American Standard Code for Information Interchange (ASCII) scheme. In ASCII, the upper case letters A to Z have consecutive numeric codes from 65 to 90. Thus, given a particular key value (a number from 1 to 25) simple addition will encypher a letter as required - except that if the sum is greater than 90 you must wrap around to the start of the alphabet again by subtracting 26, which is the length of the alphabet).

You are required to develop a program which will input a series of lines of characters, and encode them with a Caesar cipher. As each line of plaintext is input, the corresponding line of ciphertext should be output. It is up to you to devise a mechanism whereby the user can indicate that all lines have been input, and to implement the program in such a way that it will then terminate.

Originally the key for the encoding should be hard coded into your program (as opposed to being read in at run time). Later on, **update your program to read the key.**

Exercise 3: Projectile Problem (40%)

The purpose of this exercise is to calculate the theoretical trajectory of a projectile in a constant gravitational field. In other words, we want to calculate the path followed by, say, a ball thrown in the air or of an artillery shell for example.

We are going to make some simplifications. We will work in just two dimensions (i.e., as if we are looking exactly side on at the path of the ball). We will also neglect air resistance, and will treat the projectile as a point mass. This allows a simple analysis of the motion of the projectile, using Newtonian physics theory.

In our two-dimensional (x and y) coordinate system, x will denote the horizontal position, and y will denote the vertical position. According to physics the movement equation on the x direction and consequently the position x at moment in time t is given by:

$$x = x_0 + v_x * (t - t_0), \text{ where} \quad (1)$$

$$v_x = v_{x0} + a_x * (t - t_0) \quad (2)$$

In equations (1) and (2), we have denoted with x_0 the original position on Ox axis (horizontal) at moment t_0 , v_x the horizontal component of the velocity at moment t, v_{x0} the horizontal component of the velocity at moment t_0 and a_x the horizontal component of the acceleration.

Similarly, y denotes the vertical position and according to physics the movement equation on the y direction and consequently the position y at moment in time t is given by:

$$y = y_0 + v_y * (t - t_0), \text{ where} \quad (3)$$

$$v_y = v_{y0} + a_y * (t - t_0) \quad (4)$$

In equations (3) and (4), we have denoted with y_0 the original position on Oy axis (vertical) at moment t_0 , v_y the vertical component of the velocity at moment t, v_{y0} the vertical component of the velocity at moment t_0 and a_y the vertical component of the acceleration.

Command Line Redirection

Suppose your program is in a file called "projectile.c". After successful compilation, you will have an executable file called "projectile.exe". Normally you execute this simply by giving the name "projectile" as a command. However, if you want to use command line redirection, you would give a command somewhat like this instead:

```
projectile > out.txt < in.txt
```

The "<" serves as the input redirection character in a command. The effect is that the input for "projectile" will no longer be taken from the keyboard, but from the file

indicated instead. Similarly, the “>” character in a bash command provides for output redirection, to direct output to the file indicated instead of the screen. With any given command invocation you can choose to use no redirection at all, to use input redirection only, output redirection only, or both.

Task 1 Solving the Problem (1/2 of the points)

Assuming that the initial time $t_0 = 0$, initial position has $x_0 = 0$ and $y = y_0$ (representing the height of a man throwing the ball) and knowing that there is no horizontal component of acceleration $a_x = 0$ (there is only the vertical component due to the Earth gravity: $a_y = -g = -9.81 \text{ m/s}^2$), equations (1)-(4) become:

$$x = v_{x0} * t, \quad (5)$$

$$y = y_0 + v_{y0} * t - gt^2 \quad (6)$$

Write a program that reads the initial height y_0 and initial horizontal and vertical velocity components v_{x0} and v_{y0} of the projectile. These values will be assigned to suitable variables. The output from the program will be a display on the screen of the numerical values of x and y calculated using equations (5) and (6) at successive, small intervals of time. The program should terminate when y becomes zero again (i.e., the projectile falls back to earth). This display should be captured into a text file and a graph plotted of y versus x using Microsoft Excel. One way of capturing program output in a file is described in detail in the section on Command Line Redirection. Informally, the motion of the projectile is pretty simple: it moves steadily to the right (positive x direction), simultaneously going up for a while (positive y direction), and then coming down again to Earth. Technically, under the various idealizing assumptions we have made, it can be shown that the trajectory will be parabolic. You should test your program carefully. Explain, in your report, what tests you carried out, how the program behaves, and whether it has passed such tests.

Task 2 Functional Decomposition and File Writing (1/2 of the points)

Use functional decomposition to break your program into smaller, more manageable pieces. This will make it easier to write, test, and debug. For example, you might define a new function of your own, which deals with printing out the results, for one cycle of your calculations. This can then be simply called or invoked from within the while loop of your main() function - which should make the main() function shorter and easier to follow. Note that this function will have to accept two arguments (the current values for the x and y co-ordinates). At least three other functions, apart from main(), are required to be produced.

Two of these functions should be using file-related functions (that enable you to open, read, write and close a file) in order to write directly in a file with no output redirection involved.

Test the program again and record the results.

Laboratory Sessions 8 and 9

Introduction

Please note that only one submission at the end of session 9 is required.

Exercise 1: Car Registration (50%)

The file “indata.txt” contains a series of 6 lines, each containing pieces of information about a car, separated by space. In each case, the first number denotes the year of fabrication, the second text the car’s number in Irish format (year, county, number), the third piece of information is the color and the fourth – car’s engine volume in litres. You are required to develop a program which will read the data from this file into an array of structures of type `car_type` (see below), and for each of the cars calculate and output the corresponding tax to be paid using the following algorithm:

For cars no older than 5 years, the tax is Euro 150/year, if the engine is less and equal to 1.6 litres and Euro 300/year otherwise. For older cars there are three levels: less than 1.4 litres, less than 1.6 litres but greater than 1.4 litres and greater or equal to 1.6 litres. The tax is Euro 200 for the smallest engine cars, Euro 400 for the average engine size cars and Euro 600 for the highest engine size cars, respectively.

Test the program and record the test results.

Exercise 2: Car Taxation (50%)

Update the `car_type` structure to store the tax as well computed as above and write the output both on the screen and on a file using file-related functions. Use a user menu. Test the program by adding new lines to the “indata.txt” file and record the test results.

```
typedef struct car
{
    int year;
    char number[9];
    char colour[10];
    float engine;
} car_type;
```

“indata.txt”

```
1991 91D2134 red 1.8
1999 99D2939 blue 1.5
2003 03C2334 red 1.2
2006 06G1123 white 1.0
1998 98D5564 blue 1.6
2005 05D1354 green 1.3
```

Laboratory Session 10 – Mock Exam

Introduction

This is a computer-based exam simulation.

Exercise 1: Theoretical Question (30%)

Exercise 2: Complex Problem (70%)

Section 3

Laboratory Report Sample

LABORATORY 3 – REPORT

Student name: Brian Murphy
Student ID: 15565656
Programme: ECE

I hereby declare that the attached submission is all my own work, that it has not previously been submitted for assessment, and that I have not knowingly allowed it to be used by another student. I understand that deceiving or attempting to deceive examiners by passing off the work of another as one's own is not permitted. I also understand that using another's student's work or knowingly allowing another student to use my work is against the University regulations and that doing so will result in loss of marks and possible disciplinary proceedings.

Signed: Brian Murphy

Date: 1 February 2016

Note: Coursework examiners are entitled to reject any coursework which does not have a signed copy of this form attached or are submitted late.

Problem 1

The aim of this lab session is to write a C program that calculates the area of a circle, to compile the program and to test it. A radius value is read from the user and used to calculate the area of the circle. The result is printed to the standard output (on the screen). Several tests are run to check if the program performs the right computations for different input values.

Plan

Declare variables: radius, area
Print: welcome message
Input data: radius
Compute: $\text{area} = \text{PI} * \text{radius} * \text{radius}$
Output data: area
Print: goodbye message
Exit program

Development

The first step in the development of my program was to include the standard C libraries (stdio.h, stdlib.h, math.h) that in order to be able to use certain functions in the program.

I also declared the variables that I will use in the program, in this case two floats: radius (stores the radius of the circle) and area (stores the area of the circle). A constant variable called PI = 3.1415926 was also defined.

Then, I started to write the main function of the program.

The radius of a circle was requested from the keyboard using the “printf” statement and the value was read using the “scanf” statement. The area of the circle is calculated using this value. The equation for the calculation of the area of a circle is:

$$\text{Area} = \text{PI} * (\text{radius})^2$$

The result is then printed to the screen, using the “printf” statement.

Testing

I compiled and tested the program using the bcc32 compiler. I was presented with the following two errors:

Error E2379 area.c 11: Statement missing ; in function main

Error E2451 area.c 13: Undefined symbol 'radius2' in function main

Warning W8065 area.c 13: Call to function 'pow' with no prototype in function main

**** 2 errors in Compile ****

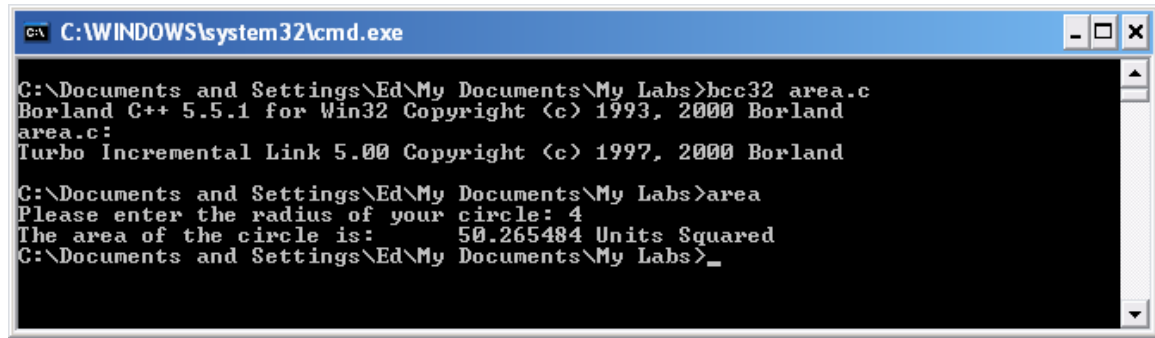
The above errors were corrected by inserting a “;” at the end of the printf statement on line 11. On line 13 an error was received due to incorrect use of the pow function. The program was then recompiled (no errors received) and executed for runtime testing.

During program execution, a runtime error was encountered after the input data was introduced. On examination of the code, it was discovered that “&” was omitted from the “scanf” statement, on line 12, causing the program to address a memory location that it did not have access to. The program was now recompiled and tested with different values for the radius of a circle. The results are presented next:

radius = 4.0	---- Area = 50.265
radius = 30.25	---- Area = 2912.89
radius = 0	---- Area = 0.00
radius = -4	---- Area = 50.265

The program is not protected against erroneous entries such as negative values (which make no geometrical sense), letters, etc.

The following print screen shows the programs output:



The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The user has navigated to the directory "C:\Documents and Settings\Ed\My Documents\My Labs" and compiled a file named "area.c" using the Borland C++ 5.5.1 compiler. The output shows the compilation was successful. Then, the user ran the "area" program, which prompted for the radius of a circle. The user entered "4", and the program outputted "The area of the circle is: 50.265484 Units Squared".

```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Ed\My Documents\My Labs>bcc32 area.c
Borland C++ 5.5.1 for Win32 Copyright (c) 1993, 2000 Borland
area.c:
Turbo Incremental Link 5.00 Copyright (c) 1997, 2000 Borland

C:\Documents and Settings\Ed\My Documents\My Labs>area
Please enter the radius of your circle: 4
The area of the circle is: 50.265484 Units Squared
C:\Documents and Settings\Ed\My Documents\My Labs>_
```

Conclusion

During this lab session I learned about some of the functions contained in the math.h library (e.g. pow). Also the errors I obtained helped me understand the difference between compile and runtime errors.

The final version of the C source code for exercise 1 is attached as *area.c* file

Code

My development resulted in the following source code:

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

#define PI 3.141592654

int main(void)
{
    /* declarations */
    float radius, area;

    /* input data */
    printf("Welcome to circle area calculation program!\n");
    printf("Please enter the radius of your circle:\t")
    scanf("%f", &radius);

    /* calculation */
    area = PI * pow(radius, 2);

    /* output */
    printf("\nThe area of the circle is:\t%f units squared", area);

    /* exit */
    printf("Goodbye!\n");
    return 0;
}
```

Section 4

Using Borland C++ Compiler

A step-by-step description on how to write a C program in a file (*filename.c*), to compile the program and to run the executable file is provided in this document.

Step 1: How to edit a *filename.c* file using TextPad application

Launch the TextPad text editor as follows:

Start > Programs > TextPad

Write the C code for your application in the opened file as presented in Figure 1.

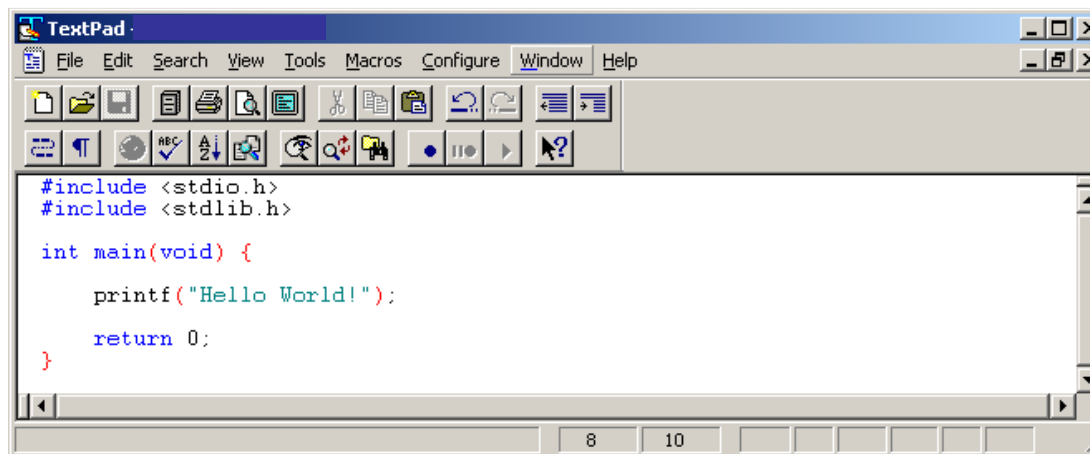


Figure 1. A file with C code lines that was edited using TextPad

Step 2: How to save the file

In order to save the data, choose *File-> Save* option and indicate the name of the file (e.g. *hello.c*) as well as the folder of your choice in your home directory (e.g. *h:\EM108*) (see Figure 2).

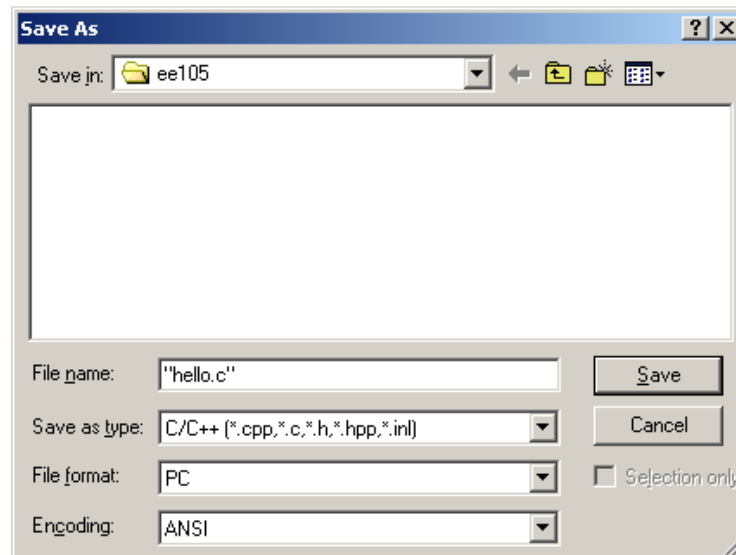


Figure 2. Save the data as *hello.c* file in the EM108 directory

Step 3: How to open a command prompt window for running the Borland C ++ Compiler

In order to compile and run the C source code from the *hello.c* file, you have to launch the Borland C++ compiler as follows:

Start > Programs > Borland C++ 5.5 > bcc55

A command prompt window will appear and will point to the temp directory on the C: drive (Figure 3).

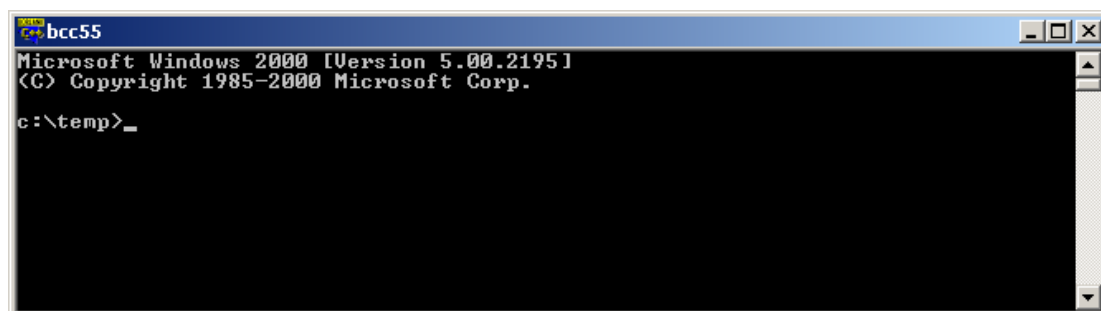
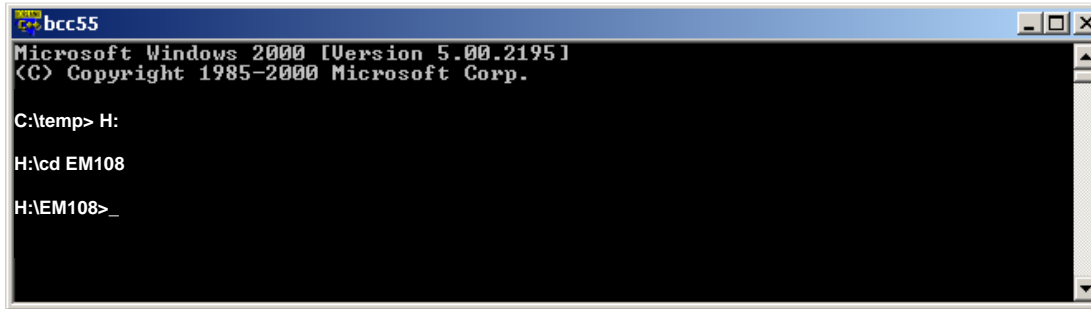


Figure 3. Command prompt window for the Borland C++ compiler

You must now navigate to the directory where you have saved your file as follows:

> *h:*
> *cd yourdirectory* (for example: > *cd EM108*)



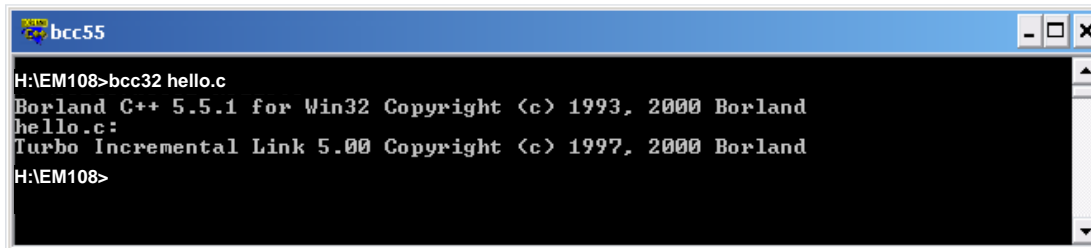
```
bcc55
Microsoft Windows 2000 [Version 5.00.2195]
Copyright 1985-2000 Microsoft Corp.

C:\temp> H:
H:\cd EM108
H:\EM108> _
```

Figure 4. Change the current location from *C:\temp* to *h:\EM108*

Step 5: How to compile the *hello.c* file using Borland C++ Compiler

You can compile a file with a C source code by typing “*bcc32 filename.c*” at the command prompt. This should result in the following output if everything has worked:



```
bcc55
H:\EM108>bcc32 hello.c
Borland C++ 5.5.1 for Win32 Copyright (c) 1993, 2000 Borland
hello.c:
Turbo Incremental Link 5.00 Copyright (c) 1997, 2000 Borland
H:\EM108>
```

Figure 5. Compile the *hello.c* file using the *bcc32* command

Step 6: How to run a C program

After the program was compiled and no errors were listed by the Borland C++ Compiler, an executable file is generated (e.g. *hello.exe*). This file is located in the same directory as the file with the source code (*hello.c*). You can run the executable file by typing *filename.exe*. (i.e. *hello.exe*) and the output of the program (if any) is listed on the screen.

NOTE:

For those students who want to install the BCC55 compiler on your laptops, please note it works on all operating systems, including Windows 7 and 8. Please download the software following the path given to you after filling this form: <http://cc.embarcadero.com/downloads.aspx?id=24778>

Install it. Please follow the advice on the next webpage to configure properly your installation: <http://rudeserver.com/how-to/install-borland/>

To help you run your compiler, you can create a text file, call it **bcc55.bat** on your desktop for example and type in it: **SET PATH=C:\borland\bcc55\bin;%PATH% C:\WINDOWS\SYSTEM32\CMD.EXE**

Save it. Every time you want to compile a **X.c** program of yours, run this **bcc55.bat** file, then go in the command line mode to your program directory (e.g. c:\EM108\lab1) and you will be able to compile your C program by typing **bcc55 X.c**

This example assumes you have maintained the original path for the installation **C:\borland\bcc55\bin** (otherwise please make the required path changes) and that your operating system is Windows. Similar settings can be done for Unix-like OS-es.

Section 5

Proposed Problems

Although these problems are optional, it is strongly recommended that you attempt to solve them. Remember the approach to problem solving: outline the solution first (flow chart), then code the program.

Problem Set No. 1

Problem 1.1: Write a program to calculate the hypotenuse of a right-angled triangle, where the lengths a and b of the triangle are 3 & 4 respectively. The program should calculate the length c of the hypotenuse given the formula: $c = \sqrt{a^2 + b^2}$. Print the value of c to the screen. Redo the problem by reading values for a and b . (*)

Problem 1.2: Write a program which takes the values of three resistors ($r_1=2000$ ohms, $r_2=1500$ ohms and $r_3=750$ ohms) and calculates their total resistance r_p when arranged in parallel and r_s when arranged in series. r_p is given by $r_p = 1/(1/r_1 + 1/r_2 + 1/r_3)$ and r_s is given by $r_s = r_1 + r_2 + r_3$. (**)

Problem 1.3: Write a program to calculate how tall is a building. To measure how tall is the building you go to the top floor and leave a stone to fall to the ground, while measuring the time it takes to reach the ground level. To increase the precision of your experiment you do this three times and take three readings of the time: t_1 , t_2 and t_3 , expressed in seconds. Calculate the height h given the equation $h = 1/2(g \cdot t^2)$ where $g=9.81$ m/s for Dublin and t is the average time. The program should print the result expressed in meters. Test the program. (***)

Problem 1.4: Write a program which reads three values a , b and c that are the coefficients of a quadratic equation $ax^2 + bx + c = 0$. Calculate the roots of the equation after you verify that $(b^2 - 4ac)$ is positive. What is the other condition you have to test for such as you can calculate the roots? The roots are given by $r = (-b \pm \sqrt{b^2 - 4ac})/2a$. Print either the roots or a message of error if you cannot calculate the roots. Test the program.

Note: you need to add the precompiler directive `#include <math.h>` at the start of your program to use the function `sqrt()`. (***)

Problem 1.5: Write a program that calculates the perimeter and the area of a rectangle. The program should read three pairs of integers (x_1, y_1) , (x_2, y_2) and (x_3, y_3) which represent three points of the rectangle. The program should display the perimeter and the area values in easily readable form. Use the Euclidean distance between the points: $\text{distance}(\text{Point0}, \text{Point1}) = \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$. Test the program.

Note: you need to add the precompiler directive `#include <math.h>` at the start of your program to use the function `sqrt()`. (***)

Problem Set No. 2

Problem 2.1: Write a program to calculate the cost of fencing a circular field. The program should read in two integers, d for the diameter of the field in meters and c for the cost of a twenty-meter roll of sheepwire. $\pi=3.1416$. The program should print in an easily readable form the number of rolls of sheepwire needed for the job and the cost of fencing the circular field.

(*)

Problem 2.2: Write a program which reads in the lengths in meters of the sides of a triangle l , m and n , calculate the perimeter and area of the triangle and determines the number of 2 litre tins of paint required to paint the area. A 2-litre paint can cover 20 square metres. The program should print in an easily readable form the number of tins requires.

(***)

Problem 2.3: Write a program to calculate the maturity value of a short-term loan at 20, 40 and 60 days. The program should read in two values, i.e. the amount of the principal value of the loan in Euro (p) and the percentage annual interest rate (i). The maturity value v is given by $v=p(1+i/100*n/365)$ where n is the number of days. The program should print the maturity value in Euro and punds for 20, 40and 60 days.

(***)

Problem 2.4: Write a program to calculate the cost of making a copper cylinder. The program should read in the radius of the cylinder, r , the height of the cylinder, h and the cost in Euro per square centimetres of copper sheeting, c . Assume that there is no wastage in the manufacturing process. The program should calculate the area of copper sheeting needed for the two circular ends of the cylinder and add the area needed for the body of the cylinder. $\pi=3.1416$. The program should print in an easily readable form the cost of the copper cylinder in Euro and punds.

(***)

Problem 2.5: Write a program to calculate the cost of a circular steel plate, which has to be cut from a square piece of steel. The diameter of the circle is equal to the side of the square. The program should read in the radius of the circle, r and the cost per square centimetre of steel plate in Euro, c . The square steel plate is purchased at cost c and the wastage can be returned to the supplier for 20% of the cost c for melting down. $\pi=3.1415$. The program should print in a easily readable form the cost of the circular steel plate.

(****)

Problem Set No. 3

Problem 3.1: Write a program that uses a loop to display the numbers from $a=1$ to $b=10$, together with the square and cube of each of their values. Redo the problem by reading values for a and b .

(*)

Problem 3.2: Write a program to display a multiplication table, such as young children use. The program reads a value n and then displays the results in a table like the one exemplified below for $n=4$. Test the program.

	1	2	3	4
1	1	2	3	4
2	2	4	6	8
3	3	6	9	12
4	4	8	12	16

(**)

Problem 3.3: Write a program to calculate the Fibonacci series of numbers, given the length of the series. The first two numbers in the series are 1 and 1. Following these, each number of the series is calculated as the sum of the previous two numbers. The program should read n – as the length of the series and to display the first n numbers from the Fibonacci series. Test the program.

(***)

Problem 3.4: Write a program to read two numbers n and s . The programs should verify that n divides exactly by s . Then the program should display all the verses of the following nursery rhyme, starting from number n that decreases with step s :

n green bottles, hanging on a wall,
 n green bottles, hanging on a wall,
If s green bottles were to accidentally fall,
There'd be $n-s$ green bottles hanging on the wall.

(***)

Problem 1.5: Write a program that implements a hand calculator with the following buttons: '+', '-', '*', '/', 'C', '='. It is supposed to repeatedly read float values, interleaved by characters (like somebody would have pressed one of the calculator's buttons). The program should either read another operand or display the result of that operation. Test the program.

(*****)

Problem Set No. 4

Problem 4.1: Write a program that reads in four integers a,b,c and d and then determines which is the biggest. Print the value of the biggest with an appropriate message.

(*)

Problem 4.2: Write a program to read in the sides of a triangle (l,m,n) and determine if it is a right angled triangle or not. If it is a right-angled triangle, then Pythagorus's theorem will hold for it. The program should display an appropriate message. Ask the user if s/he does not want to check another triangle and while so, repeat the procedure.

(**)

Problem 4.3: Write a program to read in three integers (1-100) and sort them in descending order. The program should print the values in descending order with appropriate messages. Repeat the problem while the user wants to. (***)

Problem 4.4: Write a program to read in any date in format: day, month, year. The day must be from 1 to 31, month from 1 to 12 and year must be a number from AD 1800 onwards. Determine whether the year is a leap year and print appropriate messages. If year is before 1800 or greater than 9999, print "out of range" message. A leap year is defined as a centenary year divisible by 400 or a non-centenary year divisible by 4. Check if the day number corresponds to the maximum number accepted in that month, that year (E.g. 28 or 29 in February, 30 in November, etc.), otherwise print an error message.

(****)

Problem 4.5: Write a program to calculate the commission for a sales department and print the commission in an easily readable form. The program should read in the sales figure as a real number and compute the commission as follows:

- On sales of up to 10000 Euro, a commission of 5% is paid.
- On sales over 10000 Euro and up to 20000 Euro, an additional 5% is paid on the amount exceeding 10000.
- On sales over 20000 an additional 0.25% is paid on the amount exceeding 20000 Euro. Provide a menu that would allow you to recompute the commission, to enter a new sales figure, or to quit.

(*****)

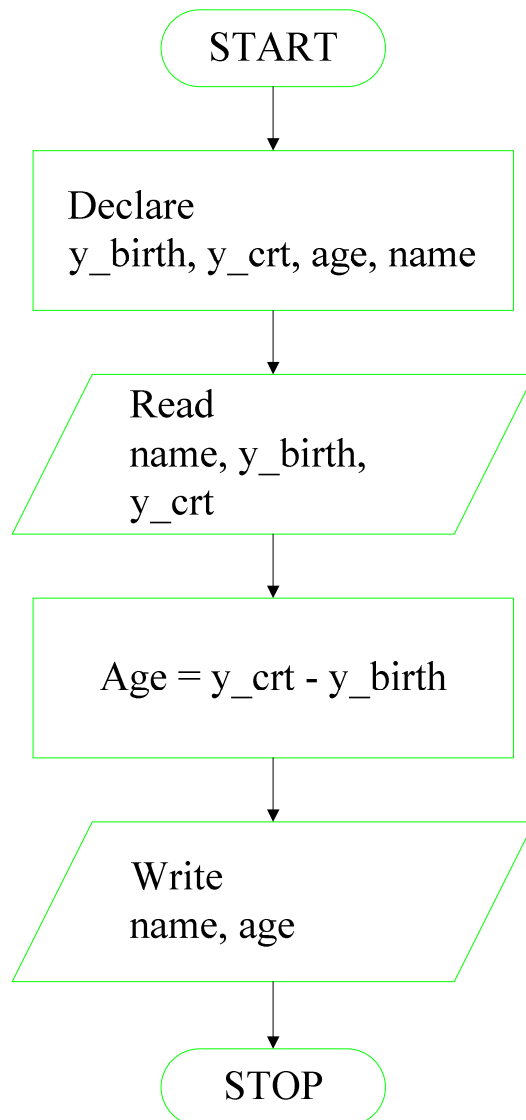
Section 6

Example Solutions

The following problems have been solved for you to have them as examples. It is strongly recommended that you compile and test them yourselves!

Problem No. 1

/* Problem that reads the year of birth, the current year and then computes and displays the age in years*/



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

int main()
{
    /* declarations */
    int y_birth, y_crt, age;
    char name [20];

    /* input */
    printf("Welcome to age program!\n");
    printf("Enter your name\n");
    scanf("%s", name);
    printf("Enter your year of birth");
    scanf("%d", &y_birth);
    printf("Enter the current year\n");
    scanf("%d", &y_crt);

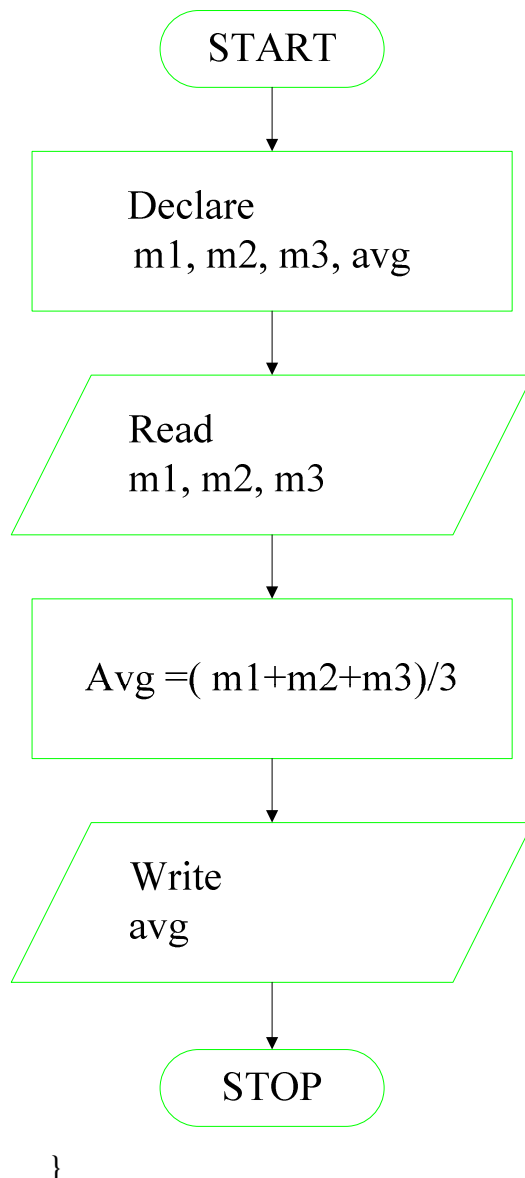
    /* computation */
    age = y_crt - y_birth;

    /* output */
    printf("%s was born in %d and is %d years old\n", name, y_birth, age);

    return (EXIT_SUCCESS);
}
```

Problem No. 2

/* Problem that reads marks for three modules, averages them and displays the final result*/



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

int main()
{
    /* declarations */
    float m1, m2, m3, avg;

    /* welcome message */
    printf("Welcome to the
    averaging program\n");

    /* input values */
    printf("Enter mark 1\n");
    scanf("%f", &m1);

    printf("Enter mark 2\n");
    scanf("%f", &m2);

    printf("Enter mark 3\n");
    scanf("%f", &m3);

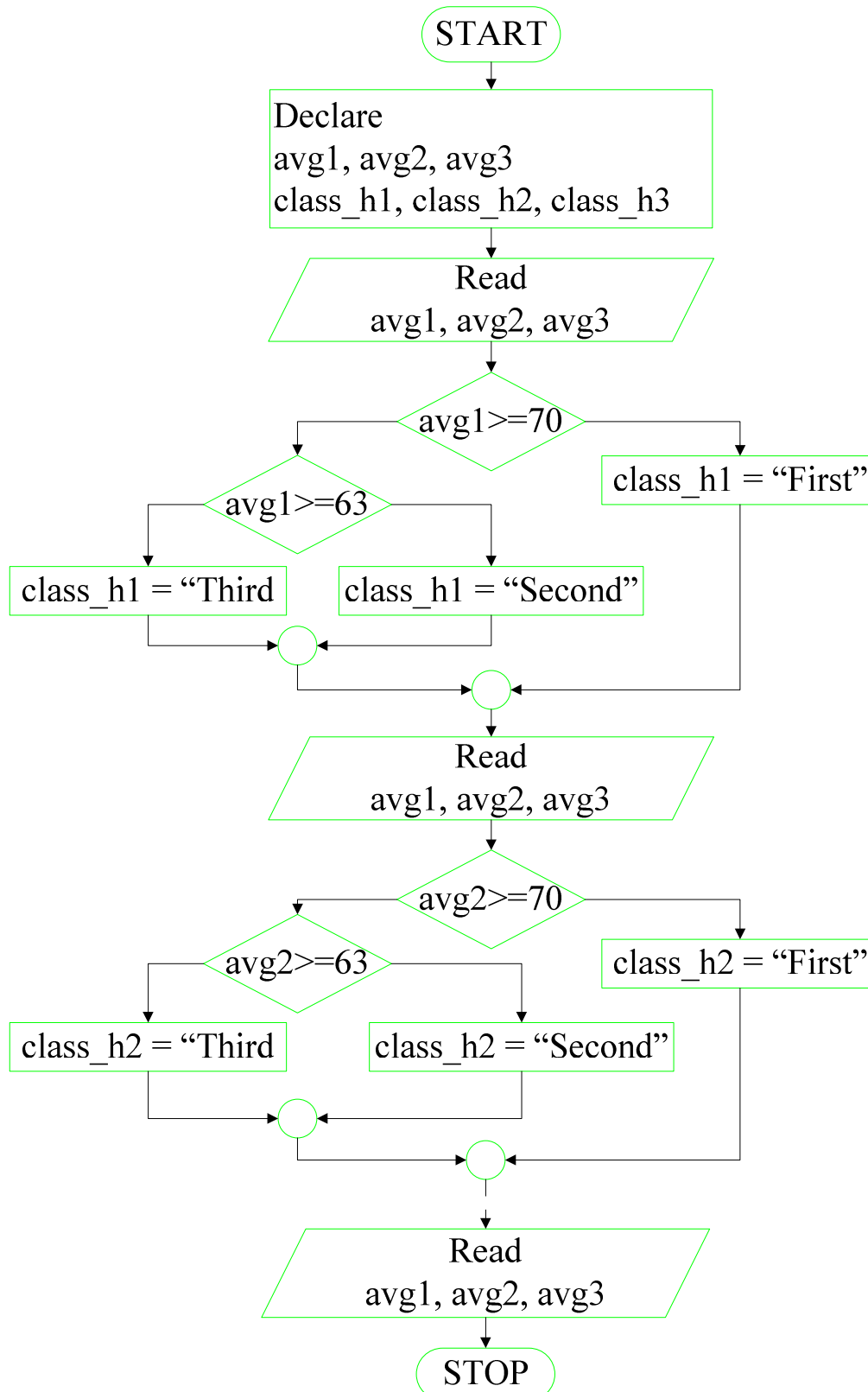
    /* computation */
    avg=(m1+m2+m3)/3;

    /* output */
    printf("The average of: %f, %f
    and %f is %f", m1, m2, m3, avg);

    /* return */
    return(EXIT_SUCCESS);
}
```

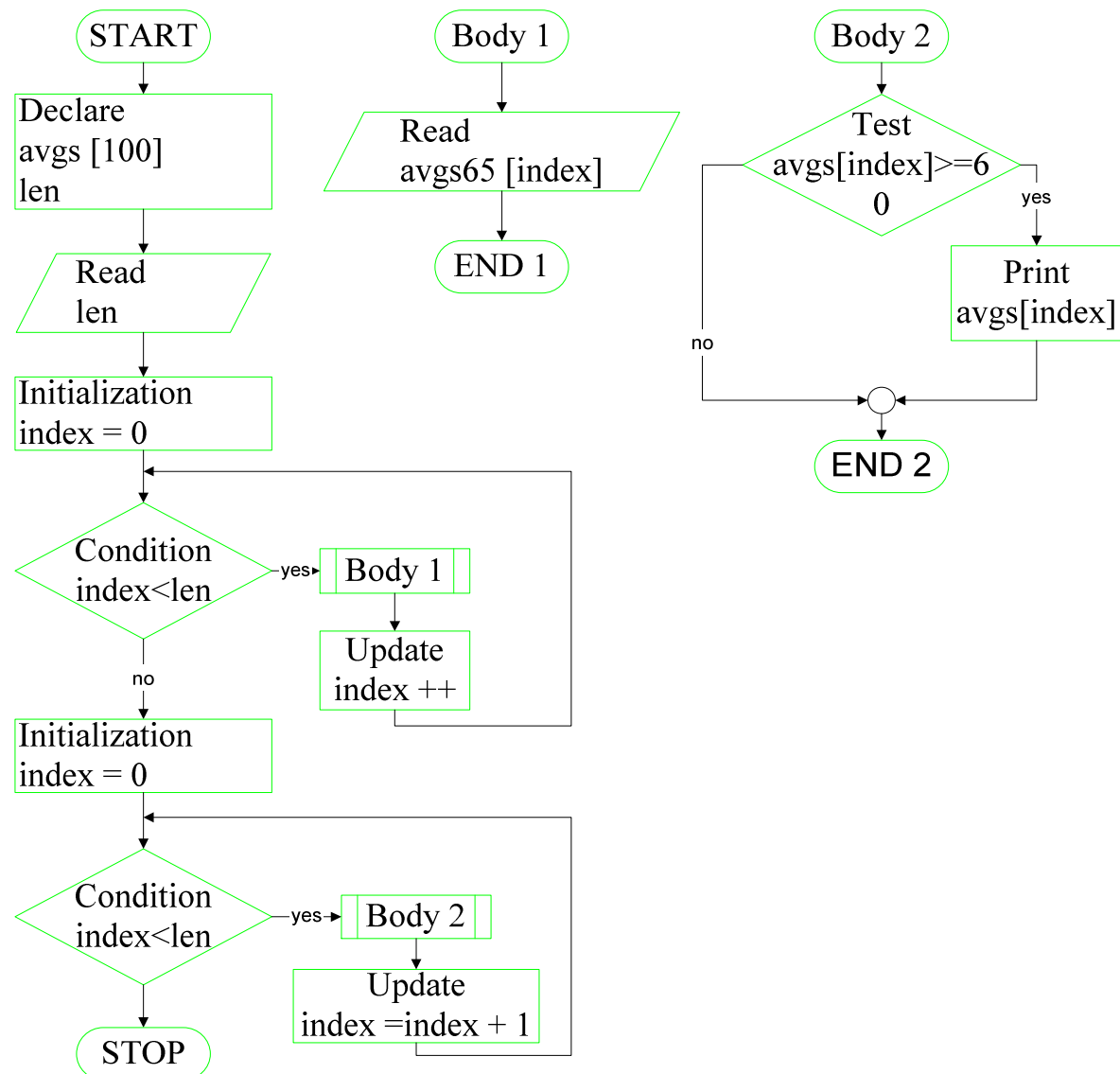
Problem No. 3

/* Program that classifies students according to their average marks in first, second or third class honours*/



Problem No. 4

/*Problem that reads the averages for all the students from a class, tests them for the admission limit (60) and prints only those values greater than this limit*/



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

int main()
{
    /* declarations */
    int avgs[100], len, index;

    /* input */
    printf("Welcome to averages!\n");

    printf("Enter the number of students\n");
    scanf("%d", &len);

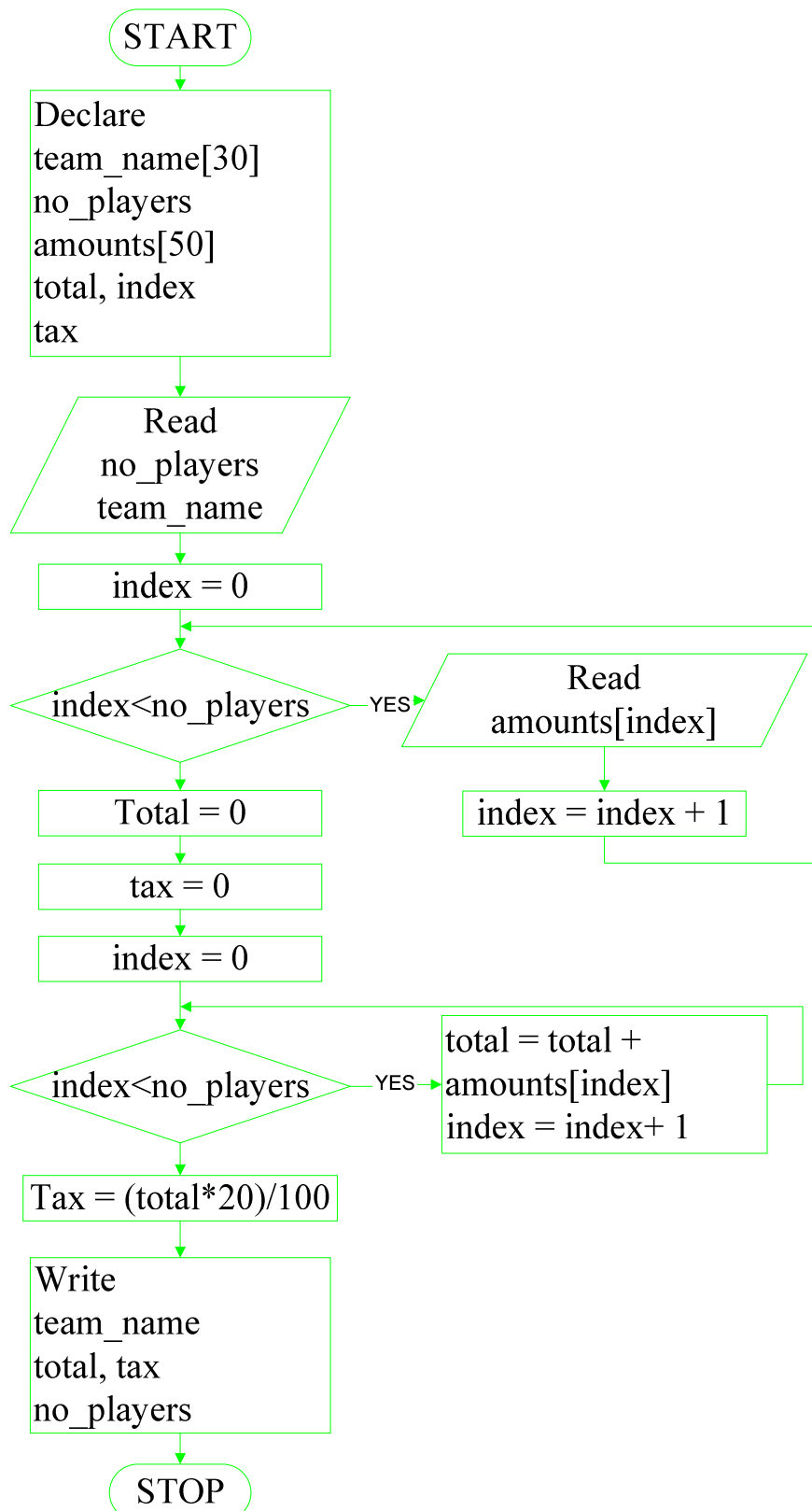
    for(index=0; index<len; index++)
    {
        printf("Enter average number: %d", index);
        scanf("%d", &avgs[index]);
    }

    /* output */
    for(index=0; index<len; index=index+1)
    {
        if(avgs[index]>=60)
        {
            printf("Value %d, has passed the admission limit\n",
                avgs[index]);
        }
        else
        {
            printf("Value %d is below 60, and therefore
                failed the admission limit\n", avgs[index]);
        }
    }

    return(EXIT_SUCCESS);
}
```


Problem No. 5

/* Problem that reads the number of players from a team and then the amounts paid to each player. It computes the total and prints it along with the tax to be paid at a rate of 20% */



```
#include <stdio.h>
#include <stdlib.h>
#define max_players 20

int main()
{
    /* declarations */
    char team_name [30];
    int no_players, total, tax, index, overall_cost;
    int amounts[max_players];

    /* input */
    printf("Enter the name of the team: ");
    scanf("%s", team_name);
    printf("Please enter the number of players: ");
    scanf("%d", &no_players);

    for(index=1; index<=no_players; index=index+1)
    {
        printf("Enter the amount for player %d: ", index);
        scanf("%d", &amounts[index]);
    }

    /* initialisations */
    total=0;
    tax=0;

    /* computations */
    for(index=1; index<=no_players; index=index+1)
    {
        total=total+amounts[index];
    }

    tax=total*20/100;
    overall_cost=total+tax;

    /* output */
    printf("Team: %s\nNumber of players: %d\nTotal\n\n", team_name, no_players);
    printf("cost: %d\nTax: %d\nOverall Cost: %d", team_name, no_players, total, tax, overall_cost);

    return(EXIT_SUCCESS);
}
```